Foundations for Decision Problems in Separation Logic with General Inductive Predicates

Timos Antonopoulos[1], Nikos Gorogiannis[2], Christoph Haase[3]*, Max Kanovich[4], and Joël Ouaknine[1]

[1] Department of Computer Science, University of Oxford, UK
[2] Department of Computer Science, Middlesex University London, UK
[3] LSV, CNRS & École Normale Supérieure (ENS) de Cachan, France
[4] Department of Computer Science, Queen Mary University of London, UK

Abstract. We establish foundational results on the computational complexity of deciding entailment in Separation Logic with general inductive predicates whose underlying base language allows for pure formulas, pointers and existentially quantified variables. We show that entailment is in general undecidable, and ExpTime-hard in a fragment recently shown to be decidable by Iosif *et al.* Moreover, entailment in the base language is Π_2^{P}-complete, the upper bound even holds in the presence of list predicates. We additionally show that entailment in essentially any fragment of Separation Logic allowing for general inductive predicates is intractable even when strong syntactic restrictions are imposed.

1 Introduction

Separation Logic (SL) is an extension of Hoare logic for reasoning about programs which manipulate heap data structures. Introduced in the early 2000s by O'Hearn, Ishtiaq, Reynolds and Yang [18,20], it has been the starting point of a line of research that has led to a large body of theoretical and practical work.

In the early days, the potential of Separation Logic was recognised by proving the (partial) correctness of the Schorr-Waite graph marking algorithm [22] and Cheney's copying garbage collector [6]. Those proofs were essentially carried out in a pen-and-paper fashion and demonstrated the strength of the paradigm underlying Separation Logic: *local reasoning.* The latter means that correctness proofs for heap-manipulating code should only depend on the portions of the heap accessed by the code and not the entire memory. Motivated by these positive results, research has been conducted on automating such proofs. On the one hand, support for Separation Logic has been integrated into proof assistants such as Coq, enabling semi-automated verification of program code, see *e.g.* [2]. On the other hand, a number of fully-automatic tools such as SmallFoot, SLAyer, Space Invader or SLAD have been developed and successfully used to show absence of memory errors in low-level real-world code, see *e.g.* [11,7,5].

* Supported by the French ANR, ReacHard (grant ANR-11-BS02-001).

A crucial requirement for any of these tools is the ability to check applications of the consequence rule in Hoare logic, as it is this rule that underpins most methods of proof based on Separation Logic. The consequence rule, in turn, requires the ability to check entailment between Separation Logic formulas. However, entailment checking in full Separation Logic is undecidable [12,10], thus these tools have to work with restricted, decidable fragments. Decidability, though, comes at the cost of reduced expressive power and, often, reduced generality. For instance, the fragment used by SMALLFOOT allows for reasoning about memory shapes built upon the hard-coded primitives of pointers and linked lists, essentially limiting its applicability to programs only involving those data structures; some efforts have been made in order to allow for reasoning about more generic list data structures, see *e.g.* [3]. The limitations of hard-coded inductive predicates have been realised by the community, and recent research has been conducted to enable automated reasoning about generic user-defined inductive predicates, inside the framework of Separation Logic, see *e.g.* [13,9], or in related frameworks such as forest automata [16]. Notable recent progress has been made by Iosif *et al.* who showed decidability of satisfiability and entailment for a syntactic fragment of Separation Logic with general recursively defined predicates by establishing a reduction to Monadic Second Order Logic on graphs with bounded tree width [17]. Finally, Brotherston *et al.* have developed an EXPTIME-complete decision procedure for satisfiability of Separation Logic with general inductively defined predicates [8]. In the same paper it is shown that the problem becomes NP-complete if the arity of all predicates is bounded by a constant.

The goal of this paper is to contribute to this line of research and to establish foundational results on the inherent computational complexity of reasoning problems in Separation Logic with general inductively defined predicates. In order to obtain meaningful lower bounds, we restrict our analysis to the most basic syntactic fragment of Separation Logic comprising (positive) Boolean combinations of judgments on stack variables, both fixed and existentially quantified, and pointers. This fragment also forms the basis of the decidable fragments of Separation Logic from [17,8]. Standard inductive data types are inductively expressible in this fragment, for instance singly-linked lists as used by SMALLFOOT [4] can be defined as follows:

$$\mathsf{ls}(\mathsf{a},\mathsf{b}) := \mathsf{emp} \wedge \mathsf{a} = \mathsf{b} \mid \exists \mathsf{c}.\ \mathsf{pt}(\mathsf{a},\mathsf{c}) * \mathsf{ls}(\mathsf{c},\mathsf{b}) \wedge \mathsf{a} \neq \mathsf{b} \tag{1}$$

Informally speaking, supposing that $\mathsf{ls}(\mathsf{x},\mathsf{y})$ holds in a memory model, this definition states that there is a singly-linked list segment from the memory cell labeled with the stack variable x to the memory cell labeled with y if either the heap is empty and x is equal to y, or x is not equal to y and the heap can be split into two disjoint parts, indicated by the $*$-conjunction, such that on the first part x is allocated and points to some cell a and heap cell a is the starting point of a singly-linked list segment ending in y in the other part.

The main results of our paper are as follows. In the first part, we consider entailment in Separation Logic with general inductive predicates. Given two assertions α, α' and a finite set of inductive predicates P referred to by α and α', entailment is to decide whether the set of memory models of α is contained in the set of memory models of α' with respect to P. We show that this problem is undecidable in general and EXPTIME-hard when restricted to the decidable syntactic fragment defined by Iosif *et al.* In the second part, we take a closer look at entailment in the basic fragment of Separation Logic in the absence of inductive predicates, *i.e.*, Separation Logic with positive pure formulas, existentially quantified variables and pointers. We show that this problem is complete for Π_2^{P}, the second level of the polynomial hierarchy. The upper bound also holds when allowing for the above list predicate hard-coded in the syntax. Subsequently, we analyse the Π_2^{P} lower bound and define a natural syntactic fragment for which entailment is decidable in polynomial time, yet NP-hard in the presence of a list predicate, *i.e.*, one of the simplest possible inductive predicates. We discuss the results obtained in the conclusion at the end of the paper.

Some proofs have been omitted due to lack of space, but are included in a longer, online version of the paper, obtainable from the authors' webpages.

2 Preliminaries

Let X and Y be sets, and let $R \subseteq X \times Y$. We say that R is *functional* if for every $x \in X$ there is at most one $y \in Y$ with $(x, y) \in R$. Let Y be a countable, possibly infinite, set. We write $X \subset_{\text{fin}} Y$ if X is a finite subset of Y. Moreover, given countable sets X, Y, we write $f : X \rightharpoonup_{\text{fin}} Y$ if f is a function whose domain is a finite subset of X and its co-domain is Y. Given $f : X \to Y$, $x \in X$, $y \in Y$, we write $f[x \mapsto y]$ to denote the function f' such that $f'(z) \stackrel{\text{def}}{=} y$ if $z = x$, and $f'(z) \stackrel{\text{def}}{=} f(z)$ otherwise. Finally, given $i \leq j \in \mathbb{N}$, we write $[i, j]$ to denote the set $\{i, \ldots, j\} \subseteq \mathbb{N}$ and $[i]$ as an abbreviation for $[1, i]$.

Graphs. Let L be a countable set of *labels*. We define *directed labeled graphs* (just *graphs* in the following) as tuples $G = (V, E, \ell)$, where V denotes the set of *nodes* or *vertices*, E is a subset of $V \times V$, and $\ell : L \rightharpoonup_{\text{fin}} V$ is a *labeling function*. If L is empty we omit ℓ and just write $G = (V, E)$. A graph $G = (V, E, \ell)$ is *undirected* if E is symmetric. If G is a graph, we also denote its set of nodes by $V(G)$ and its set of edges by $E(G)$. The *size of* G is defined as $|G| \stackrel{\text{def}}{=} |V(G)|$.

For interpretations below, we require a slightly more general class of graphs which we call selector graphs, inspired by [17]. A *selector graph* is a tuple $G = (V, E, \ell, s)$, where V and ℓ are as above, $s : V \to \mathbb{N}$ assigns an *arity* to each vertex, and $E : V \times \mathbb{N} \to V$ is a partial function such that E is defined precisely for every pair (v, i) with $v \in V$ and $i \in [s(v)]$. If $s(v) \in \{0, 1\}$ for all $v \in V$, we obtain partial functional graphs.

