Tractable Reasoning in a Fragment of Separation Logic

Byron Cook[1,3], Christoph Haase[2], Joël Ouaknine[2], Matthew Parkinson[1], and James Worrell[2]

[1] Microsoft Research Cambridge, UK
[2] Department of Computer Science, University of Oxford, UK
[3] Department of Computer Science, Queen Mary University of London, UK

Abstract. In 2004, Berdine, Calcagno and O'Hearn introduced a fragment of separation logic that allows for reasoning about programs with pointers and linked lists. They showed that entailment in this fragment is in coNP, but the precise complexity of this problem has been open since. In this paper, we show that the problem can actually be solved in polynomial time. To this end, we represent separation logic formulae as graphs and show that every satisfiable formula is equivalent to one whose graph is in a particular normal form. Entailment between two such formulae then reduces to a graph homomorphism problem. We also discuss natural syntactic extensions that render entailment intractable.

1 Introduction

Separation logic (SL) [11, 14] is an extension of Hoare logic to reason about pointer manipulating programs. It extends the syntax of assertions with predicates describing shapes of memory; aliasing and disjointness can be concisely expressed within these shapes. This extended assertion languages allows elegant and concise hand written proofs of programs that manipulate dynamically allocated data structures. However, generating such proofs in an automated fashion is constrained by the undecidability of separation logic [14]. For that reason, in recent years research has been concentrating on finding decidable fragments of this logic, see e.g. [2, 5].

In this paper, we study the SL fragment presented in [2]. This fragment allows for reasoning about structural integrity properties of programs with pointers and linked lists. In [2], the decidability of checking validity of entailments in this logic has been shown. Entailment is the problem to decide whether, given two separation logic assertions α and α', α' holds in every memory model in which α holds. Decidability was shown in model-theoretic terms and by providing a complete syntactic proof theory for this fragment. Based on these theoretical results, Berdine, Calcagno and O'Hearn [3] later developed the tool Smallfoot. This tool decides entailments via a syntactic proof search using the proof theory, however in the worst case an exponential number of proofs have to be explored. The tool demonstrated that SL could be used to automatically verify memory safety of linked list and tree manipulating programs. Based on the success of Smallfoot, this approach has been extended to allow automatic inference of specifications of systems code [1, 4], to reason about object-oriented programs [7, 12], and even to reason about non-blocking concurrent programs [3]. But fundamentally all these tools are based on the same style of syntactic proof theory.

The precise computational complexity of checking entailments was not fully answered in [2]. The authors show that a memory model disproving an entailment is polynomial in the size of the input, thus giving a coNP algorithm. As we are going to show in this paper, entailment can actually be decided in *polynomial time*. To this end, we take a fundamentally different approach to [2]: Instead of reasoning syntactically about formulae, we represent them as graphs in a particular normal form and then compute a homomorphism between those graphs to prove that an entailment holds. It is well-known [8] that computing graph homomorphisms is an NP-complete problem, however our graphs in normal form enjoy some special structural properties that allow one to compute homomorphisms in polynomial time.

This paper is structured as follows: In Section 2 we formally introduce our SL fragment, graphs and the decision problems that we consider. Section 3 then shows how we can compute in polynomial time from a given assertion a graph in normal form that represents the same set of models of the formula. We then show in Section 4 that a homomorphism between graphs in normal form witnesses an entailment, and that such a homomorphism can be computed in polynomial time. Section 5 deals with syntactic extensions that make entailment coNP-hard.

Due to space constraints, we do not present all algorithms and proofs in the main part of this paper, they can however be found in an extended version [6]. Moreover, we assume the reader to be familiar with basic notions and concepts of separation logic. For a comprehensive introduction to separation logic, see [14].

2 Preliminaries

Let Vars and V be countably infinite sets of *variables* and *nodes*. We assume some fixed total order $<$ on Vars and for any finite $S \subseteq \mathsf{Vars}$, denote by $min(S)$ the unique $x \in S$ such that $x \leq y$ for all $y \in S$.

The syntax of our assertion language is given by the following grammar, where x ranges over Vars:

$$\begin{aligned}
expr &::= x && (\textit{expressions})\\
\phi &::= expr = expr \mid expr \neq expr \mid \phi \wedge \phi && (\textit{pure formulae})\\
\sigma &::= expr \mapsto expr \mid \mathsf{ls}(expr, expr) \mid \sigma * \sigma && (\textit{spatial forumlae})\\
\alpha &::= (\phi; \sigma) && (\textit{assertions})
\end{aligned}$$

Subsequently, we call formulae of our assertion language *SL-formulae*. An example of an SL-formula is $\alpha = (x \neq y; \mathsf{ls}(x, y) * y \mapsto z)$. It describes memory models

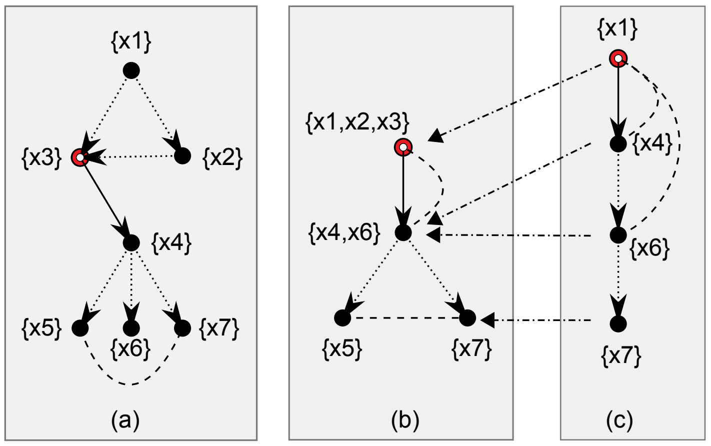

Fig. 1. Three SL-graphs, where l-edges are dotted arrows, p-edges solid arrows and d-edges dashed lines. Nodes are labelled with the variables next to them. The graphs (b) and (c) are in normal form, where (b) is obtained by reducing (a). The arrows from (c) to (b) depict a homomorphism.

in which the value of the stack variable x is not equal to the value of the stack variable y, and in which the heap can be separated into two disjoint segments such that in one segment there is a linked list from the heap cell whose address is the value of x to the heap cell whose address is the value of y, and where in the other segment the latter heap cell points to the heap cell whose address is z. We denote by $|\phi|$ the *size* of a pure formula and by $|\sigma|$ the size of a spatial formula, which is in both cases the number of symbols used to write down the formula. Given an assertion $\alpha = (\phi; \sigma)$, the size of α is $|\alpha| \overset{\text{def}}{=} |\phi| + |\sigma|$. By ϵ, we subsequently denote the empty spatial assertion of size zero.