Formulas of Separation Logic. The subsequent definitions are partly adapted or inspired from [17]. Let Vars be a totally ordered countably infinite set of *variable names*, which is partitioned into disjoint infinite sets EVars and FVars representing sets of *existential variables* and *fixed stack variables*, respectively. We will usually use $\mathsf{a}, \mathsf{b}, \mathsf{c}$ for elements from EVars, and $\mathsf{x}, \mathsf{y}, \mathsf{z}$ will usually be elements from Vars. Variables in FVars will be used to represent fixed stack variables. The purpose of the distinction between EVars and FVars is to help the reader to easily identify in which context a variable occurs. Let PNames be a finite set of *predicate names*, where each predicate has an associated arity $k \in \mathbb{N}$, and is written as $\mathsf{pred}(\mathsf{a}_1, \ldots, \mathsf{a}_k)$ with $\mathsf{a}_i \in \mathsf{EVars}$ for $i \in [k]$. The syntax of *SL-assertions or SL-formulas over* PNames is given by the following grammar, where $\mathsf{x}, \mathsf{x}_1, \ldots, \mathsf{x}_m, \mathsf{y}, \mathsf{y}_1, \ldots, \mathsf{y}_k \in \mathsf{Vars}$, $\mathsf{a}_1, \ldots, \mathsf{a}_n \in \mathsf{EVars}$, $m \geq 1$ and $n \geq 0$:

$$\begin{aligned}
\varphi &::= \top \mid \bot \mid \mathsf{x} = \mathsf{y} \mid \neg\varphi \mid \varphi \wedge \varphi && \text{(pure formulas)}\\
\sigma &::= \mathsf{emp} \mid \mathsf{tt} \mid \mathsf{pt}(\mathsf{x}, (\mathsf{x}_1, \ldots, \mathsf{x}_m)) \mid \mathsf{pred}(\mathsf{y}_1, \ldots, \mathsf{y}_k) \mid \sigma * \sigma \mid \alpha && \text{(spatial formulas)}\\
\alpha &::= \exists \mathsf{a}_1, \ldots, \mathsf{a}_n . \sigma \wedge \varphi && \text{(SL-assertions)}
\end{aligned}$$

Here, $\mathsf{pt}(\mathsf{x}, \mathsf{y})$ is the *points-to* or *pointer predicate* of an arbitrary arity m, and the $*$-conjunction is commutative, *i.e.*, SL-assertions are considered equivalent up to permutations of $*$-connected subformulas. We say that the SL-assertion $\alpha = \exists \mathsf{a}_1, \ldots, \mathsf{a}_n . \sigma \wedge \varphi$ is *flat* if σ contains no SL-assertion α' as a subformula. Given an SL-assertion $\alpha = \exists \mathsf{a}_1, \ldots, \mathsf{a}_m . (\sigma * \exists \mathsf{b}_1, \ldots, \mathsf{b}_n . (\sigma' \wedge \varphi')) \wedge \varphi$, and supposing without loss of generality that $\{\mathsf{a}_i : i \in [m]\} \cap \{\mathsf{b}_j : j \in [n]\} = \emptyset$, we can exhaustively rewrite the formula α as $\exists \mathsf{a}_1, \ldots, \mathsf{a}_m, \mathsf{b}_1, \ldots, \mathsf{b}_n . (\sigma * \sigma') \wedge \varphi \wedge \varphi'$. Thus we may assume with no loss of generality that an SL-assertion is flat.

Remark 1. We have imposed flatness and $*$-commutativity as syntactic properties. This is merely for presentational convenience in order to save space, as these properties follow from the semantics definition below.

We call φ *positive* if φ is a conjunction of literals $\mathsf{x} = \mathsf{y}$ and $\mathsf{x} \neq \mathsf{y}$. Moreover, we say that α is positive if φ is positive, and that α is *reduced* if no $\mathsf{pred}(\mathsf{y}_1, \ldots, \mathsf{y}_k)$ occurs in σ. By $vars(\alpha) \subseteq \mathsf{Vars}$ we denote the *set of all variables occurring in* α. The *size of* α, denoted by $|\alpha|$, is defined to be the number of symbols in α.

A *set P of inductive predicates over* PNames is a finite set of *definitions*

$$\mathsf{pred}(\mathsf{a}_1, \ldots, \mathsf{a}_k) := \alpha_1 \mid \cdots \mid \alpha_m$$

such that each $\mathsf{a}_i \in \mathsf{EVars}$, α_i is a flat SL-assertion $\alpha_i = \exists \mathsf{b}_1, \ldots, \mathsf{b}_n . \sigma \wedge \varphi$ over PNames such that $\{\mathsf{a}_i : i \in [m]\} \cap \{\mathsf{b}_j : j \in [n]\} = \emptyset$, and each predicate name $\mathsf{pred}(\mathsf{a}_1, \ldots, \mathsf{a}_k)$ occurs exactly once on the left-hand side of a definition in P. Moreover, we require that each α_i has no unbounded existential variables, *i.e.*, for each α_m as above, $vars(\alpha_m) \subseteq \mathsf{FVars} \cup \{\mathsf{a}_i, \mathsf{b}_j : i \in [k], j \in [n]\}$.[5] Given $\mathsf{x}_1, \ldots, \mathsf{x}_k \in \mathsf{Vars}$, define $\mathsf{pred}[\mathsf{x}_1/\mathsf{a}_1, \ldots, \mathsf{x}_k/\mathsf{a}_k] \overset{\text{def}}{=} \{\alpha_i[\mathsf{x}_1/\mathsf{a}_1, \ldots, \mathsf{x}_k/\mathsf{a}_k] : i \in [k]\}$, where $\alpha_i[\mathsf{x}_1/\mathsf{a}_1, \ldots, \mathsf{x}_k/\mathsf{a}_k]$ is obtained from α_i by replacing each occurrence of a_j with x_j for $j \in [k]$. Given a flat assertion $\alpha = \exists \mathsf{c}_1, \ldots \mathsf{c}_p . \sigma \wedge \varphi$, an *unraveling of* α *with respect to P* is obtained by replacing each $\mathsf{pred}(\mathsf{x}_1, \ldots, \mathsf{x}_k)$ occurring as a subformula in σ with some $\alpha' \in \mathsf{pred}[\mathsf{x}_1/\mathsf{a}_1, \ldots, \mathsf{x}_k/\mathsf{a}_k]$. We write $\alpha \to_P \beta$ if β is an unraveling of α with respect to P, and denote the reflexive transitive closure of $\to_P$ by $\to_P^*$. An unraveling $\alpha \to_P^* \beta$ is *complete* if no $\mathsf{pred}(\mathsf{x}_1, \ldots, \mathsf{x}_k)$ occurs in β.

[5] In a slight departure from convention, for presentational convenience we allow free variables to appear in the body of definitions; such predicates can always be transformed to equivalent ones where the previously free variables are parameters, w.l.o.g.

Example 2. Taking P to be the singleton set consisting of the definition of $\mathsf{ls}(\mathsf{a},\mathsf{b})$ from Equation (1), we have

$$\mathsf{ls}(\mathsf{x},\mathsf{y}) \rightarrow^*_P \exists \mathsf{c}_1,\mathsf{c}_2.\mathsf{pt}(\mathsf{x},\mathsf{c}_1) * \mathsf{pt}(\mathsf{c}_1,\mathsf{c}_2) * \mathsf{ls}(\mathsf{c}_2,\mathsf{y}) \wedge \mathsf{x} \neq \mathsf{y} \wedge \mathsf{c}_1 \neq \mathsf{y}$$

with respect to P, which is *not* a complete unraveling. A complete unraveling is $\mathsf{ls}(\mathsf{x},\mathsf{y}) \rightarrow^*_P \exists \mathsf{c}.\mathsf{pt}(\mathsf{x},\mathsf{c}) * \mathsf{emp} \wedge \mathsf{x} \neq \mathsf{y} \wedge \mathsf{c} = \mathsf{y}$.