Remark 1. The SL fragment considered in [2] also contains *nil* as an expression. This does however not give more expressiveness, since we can introduce a designated variable *nil* and implicitly join $nil \mapsto nil$ to every spatial assertion to obtain the same effect.

The semantics of SL-formulae is given in terms of SL-graphs, which we define to be a special class of directed graphs. Throughout this paper, SL-graphs will also be used to represent SL-formulae.

Definition 2. *An* SL-graph G *is either* $\bot$ *or* $(V_b, V_r, E_l, E_p, E_d, \ell)$ *such that*

- $V_b, V_r \subseteq_{fin} V$, $V_b \cap V_r = \emptyset$, $V_{b,r} \overset{def}{=} V_b \cup V_r$;
- $E_l \subseteq V_{b,r} \times V_{b,r}$;
- $E_p \subseteq V_r \times V_{b,r}$ *and for every* $v \in V_r$, $E_p(v)$ *is defined;*
- $E_d \subseteq \{\{v, w\} : v, w \in V_{b,r}, v \neq w\}$;
- $\ell : \mathsf{Vars} \rightharpoonup_{fin} V_{b,r}$

$$\begin{aligned}
\mathcal{I} \models x = y &\iff \ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(y)\\
\mathcal{I} \models x \neq y &\iff \ell^{\mathcal{I}}(x) \neq \ell^{\mathcal{I}}(y)\\
\mathcal{I} \models \phi_1 \wedge \phi_2 &\iff \mathcal{I} \models \phi_1 \text{ and } \mathcal{I} \models \phi_2\\
\mathcal{I} \models x \mapsto y &\iff \exists v, w \in V^{\mathcal{I}}_{b,r}. V^{\mathcal{I}}_r = \{v\}, E^{\mathcal{I}}_p = \{(v, w)\}, \ell^{\mathcal{I}}(x) = v, \ell^{\mathcal{I}}(y) = w\\
\mathcal{I} \models \mathsf{ls}(x, y) &\iff \exists n \in \mathbb{N}. \mathcal{I} \models \mathsf{ls}^n(x, y)\\
\mathcal{I} \models \mathsf{ls}^0(x, y) &\iff \ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(y) \text{ and } V^{\mathcal{I}}_r = \emptyset\\
\mathcal{I} \models \mathsf{ls}^{n+1}(x, y) &\iff \exists z \notin \mathit{dom}(\ell^{\mathcal{I}}), v \in V. \mathcal{I}[\ell/\ell[z \mapsto v]] \models x \mapsto z * \mathsf{ls}^n(z, y)\\
\mathcal{I} \models \sigma_1 * \sigma_2 &\iff \exists \mathcal{I}_1, \mathcal{I}_2. \mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2, \mathcal{I}_1 \models \sigma_1, \mathcal{I}_2 \models \sigma_2\\
\mathcal{I} \models (\phi; \sigma) &\iff \mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2, \mathcal{I}_1 \models \phi \text{ and } \mathcal{I}_1 \models \sigma, \text{ where } \mathcal{I} \models \epsilon \text{ for all } \mathcal{I}
\end{aligned}$$

Table 1. Semantics of the assertion language, where $\mathcal{I}$ is an SL interpretation.

An SL interpretation *is an SL-graph where $E_l = \emptyset$, E_p is functional and $E_d = \{\{v, w\} : v, w \in V_{b,r}, v \neq w\}$.*

An SL-graph $\perp$ indicates an inconsistent SL-graph. The set $V_{b,r}$ of *nodes* of an SL-graph partitions into sets V_b and V_r, where we refer to nodes in V_b as *black nodes* and to those in V_r as *red nodes*. We call E_p the set of *pointer edges (p-edges)*, E_l the set of *list edges (l-edges)*, E_d is the set of *disequality edges (d-edges)* and ℓ the *variable labelling function*. For convenience, $E_{p,l}$ denotes the set $E_p \cup E_l$. Given a node $v \in V$, we set $\mathit{vars}(v) \overset{\text{def}}{=} \{x \in \mathsf{Vars} : \ell(x) = v\}$ and $\mathit{var}(v) \overset{\text{def}}{=} \mathit{min}(\mathit{vars}(v))$. We sometimes wish to alter one component of a graph and, e.g., write $G[E_p/E'_p]$ to denote the graph $G' = (V_b, V_r, E'_p, E_l, E_d, \ell)$.

Example 3. Figure 1 shows three examples of SL-graphs. Subsequently, we identify nodes of an SL-graph with any of the variables they are labelled with. Graph (a) has an l-edge from the black node x_1 to the red node x_3, depicted by a dotted arrow. The latter node has a p-edge to the black node x_4, depicted by a solid arrow. Moreover, there is a d-edge between x_5 and x_7, depicted by a dashed line.

In the remainder of this paper, we denote an SL interpretation by $\mathcal{I}$ and usually denote the components of an interpretation with superscript $\mathcal{I}$, e.g., we write $V^{\mathcal{I}}_b$ to denote the black nodes of an interpretation $\mathcal{I}$. Given SL interpretations $\mathcal{I}, \mathcal{I}', \mathcal{I}''$, we define $\mathcal{I} = \mathcal{I}' * \mathcal{I}''$ if, and only if, $V^{\mathcal{I}}_r = V^{\mathcal{I}'}_r \uplus V^{\mathcal{I}''}_r$, $V^{\mathcal{I}'}_b = V^{\mathcal{I}}_b \cup V^{\mathcal{I}''}_r$, $V^{\mathcal{I}''}_b = V^{\mathcal{I}}_b \cup V^{\mathcal{I}'}_r$, $E^{\mathcal{I}}_p = E^{\mathcal{I}'}_p \uplus E^{\mathcal{I}''}_p$, and $\ell^{\mathcal{I}} = \ell^{\mathcal{I}'} = \ell^{\mathcal{I}''}$. The semantics of our assertion language is presented in Table 1. We call $\mathcal{I}$ a *model* of α if $\mathcal{I} \models \alpha$.

Remark 4. In [2], the semantics of SL-formulae is given in terms of heaps and stacks. In our setting, we can view the red nodes of an interpretation as the set of allocated heap cells, $E^{\mathcal{I}}_p$ as a representative of the contents of heap cells and $\ell^{\mathcal{I}}$ as the stack. Black nodes then correspond to dangling locations. Moreover, our semantics differs in that we employ the *intuitionistic* model of separation logic [14] and that the semantics of lists is *imprecise*. We will discuss the relationship to the semantics given in [2] in Section 4.