For convenience, we sometimes use a generalised $*$-conjunction and, given spatial formulas $\sigma_1,\ldots,\sigma_n$, write $\mathop{\Large *}_{1\leq i\leq n} \sigma_i$ for $\sigma_1 * \cdots * \sigma_n$. Likewise, we write $\mathsf{pred}(\mathsf{a}_1,\ldots,\mathsf{a}_k) := \|_{i\in[n]}\alpha_i$ for $\mathsf{pred}(\mathsf{a}_1,\ldots,\mathsf{a}_k) := \alpha_1 \mid \cdots \mid \alpha_n$. Moreover, $\exists \mathsf{a}_1,\ldots,\mathsf{a}_n.\sigma$ abbreviates $\exists \mathsf{a}_1,\ldots,\mathsf{a}_n.\sigma \wedge \top$; we may also write *e.g.* $\mathsf{pt}(\mathsf{x},(\_,\mathsf{y}))$ as a shorthand for $\exists \mathsf{a}.\mathsf{pt}(\mathsf{x},(\mathsf{a},\mathsf{y}))$, where $\mathsf{a} \in \mathsf{EVars}$ is a fresh existential variable. Furthermore, $\exists_{i\in[n]}\mathsf{a}_i.\sigma\wedge\varphi$ abbreviates $\exists \mathsf{a}_1,\ldots,\mathsf{a}_n.\sigma\wedge\varphi$. If PNames is clear from the context we will omit stating it explicitly.

Interpretations. As stated above, interpretations are given in terms of selector graphs. This diversion from the more commonly found "heap-and-stack model" found in the literature is for technical convenience only, and it is easy to translate between the two interpretation domains.

An *SL-interpretation*, or simply *interpretation*, $\mathcal{I}$ is a selector graph $\mathcal{I} = (V^{\mathcal{I}}, E^{\mathcal{I}}, \ell^{\mathcal{I}}, s^{\mathcal{I}})$ such that $\ell^{\mathcal{I}} : \mathsf{FVars} \rightharpoonup_{\text{fin}} V$. For $\mathsf{x}_1,\ldots,\mathsf{x}_n \in \mathsf{FVars}$ and for $v_1,\ldots,v_n \in V^{\mathcal{I}}$, we denote by $\mathcal{I}[\mathsf{x}_1 \mapsto v_1;\ldots;\mathsf{x}_n \mapsto v_n]$ the SL-interpretation $\hat{\mathcal{I}} = (V^{\mathcal{I}}, E^{\mathcal{I}}, \ell^{\hat{\mathcal{I}}}, s^{\mathcal{I}})$, where $\ell^{\hat{\mathcal{I}}} \overset{\text{def}}{=} \ell^{\mathcal{I}}[\mathsf{x}_1 \mapsto v_1;\ldots;\mathsf{x}_n \mapsto v_n]$, and we call such an interpretation an *extension* of $\mathcal{I}$.

In our interpretations, nodes with arity greater than zero are the equivalent to allocated heap cells in the "heap-and-stack model", while record fields are represented by the different selectors. We define the $*$-decomposition of $\mathcal{I}$ as follows: $\mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2$ iff $\mathcal{I}_1 = (V^{\mathcal{I}_1}, E^{\mathcal{I}_1}, \ell^{\mathcal{I}_1}, s^{\mathcal{I}_1})$ and $\mathcal{I}_2 = (V^{\mathcal{I}_2}, E^{\mathcal{I}_2}, \ell^{\mathcal{I}_2}, s^{\mathcal{I}_2})$ such that $V^{\mathcal{I}} = V^{\mathcal{I}_1} = V^{\mathcal{I}_2}$; $\ell^{\mathcal{I}}$, $\ell^{\mathcal{I}_1}$ and $\ell^{\mathcal{I}_2}$ coincide; for $i \in \{1,2\}$, either $s^{\mathcal{I}_i}(v) = 0$ or $s^{\mathcal{I}_i}(v) = s^{\mathcal{I}}(v)$, and $s^{\mathcal{I}}(v) = s^{\mathcal{I}_1}(v) + s^{\mathcal{I}_2}(v)$ for all $v \in V^{\mathcal{I}}$; for $i \in \{1,2\}$, if $s^{\mathcal{I}_i}(v) > 0$ then $E^{\mathcal{I}_i}(v,j) = E^{\mathcal{I}}(v,j)$ for all $v \in V^{\mathcal{I}}$ and $j \in [s^{\mathcal{I}_i}(v)]$.

Semantics of SL-assertions. The semantics of flat reduced SL-assertions is defined by structural induction. Let $\mathcal{I} = (V^{\mathcal{I}}, E^{\mathcal{I}}, \ell^{\mathcal{I}}, s^{\mathcal{I}})$ be an SL-interpretation and φ a pure formula only over variable names from FVars, the *satisfaction relation* $\mathcal{I} \models \varphi$ is defined such that $\mathcal{I} \models \top$ holds always, $\mathcal{I} \models \bot$ never holds, $\mathcal{I} \models \mathsf{x} = \mathsf{y}$ iff $\ell^{\mathcal{I}}(\mathsf{x}) = \ell^{\mathcal{I}}(\mathsf{y})$, $\mathcal{I} \models \neg\varphi$ iff $\mathcal{I} \not\models \varphi$, and $\mathcal{I} \models \varphi_1 \wedge \varphi_2$ iff $\mathcal{I} \models \varphi_1$ and $\mathcal{I} \models \varphi_2$.

For reduced flat spatial formulas σ such that $\mathit{vars}(\sigma) \subseteq \mathsf{FVars}$, we define $\mathcal{I} \models \mathsf{emp}$ iff $s^{\mathcal{I}}(v) = 0$ for all $v \in V^{\mathcal{I}}$ and $\mathcal{I} \models \mathsf{tt}$ holds always. Moreover, $\mathcal{I} \models \sigma_1 * \sigma_2$ iff $\mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2$ such that $\mathcal{I}_1 \models \sigma_1$ and $\mathcal{I}_2 \models \sigma_2$, and finally, $\mathcal{I} \models \mathsf{pt}(\mathsf{x},(\mathsf{x}_1,\ldots,\mathsf{x}_m))$ iff

- $v = \ell^{\mathcal{I}}(\mathsf{x})$, $s^{\mathcal{I}}(v) = m$, $s^{\mathcal{I}}(v') = 0$ for all $v' \in V^{\mathcal{I}} \setminus v$; and
- $E^{\mathcal{I}}(v,i) = \ell^{\mathcal{I}}(\mathsf{x}_i)$ for all $i \in [m]$.

For a flat reduced SL-assertion $\alpha = \exists \mathsf{a}_1,\ldots,\mathsf{a}_n.\sigma\wedge\varphi$, we define $\mathcal{I} \models \alpha$ iff there is an extension $\hat{\mathcal{I}} = \mathcal{I}[\mathsf{x}_1 \mapsto v_1,\ldots,\mathsf{x}_n \mapsto v_n]$ for fresh variables $\mathsf{x}_1,\ldots,\mathsf{x}_n \in \mathsf{FVars}$

such that $\hat{\mathcal{I}} \models \sigma[\mathsf{x}_1/\mathsf{a}_1,\ldots,\mathsf{x}_n/\mathsf{a}_n]$ and $\hat{\mathcal{I}} \models \varphi[\mathsf{x}_1/\mathsf{a}_1,\ldots,\mathsf{x}_n/\mathsf{a}_n]$. We call $\mathcal{I}$ a *model* of α if $\mathcal{I} \models \alpha$. Given α and a set of inductive predicates P, we write $\mathcal{I} \models_P \alpha$ if $\mathcal{I} \models \alpha'$ for some α' obtained from a complete unraveling $\alpha \rightarrow_P^* \alpha'$. Given α and α' over a set of inductive predicates P, we write $\alpha \models_P \alpha'$ iff whenever $\mathcal{I} \models_P \alpha$ then $\mathcal{I} \models_P \alpha'$. Given α over a set of inductive predicates P, *satisfiability* is to decide whether there is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models_P \alpha$. Given $\mathcal{I}$, *model checking* is to decide $\mathcal{I} \models_P \alpha$. The main decision problem of interest in this paper is entailment, defined as follows.

<u>ENTAILMENT</u>

INPUT: SL assertions α, α' with respect to a set P of inductive predicates.
QUESTION: Does $\alpha \models_P \alpha'$?