The decision problems of interest to us are *satisfiability* and *entailment*. Given an assertion α, we say α *is satisfiable* if there exists a model $\mathcal{I}$ such that $\mathcal{I} \models \alpha$. Given two assertions α_1 and α_2, we say α_1 *entails* α_2 if for any SL interpretation $\mathcal{I}$, whenever $\mathcal{I} \models \alpha_1$ then $\mathcal{I} \models \alpha_2$. We write $\alpha_1 \models \alpha_2$ if α_1 entails α_2, and $\alpha_1 \equiv \alpha_2$ if $\alpha_1 \models \alpha_2$ and $\alpha_2 \models \alpha_1$.

Given an SL-graph G, we now define its *corresponding assertion* $\alpha(G)$. If $G = \bot$ then $\alpha(G) \stackrel{\text{def}}{=} (x \neq x; \epsilon)$, i.e., an unsatisfiable SL-formula. Otherwise, the assertion $\alpha(G)$ corresponding to G is defined as follows, where we use an indexed separation operator:

$$\phi(G) \stackrel{\text{def}}{=} \bigwedge_{\substack{v \in V_{b,r} \\ x,y \in vars(v)}} x = y \wedge \bigwedge_{\{v,w\} \in E_d} var(v) \neq var(w),$$

$$\sigma(G) \stackrel{\text{def}}{=} \left(*_{(v,w) \in E_p} var(v) \mapsto var(w) \right) * \left(*_{(v,w) \in E_l} \mathsf{ls}(var(v), var(w)) \right),$$

$$\alpha(G) \stackrel{\text{def}}{=} (\phi(G), \sigma(G)).$$

We define the *size* of an SL-graph G as $|G| \stackrel{\text{def}}{=} |\alpha(G)|$.

Example 5. Graph (b) of Figure 1 corresponds to the assertion $(x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_6 \wedge x_1 \neq x_4 \wedge x_5 \neq x_7; x_1 \mapsto x_4 * \mathsf{ls}(x_4, x_5) * \mathsf{ls}(x_4, x_7))$, where we have omitted superfluous equalities.

We now give some technical definitions about paths in SL-graphs. Given a relation $E \subseteq V \times V$, a *v-w path in E of length n* is a sequence of nodes $\pi : v_1 \cdots v_{n+1}$ such that $v_1 = v$, $v_{n+1} = w$ and $(v_i, v_{i+1}) \in E$ for all $1 \leq i \leq n$. We write $|\pi|$ to denote the length of π. The *edges traversed by* π is defined as $edges(\pi) \stackrel{\text{def}}{=} \{(v_i, v_{i+1}) : 1 \leq i \leq n\}$. Two paths π_1, π_2 are *distinct* if $edges(\pi_1) \cap edges(\pi_2) = \emptyset$. If $v \neq w$, we call a v-w path *loop-free* if $v_i \neq v_j$ for all $1 \leq i \neq j \leq n + 1$. We write $v \rightsquigarrow_p w$, $v \rightsquigarrow_l w$ and $v \rightsquigarrow_{p,l} w$ if there exists a v-w path in E_p, E_l respectively $E_{p,l}$. Moreover, we write $v \rightarrow_p w$, $v \rightarrow_l w$ and $v \rightarrow_{p,l} w$ if $(v, w) \in E_p$, $(v, w) \in E_l$ respectively $(v, w) \in E_{p,l}$. Given a set of edges E, $V(E)$ denotes the set $V(E) \stackrel{\text{def}}{=} \{v : \exists w.(v, w) \in E \text{ or } (w, v) \in E\}$. As usual, E^* denotes the reflexive and transitive closure of E. For $e = (v, w) \in E$, we define $E^*(e) \stackrel{\text{def}}{=} \{u : (w, u) \in E^*\} \cup \{v\}$.

The challenging aspect in giving a polynomial time algorithm to decide entailment is that our logic is *non-convex*. As has already been observed in [2], given $\alpha = (y \neq z; \mathsf{ls}(x, y) * \mathsf{ls}(x, z))$, for any model $\mathcal{I}$ of α we have $\mathcal{I} \models (x = y; \epsilon)$ or $\mathcal{I} \models (x = z; \epsilon)$. However there are models $\mathcal{I}_1, \mathcal{I}_2$ of α such that $\mathcal{I}_1 \not\models (x = y; \epsilon)$ and $\mathcal{I}_2 \not\models (x = z; \epsilon)$. Non-convexity often makes computing entailment coNP-hard for logics that contain predicates for describing reachability relations on graphs, e.g., in fragments of XPath or description logics [13, 10]. However, in our

SL fragment we obtain tractability through the SL-graph normal form we develop in the next section and the fact that variable names only occur at exactly one node in an SL-graph, which fully determines a graph homomorphism if it exists.

3 A Normal Form of SL-Graphs

In this section, we show that given an assertion α we can compute in polynomial time an SL-graph G in a *normal form* such that $\alpha \equiv \alpha(G)$. This normal form serves three purposes: First, it makes implicit equalities and disequalities from α explicit. Second, an SL-graph in normal form has the structural property that if there is a loop-free path between two distinct vertices then there is exactly one such path. Third, any SL-graph $G \neq \bot$ in normal form can be transformed into an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \alpha(G)$, thus showing that satisfiability in our SL fragment is in polynomial time.

First, we show how given a pure formula ϕ we can construct a corresponding graph G_ϕ such that $(\phi, \epsilon) \equiv \alpha(G_\phi)$. Let $\{x_1, \ldots, x_m\} \subseteq \mathsf{Vars}$ be the set of all variables occurring in ϕ, and let $\{[e_1], \ldots, [e_n]\}$ be the set of all equivalence classes of variables induced by ϕ, i.e., $x, y \in [e_i]$ if, and only if, ϕ implies $x = y$. Let $V_b \overset{\text{def}}{=} \{v_1, \ldots, v_n\} \subseteq V$; $\ell(x) \overset{\text{def}}{=} v_i$ if, and only if, $x \in [e_i]$; and $E_d \overset{\text{def}}{=} \{\{v_i, v_j\} : \exists x, y \in \mathsf{Vars}. x \in [e_i], y \in [x_j]$ and $x \neq y$ occurs in $\phi\}$. If there is a singleton set in E_d then set $G_\phi \overset{\text{def}}{=} \bot$, otherwise $G_\phi \overset{\text{def}}{=} (V_b, \emptyset, \emptyset, \emptyset, E_d, \ell)$. The following lemma can now easily be verified.