3 Entailment in the Presence of Inductive Predicates.

In this section, we show that entailment with general inductive predicates is undecidable when no restrictions are imposed. Subsequently, we give an EXPTIME lower bound for the fragment introduced by Iosif *et al.* [17].

General undecidability. We show undecidability via a reduction from the undecidable Post Correspondence Problem [19].

<u>POST CORRESPONDENCE PROBLEM (PCP)</u>

INPUT: A finite set of tiles $(v_1,w_1),\ldots,(v_k,w_k)$, $v_i,w_i \in \{0,1\}^*$.
QUESTION: Does there exist a sequence $s_1 s_2 \cdots s_\ell \in \{1,\ldots,k\}^\ell$, $\ell > 0$ such that $v_{s_1} v_{s_2} \cdots v_{s_\ell} = w_{s_1} w_{s_2} \cdots w_{s_\ell}$?

For any $u \in \{0,1\}^*$, denote by $|u|$ the length of each tile, and by $u(i)$ the i-th symbol of u, for $1 \le i \le |u|$. For example, if $u = 01101$, we have $|u| = 5$ and $u(3) = 1$. Let $(v_1,w_1),\ldots,(v_k,w_k)$ be an instance of PCP. The set of predicates P in Figure 1 establishes a reduction such that this instance has a solution iff $\mathsf{PCP}(\mathsf{x},\mathsf{y}) \wedge \mathsf{x}_0 \neq \mathsf{x}_1 \not\models_P \overline{\mathsf{PCP}}(\mathsf{x},\mathsf{y})$. The intuition behind these definitions is as follows. For $\mathsf{x},\mathsf{y} \in \mathsf{FVars}$, $\mathsf{PCP}(\mathsf{x},\mathsf{y})$ generates the set of all possible tilings for a given instance: in any model the v_i-tilings begin at x and the w_i-tilings at y. The fixed stack variables $\mathsf{x}_0,\mathsf{x}_1 \in \mathsf{FVars}$ are used to represent the corresponding symbols 0 and 1. Likewise, $\overline{\mathsf{PCP}}(\mathsf{x},\mathsf{y})$ generates all tilings which are incorrect. This is the case if the model is empty, there are two symbols at the same position which are different (*cf.* $\mathsf{NEqualPair}(\mathsf{x},\mathsf{y})$), or the length of the strings encoded in a model is different (*cf.* $\mathsf{NEqualLen}(\mathsf{x},\mathsf{y})$).

Theorem 3. *Entailment in Separation Logic with general inductive predicates is undecidable.*

Remark 4. An anonymous referee remarked that our reduction can also be applied for showing that satisfiability in the presence of conjunction over spatial formulas and general inductive predicates is undecidable. The models of the subsequent predicate encode all pairs of equal strings:

$$\mathsf{EqPairs}(\mathsf{a},\mathsf{b}) = \mathsf{emp} \mid \|_{i\in\{0,1\}} \exists \mathsf{p},\mathsf{r}.\mathsf{pt}(\mathsf{x},(\mathsf{x_i},\mathsf{p})) * \mathsf{pt}(\mathsf{y},(\mathsf{x_i},\mathsf{r})) * \mathsf{EqPairs}(\mathsf{p},\mathsf{r}).$$

$$\begin{aligned}
\mathsf{PCP}(\mathsf{a},\mathsf{b}) &:= \mathsf{emp} \mid \mathsf{Tile}_1(\mathsf{a},\mathsf{b}) \mid \cdots \mid \mathsf{Tile}_k(\mathsf{a},\mathsf{b})\\
\mathsf{Tile}_i(\mathsf{a},\mathsf{b}), i\in[k] &:= \exists \mathsf{p}_0,\dots \mathsf{p}_{|v_i|}, \mathsf{r}_0,\dots \mathsf{r}_{|w_i|}. \mathop{\ast}_{0\le j<|v_i|} \mathsf{pt}(\mathsf{p}_j,(\mathsf{x}_{v_i(j+1)},\mathsf{p}_{j+1})) \ast\\
&\quad \ast \mathop{\ast}_{0\le j<|w_i|} \mathsf{pt}(\mathsf{r}_j,(\mathsf{x}_{w_i(j+1)},\mathsf{r}_{j+1})) \ast \mathsf{PCP}(\mathsf{p}_{|v_i|},\mathsf{r}_{|w_i|}) \wedge \mathsf{a}=\mathsf{p}_0 \wedge \mathsf{b}=\mathsf{r}_0\\
\mathsf{NEqualPair}(\mathsf{a},\mathsf{b}) &:= \exists \mathsf{p},\mathsf{r}.\mathsf{pt}(\mathsf{a},(\_,\mathsf{p})) \ast \mathsf{pt}(\mathsf{b},(\_,\mathsf{r})) \ast \mathsf{NEqualPair}(\mathsf{p},\mathsf{r})\\
&\quad \mid \exists \mathsf{c},\mathsf{d}.\ \mathsf{pt}(\mathsf{a},(\mathsf{c},\_)) \ast \mathsf{pt}(\mathsf{b},(\mathsf{d},\_)) \ast \mathsf{tt} \wedge \mathsf{c}\neq\mathsf{d}\\
\mathsf{Tail}(\mathsf{a}) &:= \mathsf{emp} \mid \exists \mathsf{b}.\mathsf{pt}(\mathsf{a},(\_,\mathsf{b})) \ast \mathsf{Tail}(\mathsf{b})\\
\mathsf{NEqualLen}(\mathsf{a},\mathsf{b}) &:= \exists \mathsf{x},\mathsf{p},\mathsf{r}.\mathsf{pt}(\mathsf{a},(\mathsf{x},\mathsf{p})) \ast \mathsf{pt}(\mathsf{b},(\mathsf{x},\mathsf{r})) \ast \mathsf{NEqualLen}(\mathsf{p},\mathsf{r})\\
&\quad \mid \exists \mathsf{p}.\mathsf{pt}(\mathsf{a},(\_,\mathsf{p})) \ast \mathsf{Tail}(\mathsf{p}) \mid \exists \mathsf{r}.\mathsf{pt}(\mathsf{b},(\_,\mathsf{r})) \ast \mathsf{Tail}(\mathsf{r})\\
\overline{\mathsf{PCP}}(\mathsf{a},\mathsf{b}) &:= \mathsf{emp} \mid \mathsf{NEqualPair}(\mathsf{a},\mathsf{b}) \mid \mathsf{NEqualLen}(\mathsf{a},\mathsf{b})
\end{aligned}$$

Fig. 1: The set P of inductive predicates for the reduction from PCP.

It is then easy to conjoin $\mathsf{EqPairs}(\mathsf{x},\mathsf{y})$ with $\mathsf{PCP}(\mathsf{x},\mathsf{y})$ such that a model exists if, and only if, the given PCP instance has a solution.

Inductive Predicates with Bounded Tree Width. Iosif *et al.* define in [17] a fragment of Separation Logic by syntactically restricting the definitions of inductive predicates such that all models have bounded tree width. In particular, their fragment requires that there is *exactly one* points-to predicate in any definition, which is clearly not the case in the reduction from PCP. Moreover, briefly speaking, further restrictions require that in each predicate definition, if a predicate name occurs in the body of a predicate definition then a points-to predicate occurs in the definition as well, that every existentially quantified variable is eventually allocated, and some further subtle technical conditions. We omit further details for space reason and show that entailment is ExpTime-hard in this fragment. The reader can easily verify that our reduction fulfils the requirements defined in [17].

Our reduction is from the language inclusion problem for non-deterministic top-down binary finite tree automata. A *prefix closed set of strings* over $\{0,1\}$ is a set of strings S such that for each $s\in S$ and any prefix s_p of s, s_p is also in S. A *binary ordered tree* t over a finite *alphabet* Σ is a tuple (N,Σ,ℓ), where N is a prefix closed set of strings over $\{0,1\}$ denoting the *nodes of the tree*, where for each $s\in N$, $s\cdot 1\in N$, if and only if $s\cdot 0\in N$, and $\ell : N\to\Sigma$ is a function assigning *labels* to nodes of the tree. The root of a tree is the empty string ϵ, and for any two nodes s and $s\cdot i$, for $i\in\{0,1\}$, $s\cdot i$ is a child node of s. We say that a node $s\in N$ is a *leaf node* if it has no child nodes, and a node is *internal* otherwise.