Lemma 6. *Let ϕ be a pure formula. There exists a polynomial time computable SL-graph G_ϕ such that $\alpha(G_\phi) \equiv (\phi, \epsilon)$.*

Next, we show how to deal with spatial assertions. When processing spatial assertions and transforming SL-graphs into normal form, we need to manipulate SL-graphs. The two operations we perform on them are *merging nodes* and *removing edges*. Due to space constraints, we relegate details of the algorithms that implement these operations to the extended version of this paper [6].

Algorithm $\textsc{Merge}(G, v, w)$ takes an SL-graph G as input and merges the node w into node v by adding all labels from w to the labels of v and appropriately updating E_l, E_p and E_d. Moreover, the algorithm makes sure that if either $v \in V_r$ or $w \in V_r$ then $v \in V_r$ in the returned graph. If both $v, w \in V_r$ or $\{v, w\} \in E_d$ then $\textsc{Merge}(G, v, w)$ returns $\bot$. Thus, $\textsc{Merge}$ is characterised as follows: If $\alpha(G) = (\phi; \sigma)$, $v, w \in V_{b,r}$, $x = \mathit{var}(v)$ and $y = \mathit{var}(w)$ then $\alpha(\textsc{Merge}(G, v, w)) \equiv (\phi \wedge x = y; \sigma)$.

Algorithm $\text{L}\textsc{Remove}(G, (v, w))$ takes an SL-graph G as input and removes the l-edge (v, w) from G. Likewise, $\text{P}\textsc{Remove}(G, (v, w))$ removes a p-edge from G and, if necessary, moves v from V_r to V_b. Both algorithms can be characterised as follows: If $\alpha(G) = (\phi; \sigma * \mathsf{ls}(x, y))$, $v, w \in V_{b,r}$, $x = \mathit{var}(v)$ and $y = \mathit{var}(w)$ then $\alpha(\text{L}\textsc{Remove}(G, (v, w))) \equiv (\phi; \sigma)$. If $\alpha(G) = (\phi; \sigma * x \mapsto y))$ then $\alpha(\text{P}\textsc{Remove}(G, (v, w))) \equiv (\phi; \sigma)$, where v, w, x and y are as before. As an

Algorithm 1 Reduce

```
Require: $G$
  while $G$ is not reduced do
    case split on violated condition at node $v$
    // conditions are as in Table 2
    // node names below refer in each case to the corresponding case in Lemma 13
    case (i): return $\perp$
    case (ii): $G = \textsc{LReMerge}(G, (v, w'))$
    case (iii): $G = \textsc{LReMerge}(G, (v, w''))$
    case (iv): $G = \textsc{Merge}(G', v, w)$
  end while
  return $G$
```

abbreviation, we introduce $\textsc{LReMerge}(G, (v, w))$ and $\textsc{PReMerge}(G, (v, w))$ which first remove an l- respectively p-edge (v, w) from G and then merge w into v.

Finally, Algorithm $\textsc{Apply}(G, \sigma)$ takes an SL-graph G and a single spatial assertion $\sigma \in \{x \mapsto y, \mathsf{ls}(x, y)\}$ as input and outputs an SL-graph G' such that if $\alpha(G) = (\phi; \sigma')$ then $\alpha(G') \equiv (\phi; \sigma' * \sigma)$. Again, the concrete algorithm can be found in the extended version due to space limitations, but it is not difficult to construct such an algorithm that runs in polynomial time. Some extra care has to be taken if an l-edge is added that is already present in G, since $(\phi; \sigma * \mathsf{ls}(x, y) * \mathsf{ls}(x, y)) \equiv (\phi \wedge x = y; \sigma * \mathsf{ls}(x, y))$. By combining all algorithms considered in this section, we obtain the following lemma.

Lemma 7. *Let α be an SL-graph. Then there exists a polynomial-time algorithm that computes an SL-graph G such that $\alpha \equiv \alpha(G)$.*

We now move towards defining the normal form of an SL-graph and show that any SL-graph can be transformed into one in normal form such that their corresponding assertions are equivalent. A key concept of the normal form is that of a persistent set of edges.

Definition 8. *Let G be an SL-graph, a set of edges $E \subseteq E_{p,l}$ is* persistent *if $V(E) \cap V_r \neq \emptyset$ or there are $v, w \in V(E)$ such that $\{v, w\} \in E_d$.*

For example, let e_1 be the l-edge from x_4 to x_5 and e_2 the l-edge from x_4 to x_7 of graph (a) in Figure 1. Neither $\{e_1\}$ nor $\{e_2\}$ is persistent, but $\{e_1, e_2\}$ is as there is a d-edge between x_5 and x_7. Intuitively, the idea behind the definition is as follows: Suppose we are given an SL-graph G with $(v, w) \in E_l$ such that $E = E^*_{p,l}(v, w)$ is persistent. Then in any model $\mathcal{I}$ of $\alpha(G)$ for $v' = \ell^{\mathcal{I}}(var(v))$, we have $v' \in V_r^{\mathcal{I}}$ since v' must have an outgoing p-edge as the persistence property enforces that there is a p-edge in E or that not all variable names occurring in E are mapped to v' in $\mathcal{I}$. Moreover, if v has a further outgoing l-edge (v, w') then $\ell^{\mathcal{I}}(var(w')) = v$ since v can only have one outgoing p-edge in $\mathcal{I}$. For graph (a) in Figure 1, this means that x_6 becomes equivalent to x_4 in any model of

(i) if $v \in V_r$ then $|E_p(v)| = 1$
(ii) if $v \rightarrow_{p,l} w$ such that $E^*_{p,l}(v, w)$ is persistent then $E_l(v) \subseteq \{w\}$
(iii) if $v \rightarrow_l w_1$ and $v \rightarrow_l w_2$ such that $E^*_{p,l}(v, w_1) \cup E^*_{p,l}(v, w_2)$ is persistent then $E_l(v) \subseteq \{w_1, w_2\}$
(iv) there are no distinct loop-free v-w paths π_1, π_2 in E_l.

Table 2. Conditions for an SL-graph G to be reduced.

the corresponding SL-formula. Thus persistency allows us to make some implicit equalities in G explicit.

Definition 9. *An SL-graph G is reduced if $G = \perp$ or if it fulfils the conditions in Table 2.*

The definition of a reduced SL-graph is the first step towards the normal form of SL-graphs. Table 2 consists of four conditions, and the idea is that if any of those conditions is violated by an SL-graph G then we can make some implicit facts explicit. Clearly, if (i) is violated then $\alpha(G)$ is unsatisfiable as the spatial part of $\alpha(G)$ consists of a statement of the form $x \mapsto y * x \mapsto z$. If (ii) or (iii) is violated then by the previous reasoning any further outgoing l-edge can be collapsed into v. Condition (iv) contributes to making sure that between any two different nodes there is at most one loop-free path, as can be seen by the following lemma.