Recall that a *finite non-deterministic top-down tree automaton (NFTA) A* is a tuple (Σ,Q,δ,I), where Σ is a finite *alphabet*, Q is a finite set of *states* with a designated state q_{leaf}, $I\subseteq Q$ is the set of *initial or accepting states*, and $\delta : Q\times\Sigma\to 2^{Q\times Q}$ is the *transition function* such that for all $\sigma\in\Sigma$,

$\delta(q_{leaf},\sigma)=\emptyset$. A *run of* A on a tree $t=(N,\Sigma,\ell)$ is a function $\rho : N \to Q$ assigning states to the nodes of t such that for each internal node $s \in N$, $(\rho(s\cdot 0),\rho(s\cdot 1)) \in \delta(\rho(s),\ell(s))$ and for each leaf node $s \in N$, $\rho(s) = q_{leaf}$. A run is *accepting* if $\rho(\epsilon) \in I$, and the *language* $\mathcal{L}(A)$ *accepted by a NFTA* A is the set of trees t for which there is an accepting run of A on t.

<u>NFTA Language Inclusion Problem</u>
INPUT: Two NFTA A_1 and A_2.
QUESTION: Does $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ hold?

A classical result states that the language inclusion problem for non-deterministic tree automata is complete for ExpTime [21]. Let $A=(\Sigma,Q,\delta,I)$ be an NFTA. We define the subsequent set P of inductive predicates, where $\mathsf{Tree}_{q,\sigma}(\mathsf{a})$ is defined for every $\sigma \in \Sigma$ and $q \in Q$ for which $\delta(q,\sigma)$ is non-empty:

$$\begin{aligned}
\mathsf{Tree}_{q,\sigma}(\mathsf{a}) &:= \|_{\substack{(q_1,q_2)\in\delta(q,\sigma)\\ \sigma_1,\sigma_2\in\Sigma\\ \delta(q_1,\sigma_1)\neq\emptyset\vee q_1=q_{leaf}\\ \delta(q_2,\sigma_2)\neq\emptyset\vee q_2=q_{leaf}}} \exists \mathsf{l},\mathsf{r}.\ \mathsf{pt}(\mathsf{a},(\mathsf{s}_\sigma,\mathsf{l},\mathsf{r})) * \mathsf{Tree}_{q_1,\sigma_1}(\mathsf{l}) * \mathsf{Tree}_{q_2,\sigma_2}(\mathsf{r})\\
\mathsf{Tree}_{q_{leaf},\sigma}(\mathsf{a}) &:= \mathsf{pt}(\mathsf{a},\mathsf{s}_\sigma)\\
\mathsf{Trees}_A(\mathsf{a}) &:= \|_{\substack{q\in I\\ \sigma\in\Sigma}} \exists \mathsf{b}.\ \mathsf{pt}(\mathsf{a},\mathsf{b}) * \mathsf{Tree}_{q,\sigma}(\mathsf{b})
\end{aligned}$$

In any model, the predicate $\mathsf{Trees}_A(\mathsf{x})$ encodes all trees in $\mathcal{L}(A)$: apart from the node labeled with x, each allocated vertex represents a node of the tree, the first selector represents the label of the node, and the subsequent selectors represent respectively the left and right descendants, if the node is an internal node. The additional pointer at x is for technical reasons in order to comply with the restrictions defined in [17]. It is now easily checked that given two NFTA A_1 and A_2 over some alphabet $\Sigma = \{\sigma_1,\ldots,\sigma_n\}$,

$$\mathsf{Trees}_{A_1}(\mathsf{x}) \wedge \bigwedge_{1\le i\neq j\le n} \mathsf{s}_{\sigma_i} \neq \mathsf{s}_{\sigma_j} \models_P \mathsf{Trees}_{A_2}(\mathsf{x})$$

is a valid entailment if, and only if, $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$.

Theorem 5. *Deciding entailment in Separation Logic with inductive predicates with bounded tree width as defined in [17] is* ExpTime*-hard.*

It is worth emphasizing that hardness already holds if the arity of the pointer predicates is fixed to three. Also note that the ExpTime-hardness proof for satisfiability provided in [8] does not trivially establish that entailment in the fragment defined in [17] is ExpTime-hard since the definitions given in [8] are not in the fragment of [17].

Also, note that in [21] it is shown that for two NFTA that accept finite languages, the language inclusion problem is PSpace-complete, and therefore the proof of Theorem 5 can be adapted to show PSpace-hardness of entailment with inductive predicates not involving cyclic definitions.

4 Entailment for Fixed Fragments

The primary goal of this section is to first study the complexity of entailment in the base language defined in Section 2, and subsequently in the base language equipped with a fixed list predicate as defined in (1), which is a fragment commonly found in the program verifiers discussed in the introduction.

We first show that entailment in the base language is Π_2^{P}-complete. Moreover, we additionally outline that the upper bound even holds in the presence of the aforementioned list predicate. This result complements the previous section in that it indicates that for specific and fixed natural decidable fragments involving cyclic definitions of small arity the ExpTime lower bound can be avoided.

In the second part we analyse the lower bound from the first part and consider natural syntactic fragments defined in terms of structural properties of graphs representing SL-assertions. It has been shown that such restrictions can lead to polynomial-time decision procedures for entailment when dropping existentially quantified variables [14] and also decidability results for more expressive extensions of our base assertion language [7]. We show that basically there is no hope of achieving polynomial-time decision procedures in the presence of list predicates and existentially quantified variables, even when strong syntactic restrictions on the assertions are imposed.

Entailment in the General Case. We begin with the lower bound and show hardness for the base language, *i.e.* the language having only points-to predicates, via a reduction from a generalisation of *graph three-colorability* that has been defined in [1]. Recall that given an undirected graph $G = (V, E)$, graph three-colorability is to decide whether there is a *three-coloring* $f : V \to \{1, 2, 3\}$ such that $f(v) \neq f(w)$ for all $\{v, w\} \in E$. A *leaf coloring* of G is a function $f : V_l \to \{1, 2, 3\}$, where V_l is the set of vertices of G with degree one, *i.e.*, those nodes that have exactly one incident edge. The generalisation of graph three-colorability is given as follows.

<u>2-Round 3-Colorability</u>

INPUT: Undirected graph $G = (V, E)$.
QUESTION: For every fixed leaf coloring f of G, can f be extended to a three-coloring of G?

It has been shown in [1] that 2-Round 3-Colorability is Π_2^{P}-complete. We now show hardness of entailment for SL-formulas via a reduction from 2-Round 3-Colorability. To this end, we construct flat reduced SL-assertions α, α' such that the graph $G = (V, E)$ is a valid instance of 2-Round 3-Colorability iff $\alpha \models \alpha'$. We partition V into disjoint sets $V' = \{v_1, \ldots, v_n\}$ of nodes with degree greater than one and $V'' = \{v_{n+1}, \ldots, v_m\}$ of nodes with degree equal to one, and define α and α' such that

$$\alpha \overset{\mathrm{def}}{=} \mathop{\ast}_{\substack{i \in [3] \\ j \in [n]}} \mathsf{pt}(\mathsf{x}_{i,j}, \mathsf{y}_i) \ast \mathop{\ast}_{n < j \leq m} \mathsf{pt}(\mathsf{x}_j, \mathsf{z}_j) \wedge \bigwedge_{n < i \leq m} \bigvee_{j \in [3]} \mathsf{z}_i = \mathsf{y}_j \wedge \bigwedge_{1 \leq i \neq j \leq 3} \mathsf{y}_i \neq \mathsf{y}_j$$

$$\alpha' \overset{\mathrm{def}}{=} \exists_{i \in [n]} \mathsf{a}_i . \exists_{j \in [m]} \mathsf{b}_j . \mathop{\ast}_{i \in [n]} \mathsf{pt}(\mathsf{a}_i, \mathsf{b}_i) \ast \mathop{\ast}_{n < j \leq m} \mathsf{pt}(\mathsf{x}_j, \mathsf{b}_j) \ast \mathsf{tt} \wedge \bigwedge_{(v_i, v_j) \in E} \mathsf{b}_i \neq \mathsf{b}_j .$$

Intuitively speaking, the pointers $\mathsf{pt}(\mathsf{x}_j, \mathsf{z}_j)$ in α can choose in any model $\mathcal{I}$ of α a coloring of the leaves of G, represented by the variables y_i, $i \in [3]$. The $\mathsf{x}_{i,j}$ are allocated in order to enable α' to choose a coloring of the nodes which are not leaves. Consequently in a model $\mathcal{I}$ of α, an extension of $\mathcal{I}$ determining the existentially quantified variables b_i of α' determines a three-coloring of G. Conversely, if $\alpha \not\models \alpha'$ then the counter-model $\mathcal{I}$ encodes a coloring of the leaves which cannot be extended to a three-coloring.

It is not difficult to see that Π_2^{P}-hardness of entailment can already be established for SL-assertions without disjunction, by only requiring $\mathsf{y}_i \neq \mathsf{y}_j$ for all $1 \leq i \neq j \leq 3$ in the pure part of α: if $\alpha \models \alpha'$ then, in particular, any leaf coloring can be extended to a three-coloring since a subset of the models of α will encode all possible leaf colorings. The converse direction then follows as above. In addition, by introducing additional existentially quantified slack variables, the hardness proof can also be tightened in a way such that no "tt" is required in spatial formulas, *i.e.*, hardness holds in non-intuitionistic fragments. Finally, this reduction can be reused in order to show NP-hardness of the model checking problem via a reduction from 3-Colorability.

Since the size of all relevant models is *a priori* fixed by the size of the formulas under consideration, a Π_2^{P} upper bound follows trivially.

Theorem 6. *Entailment for flat reduced SL-assertions is* Π_2^{P}*-complete.*

For the remainder of this section, we turn towards entailment in the base language equipped with an additional fixed list predicate as defined in (1) and restrict our attention to pointer predicates of arity one, and consequently to interpretations which are functional graphs.

First, we can also establish a Π_2^{P} upper bound for entailment in this fragment by showing a small-model property. The main idea is that for a sufficiently large $\mathcal{I}$ such that $\mathcal{I} \models \alpha$ and $\mathcal{I} \not\models \alpha'$, we can find an instantiation of an $\mathsf{ls}(\mathsf{x}, \mathsf{y})$ predicate in $\mathcal{I}$ such that the path between x and y is long enough that we can obtain a new $\mathcal{I}'$ by removing a vertex occurring on this path while ensuring that the newly obtained $\mathcal{I}'$ is still a counter-model witnessing $\alpha \not\models \alpha'$.

Lemma 7. *Let* α, α' *be SL-assertions, let* $n = |vars(\alpha)| + |vars(\alpha')|$ *and suppose that* $\alpha \not\models \alpha'$*. Then there exists an* $\mathcal{I}$ *witnessing* $\alpha \not\models \alpha'$ *with* $|V^{\mathcal{I}}| \in O(n^2)$*.*

This lemma immediately yields a Π_2^{P} upper bound: in order to show $\alpha \not\models \alpha'$, we can guess a small model $\mathcal{I}$ of α and then verify with an NP oracle that $\mathcal{I} \not\models \alpha'$.

Theorem 8. *Entailment for flat reduced SL-assertions with a fixed list predicate is* Π_2^{P}*-complete.*

Entailment under Structural Restrictions. The goal of this section is to argue that entailment in essentially any useful fragment is intractable in the presence of existential quantification and list predicates, even under severe syntactic restrictions. In the following, we will only consider positive SL-assertions, since otherwise non-entailment is trivially coNP-hard.

In order to identify syntactic fragments with potentially polynomial-time entailment problems, we look at properties of graphs representing SL-formulas. Let $G = (V, E)$ be a directed graph and $v \in V$ a vertex of G. Then, define functions $\mathit{pred}(v) \stackrel{\text{def}}{=} \{v' \in V : (v', v) \in E\}$ and $\mathit{succ}(v) \stackrel{\text{def}}{=} \{v' \in V : (v, v') \in E\}$. A node $v \in V$ is a *source node* if $\mathit{pred}(v) = \emptyset$, and v is a *sink node* if $\mathit{succ}(v) = \emptyset$. Let $\alpha = \sigma \wedge \varphi$ be an SL-assertion, and $\mathsf{x} \in \mathit{vars}(\alpha)$ be a variable of α. Then define the set $\mathsf{Eq}(\varphi, \mathsf{x}) = \{\mathsf{y} \in \mathit{vars}(\alpha) : \text{for all } \mathcal{I}, \text{ if } \mathcal{I} \models \varphi \text{ then } \mathcal{I} \models \mathsf{x} = \mathsf{y}\}$. Next, we define the *graph $G(\alpha)$ corresponding to α* as $G(\alpha) = (V_\alpha, E_\alpha, \ell_\alpha)$, where the set of vertices is defined as $V_\alpha \stackrel{\text{def}}{=} \{\mathsf{Eq}(\varphi, \mathsf{x}) : \mathsf{x} \in \mathit{vars}(\alpha)\}$, and the set of edges as $E_\alpha \stackrel{\text{def}}{=} \{(\mathsf{Eq}(\varphi, \mathsf{x}), \mathsf{Eq}(\varphi, \mathsf{y})) : \mathsf{pt}(\mathsf{x}, \mathsf{y}) \text{ or } \mathsf{ls}(\mathsf{x}, \mathsf{y}) \text{ occurs in } \sigma\}$. Finally, ℓ_α is such that $\ell_\alpha(\mathsf{x}) = \mathsf{Eq}(\varphi, \mathsf{x})$ for all $\mathsf{x} \in \mathit{vars}(\alpha)$. A node $v \in V_\alpha$ is *fixed* if there is $\mathsf{x} \in \mathsf{FVars}$ such that $\ell_\alpha(\mathsf{x}) = v$.

Inspecting the Π_2^{P}-hardness proof above, we see that one fundamental source of complexity comes from having pointers $\mathsf{pt}(\mathsf{a}, \mathsf{b})$ with $\mathsf{a}, \mathsf{b} \in \mathsf{EVars}$, *i.e.*, the graph corresponding to α' above has source and sink nodes which are not fixed. On the one hand, when not allowing for list predicates, if all source nodes of an SL-assertion were to be fixed, entailment between formulas in such a suitably defined fragment would trivially become polynomial-time decidable. The main reason for this is that in any model $\mathcal{I}$ of $\exists \mathsf{a}.\mathsf{pt}(\mathsf{x}, \mathsf{a})$, a would have to be instantiated with the *unique* successor of the vertex labeled with x. However, such a fragment would only allow for reasoning about models whose size is *a priori* fixed by the size of the antecedent, which limits its usefulness. On the other hand, when allowing for list predicates, the proof of NP-hardness of abduction (given in [15]) can be easily adapted to show that entailment becomes intractable even if source nodes are required to be fixed.

This leaves us with an interesting case, which we introduce by considering an example of an instance of entailment:

$$\mathsf{ls}(\mathsf{x}, \mathsf{y}) \wedge \mathsf{x} \neq \mathsf{y} \models \exists \mathsf{a}.\mathsf{pt}(\mathsf{x}, \mathsf{a}) * \mathsf{ls}(\mathsf{a}, \mathsf{y}).$$

The validity of this entailment rests on the fact that x has a successor in any model containing a non-empty list from x to y. In this example, the consequent is a formula of the fragment of the assertion language which we are going to consider in the remainder of this section: an SL-assertion α is in the *fixed endpoints fragment* if all source and sink nodes of the graph $G(\alpha)$ corresponding to α are fixed. We show coNP-hardness of entailment in this fragment via a reduction from an NP-complete variant of the Hamiltonian path problem.

Fixed Vertex Hamiltonian Path (FVHP)

INPUT: A directed graph $G = (V, E)$ and $v \in V$.
QUESTION: Does there exist a Hamiltonian path in G ending in v?