Lemma 10. *Let $G \neq \perp$ be a reduced SL-graph, v, w be distinct nodes in $V_{b,r}$ and $\pi : v \rightsquigarrow_{l,r} w$ a loop-free path. Then π is the unique such loop-free path.*

Proof. To the contrary, assume that there are two different loop-free v-w paths π_1, π_2. Then there are nodes v', w' such that there are distinct v'-w' paths π'_1 and π'_2 that are segments of π_1 respectively π_2, where at least one of π_1 or π_2 is of non-zero length. If $v' = w'$ then this contradicts to π_1 or π_2 being loop-free. Thus, assume $v' \neq w'$. If both π'_1, π'_2 are l-paths then this contradicts to G being reduced, as condition (iv) is violated. Otherwise, if π'_1 reaches a red node then $edges(\pi'_1)$ is persistent and hence v' has one outgoing edge, contradicting to π'_1 and π'_2 being distinct. The case when π'_2 reaches a red node is symmetric.

It is easy to see that deciding whether a graph G is reduced can be performed in polynomial time in $|G|$. In order to transform an arbitrary SL-graph into a reduced SL-graph, Algorithm Reduce just checks for a given input G if any condition from Table 2 is violated. If this is the case, the algorithm removes edges and merges nodes, depending on which condition is violated, until G is reduced. We will subsequently prove Reduce to be correct. First, we provide two technical lemmas that will help us to prove correctness. They allow us to formalise our intuition about persistent sets of edges. Due to space constraints, we omit the proof of the following lemma.

Lemma 11. *Let G be an SL-graph and $v, w, w' \in V_{b,r}$ such that $x = var(v)$, $y = var(w)$, $v \rightsquigarrow_l w$, and let $\mathcal{I}$ be a model of $\alpha(G)$. Then the following holds:*

(i) *if $\ell^{\mathcal{I}}(y) \in V_r^{\mathcal{I}}$ then $\ell^{\mathcal{I}}(x) \in V_r^{\mathcal{I}}$; and*
(ii) *if $v \rightsquigarrow_l w'$ and $\{w, w'\} \in E_d$ then $\ell^{\mathcal{I}}(x) \in V_r^{\mathcal{I}}$.*

Lemma 12. *Let $\alpha = (\phi, \sigma)$ and $x \in \mathsf{Vars}$ be such that for all models $\mathcal{I}$ of α, $\ell^{\mathcal{I}}(x) \in V_r^{\mathcal{I}}$. Then for all $y \in \mathsf{Vars}$ and $\alpha' = (\phi, \sigma * \mathsf{ls}(x, y)), \alpha'' = (\phi \wedge x = y, \sigma)$, we have $\alpha' \equiv \alpha''$.*

Proof. We clearly have that $\alpha'' \models \alpha'$. For the other direction, let $\mathcal{I}'$ be a model of α'. By definition, there are $\mathcal{I}_1, \mathcal{I}_2$ such that $\mathcal{I}' = \mathcal{I}_1 * \mathcal{I}_2$, $\mathcal{I}_1 \models (\phi; \sigma)$ and $\mathcal{I}_2 \models (\phi; \mathsf{ls}(x, y))$. By assumption, $\ell^{\mathcal{I}_1}(x) \in V_r^{\mathcal{I}_1}$ and hence $\ell^{\mathcal{I}_2}(x) \notin V_r^{\mathcal{I}_2}$. Consequently, $\ell^{\mathcal{I}_2}(x) = \ell^{\mathcal{I}_2}(y)$. Hence $\ell^{\mathcal{I}'}(x) = \ell^{\mathcal{I}'}(y)$, which yields $\mathcal{I}' \models (\phi \wedge x = y; \sigma)$.

We are now prepared to show the correctness of Reduce. Each case in the lemma below captures a violated condition from Table 2 and shows that the manipulation performed by Reduce is sound and correct.

Lemma 13. *Let G be an SL-graph,*

(i) *if there is $v \in V_r$ such that $|E_p(v)| > 1$ then $\alpha(G)$ is unsatisfiable;*
(ii) *if there are $v, w, w' \in V_{b,r}$, $x, y \in \mathsf{Vars}$ such that $v \to_{p,l} w$, $v \to_l w'$, $x = var(v)$, $y = var(w')$, $E^*_{p,l}(v, w)$ is persistent and $\alpha(G) = (\phi, \sigma * \mathsf{ls}(x, y))$ then $\alpha(G) \equiv (\phi \wedge x = y; \sigma)$;*
(iii) *if there are $v, w, w', w'' \in V_{b,r}$, $x, y \in \mathsf{Vars}$ such that $v \to_l w$, $v \to_l w'$, $v \to_l w''$, $x = var(v)$, $y = var(w'')$, $E^*_{p,l}(v, w) \cup E^*_{p,l}(v, w')$ is persistent and $\alpha(G) = (\phi, \sigma * \mathsf{ls}(x, y))$ then $\alpha(G) \equiv (\phi \wedge x = y; \sigma)$;*
(iv) *if there are $v, w \in V_b$, $x, y \in \mathsf{Vars}$ such that $x = var(v)$, $y = var(w)$, $\alpha(G) = (\phi, \sigma)$ and there are distinct loop-free v-w l-paths π_1, π_2 in E_l then $\alpha(G) \equiv (\phi \wedge x = y; \sigma)$.*

Proof. Case (i): Let $x = var(v)$; we have that there are $y, z \in \mathsf{Vars}$ such that $(\phi; \sigma * x \mapsto y * x \mapsto z)$, which clearly is unsatisfiable.

Case (ii): We show that for all models $\mathcal{I}$ of $\alpha(G)$, $\ell^{\mathcal{I}}(x) \in V_r$. The statement then follows from Lemma 12. If there is $u \in V(E^*_{p,l}(v, w)) \cap V_r$ then by Lemma 11(i) we have $x \in V_r^{\mathcal{I}}$. Otherwise, if there are $u, u' \in V(E^*_{p,l}(v, w))$ such that $\{u, u'\} \in E_d$ then Lemma 11(ii) gives $x \in V_r^{\mathcal{I}}$.

Case (iii): Again, we show that for all models $\mathcal{I}$ of $\alpha(G)$, $\ell^{\mathcal{I}}(x) \in V_r$. The statement then follows from Lemma 12. It is sufficient to consider the case in which there are $u, u' \in V_{b,r}$ such that $w \rightsquigarrow_l u$, $w \rightsquigarrow_l u'$ and $\{u, u'\} \in E_d$ as all other cases are subsumed by (ii). But then, Lemma 11(ii) again yields $x \in V_{b,r}^{\mathcal{I}}$.

Case (iv): Let $\pi_1 = vw_1 \cdot \pi_1'$ and $\pi_2 = vw_2 \cdot \pi_2'$ be v-w paths. Thus, $w_1 \neq w_2$ and hence $m \overset{\text{def}}{=} |\pi_1| + |\pi_2| \geq 3$. We show the statement by induction on m. For $m = 3$, the statement follows from a similar reasoning as in Lemma 12. For the induction step, let $m > 3$ and $\mathcal{I}$ be model of $\alpha(G)$. Let $y_1 = var(w_1)$ and $y_2 = var(w_2)$, we have that $\alpha(G) = (\phi; \sigma * \mathsf{ls}(x, y_1) * \mathsf{ls}(x, y_2))$ and consequently $\mathcal{I} \models \sigma * \mathsf{ls}^{n_1}(x, y_1) * \mathsf{ls}^{n_2}(x, y_2)$ for some $n_1, n_2 \in \mathbb{N}$. If $n_1 = 0$ then $\mathcal{I} \models G'$, where $G' = \text{LReMerge}(G, (v, w_1))$ and the induction hypothesis yields $\ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(y)$. The case $n_2 = 0$ follows symmetrically.

Lemma 14. *Let G, G' be SL-graphs such that $G' = \textsc{Reduce}(G)$. Then G' is reduced and $\alpha(G) \equiv \alpha(G')$. Moreover,* Reduce *runs in polynomial time on any input G.*

Proof. Clearly, Reduce only returns graphs that are reduced. Moreover, Lemma 13 shows that in every iteration equivalent graphs are generated and hence $\alpha(G) \equiv \alpha(G')$. Regarding the complexity, checking if G is reduced can be performed in polynomial time in $|G|$. Removing edges and merging nodes in the **while** body can also be performed in polynomial time. Moreover, the size of G strictly decreases after each iteration of the **while** body. Hence the **while** body is only executed a polynomial number of times.

A nice property of reduced SL-graphs is that they allow to easily construct a model of their corresponding SL-formulae.

Lemma 15. *Let $G \neq \bot$ be a reduced SL-graph and $v, w \in V_{b,r}$ such that $v \neq w$. Then $\alpha(G)$ has a model $\mathcal{I}$ such that $\ell^{\mathcal{I}}(var(v)) \neq \ell^{\mathcal{I}}(var(w))$ and for all $x, y \in \mathsf{Vars}$, $\ell(x) = \ell(y)$ implies $\ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(y)$.*

Proof. We sketch how G can iteratively be turned into a desired model $\mathcal{I}$. Suppose w is reachable from v and let π be the loop-free path from v to w. First, we replace any l-edge occurring on π by *two* consecutive p-edges. For all nodes $v' \neq w$ along π that have further outgoing l-edges, we merge all nodes reachable via l-paths from v' into v' and remove the connecting l-edges. If v is reachable from w via a loop-free path π', we apply the same procedure to π'. Finally, we iterate the following procedure: if there is a node u with more than one outgoing l-edge, we fix an l-edge e and merge all nodes reachable from u via the remaining l-edges different from e into u and remove the connecting l-edges. We then replace e with two new consecutive p-edges. Once this procedure has finished, we obtain an SL-graph containing no l-edges that can be turned into an interpretation $\mathcal{I}$. It is easily checked that $\mathcal{I}$ is a model of $\alpha(G)$ and $\ell^{\mathcal{I}}(var(v)) \neq \ell^{\mathcal{I}}(var(w))$.

Theorem 16. *Satisfiability of SL-formulae is decidable in polynomial time.*

Remark 17. When we expand l-edges in the proof of Lemma 15, we replace them by two consecutive p-edges. When we consider entailment in the next section, this will make sure that we obtain a model in which $\mathsf{ls}(x, y)$ holds, but $x \mapsto y$ does not hold. This corresponds to the observation made in [2] that in order to find a counter-model of an entailment, each l-edge has to be expanded at most to length two.

Finally, we can now define our normal form. An SL-graph is in normal form if it is reduced and if its set of disequalities is maximal. Note that in particular any interpretation $\mathcal{I}$ is an SL-graph in normal form.

Definition 18. *Let G be an SL-graph. Then G is in* normal form *if G is reduced and for all $v, w \in V_{b,r}$ such that $\alpha(G) = (\phi; \sigma)$, $x = var(v)$ and $y = var(w)$, whenever $(\phi \wedge x = y; \sigma)$ is unsatisfiable then $\{v, w\} \in E_d$.*

Proposition 19. *For any SL-formula α, there exists a polynomial time computable SL-graph G in normal form such that $\alpha \equiv \alpha(G)$.*

Proof. Given an assertion $\alpha = (\phi; \sigma)$, by Lemma 7 we can construct an SL-graph G' such that $\alpha(G') \equiv \alpha$. Applying REDUCE to G' yields a reduced graph G'' such that $\alpha(G') \equiv \alpha(G'')$. In order to bring G'' into normal form, we check for each of the polynomially many pairs $v, w \in V_{b,r}$ if REDUCE returns $\bot$ on input MERGE(G'', v, w). If this is the case, we add $\{v, w\}$ to E_d, which finally gives us the desired graph G. As argued before, all constructions can be performed in polynomial time.

We close this section with an example. Graph (b) in Figure 1 is in normal form and obtained from the graph (a) by applying REDUCE. Graph (a) violates condition (iii) as $\{(\ell(x_4), \ell(x_5)), (\ell(x_4), \ell(x_7)\}$ is persistent, which results in REDUCE merging x_6 into x_4. Moreover, the graph also violates condition (iv) since there are two distinct l-paths from x_1 to x_3. Hence, REDUCE merges x_1 and x_3 and then removes all newly obtained outgoing l-edges from x_3 due to a violation of condition (ii). Finally, $\{(\ell(x_3), \ell(x_4))\}$ is added to E_d in order to obtain graph (b) as merging the nodes x_3 and x_4 and applying REDUCE results in an inconsistent graph.

4 Computing Entailment via Homomorphisms between SL-Graphs in Normal Form

In this section, we show that entailment between SL-formulae can be decided by checking the existence of a graph homomorphism between their corresponding SL-graphs in normal form. Throughout this section, we will assume that all SL-formulae considered are satisfiable and all SL-graphs $G \neq \bot$, since deciding entailment becomes trivial otherwise, and checking for satisfiability can be done in polynomial time.