Given a graph $G = (V, E)$, a vertex $v \in V$ and an instance of FVHP such that $V = \{v_1, \ldots, v_{N+1}\}$ and $v = v_{N+1}$, we show how to compute in polynomial time SL-formulas α, α' in the fixed endpoints fragment such that $\alpha \not\models \alpha'$ if and only if G is a valid instance of FVHP. For $i \in [N+1]$ and $j \in [N]$, for the spatial parts of α and α' we define:

$$\begin{aligned}
\mathsf{node}_j &\overset{\text{def}}{=} \mathsf{ls}(\mathsf{e}^{\mathsf{s}}_j, \mathsf{a}^0_j) * \mathop{\ast}_{k\in[0,N-1]} \mathsf{ls}(\mathsf{a}^k_j, \mathsf{b}^{k+1}_j) * \mathop{\ast}_{k\in[N-1]} \mathsf{ls}(\mathsf{b}^k_j, \mathsf{a}^k_j) * \mathsf{ls}(\mathsf{b}^N_j, \mathsf{e}^{\mathsf{f}}_j) \\
\mathsf{order}^0_i &\overset{\text{def}}{=} \mathsf{pt}(\mathsf{s}^0_i, \mathsf{f}^0_i) \\
\mathsf{order}^j_i &\overset{\text{def}}{=} \mathsf{ls}(\mathsf{s}^j_i, \mathsf{d}^j_i) * \mathsf{ls}(\mathsf{d}^j_i, \mathsf{f}^j_i) \\
\sigma &\overset{\text{def}}{=} \mathop{\ast}_{j\in[N]} \mathsf{node}_j * \mathop{\ast}_{\substack{i\in[N+1]\\ j\in[0,N]}} \mathsf{order}^j_i
\end{aligned}$$

A graphical illustration of the formulas node_j, order^j_i and order^0_i is given in Figure 2, where list predicates are represented as dashed arrows and pointer predicates as full arrows. Consider the submodels of each of the formulas above. The intuition behind these formulas, in conjunction with the inequalities introduced below, is that there will be a model comprising a long concatenation, loosely speaking, of such submodels, if and only if there is a hamiltonian path in the given graph. The inequalities introduced below, will additionally ensure that such a long concatenation can happen only in the models of α and not of α', and thus entailment will not hold in such a case. The variables a^k_j and b^k_j are essentially used in a way to count how long the concatenation is.

We now turn towards the pure parts of α and α'. For notational convenience, given $\mathsf{x} \in \mathit{vars}(\alpha)$ and $\mathsf{S} \subseteq \mathit{vars}(\alpha)$, subsequently $\mathsf{x} \approx S$ abbreviates $\bigwedge_{\mathsf{y}\in \mathit{vars}(\alpha)\setminus(\mathsf{S}\cup\{\mathsf{x}\})} \mathsf{x} \neq \mathsf{y}$. In other words, in any model $\mathcal{I}$ of $\mathsf{x} \approx \mathsf{S}$, if $\ell^{\mathcal{I}}(\mathsf{x}) = \ell^{\mathcal{I}}(\mathsf{y})$ for some $\mathsf{y} \in \mathit{vars}(\alpha)$ then $\mathsf{y} \in \mathsf{S}$ or $\mathsf{x} \equiv \mathsf{y}$. We define Dvars to be the set $\{\mathsf{d}^\ell_k : k \in [N+1], \ell \in [0,N]\}$, and we define φ to be the conjunction of the subsequent pure formulas:

$$\begin{aligned}
&\mathsf{s}^0_i \approx \emptyset \wedge \mathsf{f}^N_i \approx \mathsf{Dvars} \wedge \mathsf{d}^0_i = \mathsf{f}^0_i, && i \in [N+1] \\
&\mathsf{s}^j_i \approx \bigcup_{\substack{v_p\in pred(v_i),\\ N-j<k\leq N-1}} \{\mathsf{a}^k_p, \mathsf{b}^k_p, \mathsf{b}^N_p, \mathsf{e}^{\mathsf{f}}_p\} \cup \mathsf{Dvars}, && i \in [N+1], j \in [N] \\
&\mathsf{f}^j_i \approx \{\mathsf{e}^{\mathsf{s}}_i, \mathsf{a}^0_i, \mathsf{b}^{N-j}_i\} \cup \bigcup_{k\in[N-j-1]} \{\mathsf{a}^k_i, \mathsf{b}^k_i\} \cup \mathsf{Dvars}, && i \in [N], j \in [0,N-1] \\
&\mathsf{f}^j_{N+1} \approx \mathsf{Dvars}, && j \in [0,N-1] \\
&\mathsf{e}^{\mathsf{f}}_i \neq \mathsf{e}^{\mathsf{f}}_j, && 1 \leq i \neq j \leq N \\
&\mathsf{d}^j_i \neq \mathsf{b}^{N-j}_i, && i \in [N], j \in [0,N-1] \\
&\bigwedge_{k\in[N]\setminus\{i\}} \mathsf{d}^j_i \neq \mathsf{b}^{N-j+1}_k && i \in [N+1], j \in [N]
\end{aligned}$$

Finally, we define α and α' using the set of variables V shown below:

$$\mathsf{V} \overset{\text{def}}{=} \{\mathsf{a}^0_i, \mathsf{a}^j_i, \mathsf{b}^j_i, \mathsf{b}^N_i : i \in [N], j \in [N-1]\} \cup \{\mathsf{d}^j_i : i \in [N+1], j \in [0,N]\}$$

$$\alpha \overset{\text{def}}{=} \exists_{\mathsf{x}\in\mathsf{V}} \mathsf{x}.\sigma \wedge \varphi \qquad \text{and} \qquad \alpha' \overset{\text{def}}{=} \exists_{\mathsf{x}\in\mathsf{V}} \mathsf{x}.\sigma \wedge \varphi \wedge \mathsf{d}^N_{N+1} \neq \mathsf{f}^N_{N+1}$$

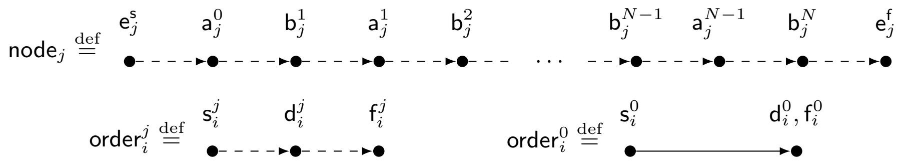
$\mathsf{node}_j \overset{\text{def}}{=}$
$\mathsf{e}^{\mathsf{s}}_j$
a^0_j
b^1_j
a^1_j
b^2_j
$\cdots$
b^{N-1}_j
a^{N-1}_j
b^N_j
$\mathsf{e}^{\mathsf{f}}_j$
$\mathsf{order}^j_i \overset{\text{def}}{=}$
s^j_i
d^j_i
f^j_i
$\mathsf{order}^0_i \overset{\text{def}}{=}$
s^0_i
$\mathsf{d}^0_i, \mathsf{f}^0_i$

Fig. 2: Graphical representation of the formulas node_i and order^j_i.

Note that φ includes $\mathsf{f}^N_i \approx \mathsf{Dvars}$ for all $i \in [N+1]$. Given the additional disequality in α', we now have $\mathsf{f}^N_{N+1} \approx \mathsf{Dvars} \setminus \{\mathsf{d}^N_{N+1}\}$ in the pure part of α'.

Also note that in order to simplify the presentation, we have defined α and α' such that we use the same existentially quantified variables both in α and α'. It is important to note that given a model $\mathcal{I}$ of both α and α', the extension $\hat{\mathcal{I}}_1$ of $\mathcal{I}$ that witnesses the satisfaction of the formula α and the extension $\hat{\mathcal{I}}_2$ that witnesses the satisfaction of the formula α' do not in general agree on the mapping of those existentially quantified variables. The existentially quantified variables in α could also be seen as fixed variables with names different from the existentially quantified variables of α'. As these variables act in the same way in both formulas, in order to avoid writing the above definitions twice and to simplify our proof, we have decided to treat them as existentially quantified and define them such that they have the same name in both formulas.