A homomorphism is a mapping between the nodes of two SL-graphs that, if it exists, preserves the structure of the source graph in the target graph. In the definition of a homomorphism, we make use of the property of SL-graphs in normal form that between any disjoint nodes there is at most one loop-free path connecting the two nodes. For nodes $v \neq w$, we denote this path by $\pi(v, w)$ if it exists. If $v = w$ then $\pi(v, w)$ is the zero-length path $\pi(v, w) \overset{\text{def}}{=} v$.

Definition 20. *Let G, G' be SL-graphs in normal form. A mapping $h : V_{b,r} \to V'_{b,r}$ is a* homomorphism *from G to G' if the homomorphism conditions from Table 3 are satisfied.*

Given a mapping h, it is easy to see that checking whether h is a homomorphism can be performed in polynomial time in $|G| + |G'|$. The goal of this section is to prove the following proposition, which gives us the relationship between homomorphisms and entailment.

(i)	$vars(v) \subseteq vars(h(v))$
(ii)	if $\{v, w\} \in E_d$ then $\{h(v), h(w)\} \in E'_d$
(iii)	if $v \rightarrow_p w$ then $h(v) \rightarrow'_p h(w))$
(iv)	if $v \rightarrow_l w$ then $h(v) \rightsquigarrow'_{p,l} h(w)$
(v)	for all $v_1 \rightarrow_{p,l} w_1$ and $v_2 \rightarrow_{p,l} w_2$ such that $(v_1, w_1) \neq (v_2, w_2)$, $edges(\pi(h(v_1), h(w_1))) \cap edges(\pi(h(v_2), h(w_2))) = \emptyset$
(vi)	if $v, w \in V_r$ and $v \neq w$ then $h(v) \neq h(w)$

Table 3. Conditions for a homomorphism h from G to G'.

Proposition 21. *Let G, G' be SL-graphs in normal form. Then $\alpha(G') \models \alpha(G)$ if, and only if, there exists a homomorphism h from G to G'.*

Before we begin with formally proving the proposition, let us discuss its validity on an intuitive level. Suppose there is a homomorphism from G to G'. Condition (i) makes sure that for any node v of G its image under h is labelled with at least the same variables. If this were not the case, we could easily construct a counter-model of $\alpha(G')$ disproving entailment. Likewise, condition (ii) ensures that whenever two nodes are required to be not equivalent, the same is true for the two nodes under the image of h. Since G' is in normal form, merging the two nodes in the image of h would otherwise be possible since E'_d is maximal. Condition (iii) requires that whenever there is a p-edge between any two nodes v, w, such an edge also exists in G'. Again, it is clear that if this were not the case we could construct a counter-model $\mathcal{I}$ of $\alpha(G')$ such that there is no p-edge between $\ell^{\mathcal{I}}(var(v))$ and $\ell^{\mathcal{I}}(var(w))$. Condition (iv) is of a similar nature, but here we allow that there is a whole path between $h(v)$ and $h(w)$. In condition (v), we require that the paths obtained from the image of two disjoint edges do not share a common edge in G'. If this were the case, we could construct a model of $\alpha(G')$ in which separation is violated. Finally, condition (vi) makes sure that no two different nodes from V_r are mapped to the same node. This condition is needed to handle p-edges of the form (v, v), which may not be covered by condition (v). We now proceed with formally proving Proposition 21. First, the following lemma shows the relationship between models and homomorphisms and that homomorphisms can be composed.

Lemma 22. *Let G, G', G'' be SL-graphs in normal form and $\mathcal{I}$ an interpretation. Then the following holds:*

(i) let $h : V_{b,r} \rightarrow V^{\mathcal{I}}_{b,r}$ be such that for all $v \in V_{b,r}$, $h(v) \overset{def}{=} \ell^{\mathcal{I}}(var(v))$; then $\mathcal{I} \models \alpha(G)$ if, and only if, h is a homomorphism from G to $\mathcal{I}$; and

(ii) given homomorphisms h', h'' from G' to G respectively G'' to G'; then $h \overset{def}{=} h'' \circ h'$ is a homomorphism from G'' to G.

The proof of the lemma is rather technical but not difficult and deferred to the extended version of this paper. Proposition 21 now is a consequence of the following lemma. Note that the homomorphism is fully determined by G and G'.

Lemma 23. *Let G, G' be SL-graphs in normal form and let $h : V_{b,r} \to V'_{b,r}$ be defined as $h(v) \overset{def}{=} \ell'(var(v))$ for all $v \in V_{b,r}$. Then $\alpha(G') \models \alpha(G)$ if, and only if, h is a homomorphism from G to G'.*

Proof. ("$\Leftarrow$") Let h be a homomorphism from G to G' and $\mathcal{I}$ be such that $\mathcal{I} \models \alpha(G')$. By Lemma 22(i), there exists a homomorphism h' from G' to $\mathcal{I}$. By Lemma 22(ii), $h'' \overset{\text{def}}{=} h' \circ h$ is a homomorphism from G to $\mathcal{I}$. Consequently, Lemma 22(i) yields $\mathcal{I} \models \alpha(G)$.

("$\Rightarrow$") Due to space constraints, we defer the full proof of this direction to the appendix. The direction is shown via the contrapositive by constructing a counter-model depending on which homomorphism condition from Table 3 is violated, as discussed earlier in this section.

We can now combine all results of this paper so far. Given satisfiable SL-formulae α and α', by Proposition 19 we can compute in polynomial time SL-graphs G and G' in normal form such that $\alpha \equiv \alpha(G)$ and $\alpha' \equiv \alpha(G')$. Next, we can compute in polynomial time a mapping h from $\alpha(G')$ to $\alpha(G)$ and check in polynomial time whether h is a homomorphism. By the previous lemma, this then is the case if, and only if, $\alpha \models \alpha'$.

Theorem 24. *Entailment between SL-formulae is decidable in polynomial time.*

An example of a homomorphism can be found in Figure 1. The arrows from graph (c) to graph (b) depict a homomorphism witnessing an entailment between the corresponding formulae of the graphs.

As promised in Section 2, we are now going to briefly discuss the differences between the semantic models in [2] and this paper. In [2], $\mathcal{I} \models \alpha = (\phi; \sigma)$ if, and only if, $\mathcal{I} \models \phi$ and $\mathcal{I} \models \sigma$, i.e., the semantics is non-intuitionistic. Informally speaking, in our semantics models can contain more red nodes than actually required. It is not difficult to see that the concept of SL-graphs in normal form can be adopted to non-intuitionistic semantics, in fact most proofs carry over straight forwardly. However, the homomorphism conditions need some adjustments. One basically needs to add extra conditions that ensure that when h is a homomorphism from G to G', all edges from G' are covered by h. These extra conditions ensure that no model of $\alpha(G')$ can contain extra red nodes that, informally speaking, do not get used up by $\alpha(G)$. Moreover, some further adjustments have to be made in order to deal correctly with precise list semantics. The details are messy and deferred to a full version of this paper.

5 Syntactic Extensions Leading to Intractability

As stated in Section 2, due to the non-convexity, it is rather surprising that entailment in our fragment is decidable in polynomial time. In this section, we briefly discuss syntactic extensions that render satisfiability or entailment intractable. It turns out that even small syntactic extensions make computing entailment intractable.

First, we consider additional Boolean connectives in pure and spatial formulae. Allowing disjunction in pure formulae with the standard semantics makes entailment coNP-hard, since implication between Boolean formulae is coNP-hard. Less obviously, allowing conjunction in spatial assertions makes satisfiability NP-hard and thus entailment coNP-hard. To be more precise, suppose we allow assertions of the form $\alpha = (\phi; \sigma_1 \wedge \sigma_2)$, where σ_1 and σ_2 are spatial formulae and $\mathcal{I} \models (\phi; \sigma_1 \wedge \sigma_2)$ if, and only if, $\mathcal{I} \models (\phi; \sigma_1)$ and $\mathcal{I} \models (\phi; \sigma_2)$. We reduce from graph three colourability (3Col), which is known to be NP-complete [8]. Given an instance $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of 3Col with $\mathcal{V} = \{v_1, \ldots, v_n\}$, we construct an assertion α such that $\mathcal{G}$ can be coloured with three colours if, and only if, α is satisfiable. We set $\alpha \overset{\text{def}}{=} (\phi, \sigma)$, where $\phi \overset{\text{def}}{=} \bigwedge_{(v_i,v_j)\in\mathcal{E}} x_i \neq x_j$ and $\sigma \overset{\text{def}}{=} y_1 \mapsto y_2 * y_2 \mapsto y_3 \wedge \bigwedge_{v_i \in \mathcal{V}} \mathsf{ls}(y_1, x_i) * \mathsf{ls}(x_i, y_3)$. Let us sketch the correctness of our reduction. The first conjunct of σ ensures that any model of α contains a list of three nodes that are successively labelled with the variable names y_1, y_2 and y_3. The remaining conjuncts enforce that for any $v_i \in \mathcal{V}$, some y_j-node is additionally labelled with the variable name x_i. Our intention is that y_j is additionally labelled with x_i in a model of α if v_i is coloured with colour j in a three-colouring induced by that model. We use ϕ to enforce that that two labels x_i, x_k are not placed on the node labelled with the same y_j if v_i and v_k are adjacent in $\mathcal{G}$, i.e., they must have a different colour in the induced three colouring. Hence $\mathcal{G}$ can be three coloured if, and only if, α is satisfiable. The coNP-hardness of entailment then follows from the fact that α is satisfiable if, and only if, $\alpha \not\models (x \neq x; \epsilon)$.

Finally, we briefly discuss allowing existentially quantified variables in assertions. An example of such an assertion is $\alpha = \exists y.(x \neq y; x \mapsto y)$, where $\mathcal{I} \models \alpha$ if $\mathcal{I}$ can be extended to $\mathcal{I}'$ in which some node of $\mathcal{I}$ is labelled with y such that $\mathcal{I}' \models (x \neq y; x \mapsto y)$. It is easily seen that satisfiability in this extended fragment is still decidable in polynomial time by just dropping the existential quantifier. However, it follows from recent results that entailment becomes coNP-hard [9].

6 Conclusion

In this paper, we have studied the complexity of entailment in a fragment of separation logic that includes pointers and linked lists. Despite the non-convexity of this logic, we could show that entailment is computable in polynomial time. To this end, we showed that for any SL-formula we can compute in polynomial time a corresponding SL-graph in a particular normal form which has an equivalent corresponding SL-formula. Moreover, we showed that deciding entailment between two SL-formulae then reduces to checking for the existence of a homomorphism between their associated SL-graphs in normal form. A key advantage was that the homomorphism, if it exists, is uniquely determined by the SL-graphs, and that checking the homomorphism conditions can be performed in polynomial time. Moreover, we discussed how minor syntactic extensions to our fragment lead to intractability of the entailment problem.

Acknowledgements. We would like to thank Nikos Gorogiannis for interesting and helpful discussions and the anonymous referees for helpful suggestions on how to improve the presentation of the paper.

References

1. Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano, Peter O'Hearn, Thomas Wies, and Hongseok Yang. Shape analysis for composite data structures. In *CAV'07*, pages 178–192. Springer, 2007.
2. Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A Decidable Fragment of Separation Logic. In *FSTTCS'04*, pages 97–109. Springer, 2004.
3. Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO'05*, pages 115–137. Springer, 2005.
4. Cristiano Calcagno, Dino Distefano, Peter W. OHearn, and Hongseok Yang. Space invading systems code. In *Logic-Based Program Synthesis and Transformation*, pages 1–3. Springer, 2009.
5. Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *FSTTCS'01*, pages 108–119. Springer, 2001.
6. Byron Cook, Christoph Haase, Joël Ouaknine, Matthew Parkinson, and James Worrell. Tracatable reasoning in a fragment of separation logics (full version). Technical report, University of Oxford, 2011. Available on-line: http://www.cs.ox.ac.uk/people/christoph.haase/sl.pdf.
7. Dino Distefano and Matthew Parkinson. jstar: towards practical verification for java. In *OOPSLA'08*, pages 213–226. ACM, 2008.
8. Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.
9. Nikos Gorogiannis, Max Kanovich, and Peter O'Hearn. The complexity of abduction for separated heap abstraction. In *SAS'11*. Springer, 2011. To appear.
10. Christoph Haase and Carsten Lutz. Complexity of subsumption in the EL family of description logics: Acyclic and cyclic tboxes. In *ECAI'08*, pages 25–29. IOS Press, 2008.
11. Samin S. Ishtiaq and Peter W. O'Hearn. Bi as an assertion language for mutable data structures. In *POPL'01*, pages 14–26. ACM, 2001.
12. Bart Jacobs and Frank Piessens. The VeriFast program verifier. Technical Report 520, Department of Computer Science, Katholieke Universiteit Leuven, 2008.
13. Gerome Miklau and Dan Suciu. Containment and equivalence for an XPath fragment. In *PODS'02*, pages 65–76. ACM, 2002.
14. John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS'02*. IEEE Computer Society, 2002.