Lemma 9. *$G = (V, E)$ and $v \in V$ is a valid instance of* FVHP *iff $\alpha \not\models \alpha'$.*

Proof (sketch). First, a crucial observation is that for any $\mathcal{I}$ such that $\mathcal{I} \models \alpha$ and $\mathcal{I} \not\models \alpha'$, in *any* extension $\hat{\mathcal{I}}$ witnessing $\mathcal{I} \models \alpha$, we have that the instantiation of d^N_{N+1} coincides with f^N_{N+1}. Suppose this was not the case, then $\hat{\mathcal{I}}$ would also witness $\mathcal{I} \models \alpha'$, contradicting our assumption. In this case we say that d^N_{N+1} is *forced on* f^N_{N+1}. We can show that forcing d^N_{N+1} on f^N_{N+1} is only possible if b^1_p is forced on s^N_{N+1} for some unique predecessor $v_p \in \mathit{pred}(v_N)$ of v_N. In order to force b^1_p on s^N_{N+1}, it can then be shown that d^{N-1}_p and therefore f^{N-1}_p is forced on a^0_p. In summary, we can establish the following chain of inductive reasoning:

if $\mathsf{d}^N_{i_0}$ is forced on $\mathsf{f}^N_{i_0}$ then $\mathsf{b}^1_{i_1}$ is forced on $s^N_{i_0}$ for some $v_{i_1} \in \mathit{pred}(v_{i_0})$
if $\mathsf{b}^1_{i_1}$ is forced on $\mathsf{s}^N_{i_0}$ then $\mathsf{d}^{N-1}_{i_1}$ is forced on $\mathsf{f}^{N-1}_{i_1}$
$\vdots \quad \vdots \quad \vdots \quad \vdots$
if $\mathsf{d}^N_{i_{N-1}}$ is forced on $\mathsf{f}^N_{i_{N-1}}$ then $\mathsf{b}^N_{i_1}$ is forced on $s^1_{i_0}$ for some $v_{i_N} \in \mathit{pred}(v_{i_{N-1}})$
if $\mathsf{b}^N_{i_N}$ is forced on $\mathsf{s}^1_{i_{N-1}}$ then $\mathsf{d}^0_{i_N}$ is forced on $\mathsf{f}^0_{i_N}$

Now considering the variable names b^j_i in the implication chain, we obtain a sequence $\mathsf{b}^1_{i_1}, \mathsf{b}^2_{i_2}, \ldots, \mathsf{b}^{N-1}_{i_{N-1}}, \mathsf{b}^N_{i_N}$ such that for all $1 \leq j < N$, v_{i_j} is a successor of $v_{i_{j+1}}$ in G. Consequently, the sequence of nodes $\pi = v_{i_N} \cdots v_{i_2} v_{i_1} v_{N+1}$ is a path of length $N+1$ in G ending in v_{N+1}. Using the definition of φ, it follows that any b^j_i can only be "used" once, hence π does not contain duplicate nodes and thus is a Hamiltonian path ending in v_{N+1}. $\square$

It is readily checked that source and sink nodes in the graph corresponding to the definition of α and α' are fixed. Hence, we have established the following.

Theorem 10. *Entailment in the fixed endpoints fragment is* coNP*-hard.*

5 Conclusion

The results in this paper can be interpreted as follows: when considering fragments of Separation Logic which allow for existential quantification and unbounded data structures, having tractable, polynomial-time decision procedures will require severe syntactic restriction, since entailment is coNP-hard even in the strongly constrained fixed endpoints fragment. In the presence of general inductive predicates, although satisfiability is decidable [8], we have shown that entailment becomes undecidable. This result complements the decidability result obtained by Iosif *et al.* [17] and shows that the syntactic restrictions defined in [17] are not only natural but also crucial for decidability. However, we have shown that entailment in this fragment is ExpTime-hard. On the more positive side, we have shown that entailment is "only" Π_2^{P}-complete in the presence of existential quantification, pointers and linked lists. Since this is a fragment that has been shown to be useful in program verifiers, this result may be seen as an argument in favour of supporting the development of decision procedures for domain-specific fragments of Separation Logic with a fixed set of predicates.

A number of problems remain open which we intend to investigate in the future. For instance, an open issue is whether a restriction to a one-selector fragment leads to decidable entailment. Also, although we have shown ExpTime-hardness for the bounded-tree width fragment, we currently do not know whether this is a tight bound. This is also true for the fixed endpoints fragment and its coNP-hardness. Finally, of great interest would be decision procedures for entailment in these fragments, since most tools use incomplete heuristics.

Acknowledgments. We would like to thank the referees for their helpful comments. In particular, we wish to thank one referee who suggested the reduction from the inclusion problem for tree automata for the proof of Theorem 5.

References

1. M. Ajtai, R. Fagin, and L. Stockmeyer. The closure of monadic NP. *Journal of Computer and System Sciences*, 60(3):660–716, 2000.
2. J. Bengtson, J. Braband Jensen, and L. Birkedal. Charge! In L. Beringer and A. Felty, editors, *Interactive Theorem Proving*, volume 7406 of *LNCS*, pages 315–331. Springer, 2012.
3. J. Berdine, C. Calcagno, B. Cook, D. Distefano, P. W. O'Hearn, T. Wies, and H. Yang. Shape analysis for composite data structures. In W. Damm and H. Hermanns, editors, *Computer Aided Verification*, volume 4590 of *LNCS*, pages 178–192. Springer, 2007.
4. J. Berdine, C. Calcagno, and P. O'Hearn. A decidable fragment of separation logic. In *Foundations of Software Technology and Theoretical Computer Science*, volume 3328 of *LNCS*, pages 110–117. Springer, 2005.

5. J. Berdine, B. Cook, and S. Ishtiaq. SLAyer: Memory safety for systems-level code. In G. Gopalakrishnan and S. Qadeer, editors, *Computer Aided Verification*, volume 6806 of *LNCS*, pages 178–183. Springer, 2011.
6. L. Birkedal, N. Torp-Smith, and J. C. Reynolds. Local reasoning about a copying garbage collector. In *Principles of Programming Languages*, pages 220–231, New York, NY, USA, 2004. ACM.
7. A. Bouajjani, C. Drăgoi, C. Enea, and M. Sighireanu. Accurate invariant checking for programs manipulating lists and arrays with infinite data. In *Automated Technology for Verification and Analysis*, LNCS, pages 167–182. Springer, 2012.
8. J. Brotherston, C. Fuhs, N. Gorogiannis, and J. Navarro Pérez. A decision procedure for satisfiability in separation logic with inductive predicates. Technical Report RN/13/15, University College London, 2013.
9. J. Brotherston, N. Gorogiannis, and R.L. Petersen. A generic cyclic theorem prover. In *Asian Symposium on Programming Languages and Systems*, pages 350–367. Springer, 2012.
10. J. Brotherston and M. Kanovich. Undecidability of propositional separation logic and its neighbours. In *Logic in Computer Science*, pages 137–146. IEEE Computer Society, 2010.
11. C. Calcagno, D. Distefano, P. W. O'Hearn, and H. Yang. Beyond reachability: Shape abstraction in the presence of pointer arithmetic. In K. Yi, editor, *Static Analysis*, volume 4134 of *LNCS*, pages 182–203. Springer, 2006.
12. C. Calcagno, H. Yang, and P.W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *Asian Symposium on Programming Languages and Systems*, pages 289–300, 2001.
13. W.-N. Chin, C. David, H. H. Nguyen, and S. Qin. Automated verification of shape, size and bag properties via user-defined predicates in separation logic. *Science of Computer Programming*, 77(9):1006 – 1036, 2012.
14. B. Cook, C. Haase, J. Ouaknine, M. Parkinson, and J. Worrell. Tractable reasoning in a fragment of separation logic. In *Concurrency Theory*, volume 6901 of *LNCS*, pages 235–249. Springer, 2011.
15. N. Gorogiannis, M. Kanovich, and P. O'Hearn. The complexity of abduction for separated heap abstractions. In *Static Analysis*, volume 6887 of *LNCS*, pages 25–42. Springer, 2011.
16. P. Habermehl, L. Holík, A. Rogalewicz, J. Šimáček, and T. Vojnar. Forest automata for verification of heap manipulation. In G. Gopalakrishnan and S. Qadeer, editors, *Computer Aided Verification*, volume 6806 of *LNCS*, pages 424–440. Springer, 2011.
17. R. Iosif, A. Rogalewicz, and J. Šimáček. The tree width of separation logic with recursive definitions. In M. P. Bonacina, editor, *Automated Deduction – CADE*, volume 7898 of *LNCS*, pages 21–38. Springer, 2013.
18. S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *Principles of Programming Languages*, pages 14–26. ACM, 2001.
19. E. L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.
20. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Logic in Computer Science*. IEEE Computer Society, 2002.
21. H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, June 1990.
22. H. Yang. *Local Reasoning for Stateful Programs.* PhD thesis, University of Illinois at Urbana-Champaign, 2001. (Technical Report UIUCDCS-R-2001-2227).