

A Decidable Fragment of Separation Logic

Josh Berdine¹, Cristiano Calcagno², and Peter O’Hearn¹

¹ Queen Mary, University of London {berdine,ohearn}@dcs.qmul.ac.uk

² Imperial College, London ccris@doc.ic.ac.uk

Abstract. We present a fragment of separation logic oriented to linked lists, and study decision procedures for validity of entailments. The restrictions in the fragment are motivated by the stylized form of reasoning done in example program proofs. The fragment includes a predicate for describing linked list segments (a kind of reachability or transitive closure). Decidability is first proved by semantic means: by showing a small model property that bounds the size of potential countermodels that must be checked. We then provide a complete proof system for the fragment, the termination of which furnishes a second decision procedure.

1 Introduction

Separation logic is a new approach to reasoning about programs that manipulate pointer structures [1]. The main advantage of the logic is the way it supports reasoning about different portions of heap which can be combined in a modular way using the separating conjunction operation. In this paper we present a fragment of separation logic and study decision procedures for validity of entailments.

These results are part of a bigger project that aims to provide algorithms and tools to transfer the simplicity of handwritten proofs with separation logic to an automatic setting. To make the task of automatic verification more feasible, we restrict our attention to structural integrity properties (like not following dangling pointers, preserving noncircularity of linked lists, not leaking memory), rather than full correctness. Moreover, we restrict the language by disallowing pointer arithmetic.

Even with these restrictions, the decidability questions are nontrivial. In particular, one of the most treacherous passes in pointer verification and analysis is *reachability*. To describe common loop invariants, and even some pre- and post-conditions, one needs to be able to assert that there is a path in the heap from one value to another; a fragment that cannot account for reachability in some way will be of very limited use. When we inquire about decidability we are then square up against the bugbear of transitive closure (reachability is the transitive closure of points-to); there are various decidable fragments of, say, the first-order logic of graphs, but for many of these decidability breaks if transitive closure is added.

So, a main technical challenge is to take on a form of reachability, in a way that fits with the separating conjunction (and the possibility of dangling pointers). We begin simply, with linked list structures only, instead of general

heap structures with arbitrary sharing. Our analysis can be adapted to certain kinds of tree structure, but we do not yet have a general picture of the kinds of inductive definitions that are amenable to the style of analysis presented here.

Our approach started by observing the stylized reasoning that was done in typical manual proofs using separation logic (e.g., [2–4]). For instance, we would often say “I have a list here, and another there occupying separate storage”, but never would we assert the negation of such a statement. Generally, in many examples that have been given, the assertions include a heap-independent, or pure, boolean condition, and a number of heap-dependent (or “spatial”) assertions separately conjoined. So, we consider a restricted fragment where the formulæ are of the form $\Pi \upharpoonright \Sigma$, where Π is a conjunction of equalities and inequalities and Σ is a separating conjunction of points-to facts and list segment remarks. We show the decidability of entailment between formulæ of this form.

In fact, two decision procedures are given. The first, a semantic procedure, is based on a “small model property”. In essence, we have designed the fragment so that formulæ do not admit any “unspecified” sharing, and then exploited separation logic’s local reasoning to capitalize on the absence of interference by avoiding case analysis on the possible interaction patterns between formulæ. The essential result, which fails for separation logic as a whole, is that when considering the possible models of our list segment predicate, no case analysis on the possible interference patterns is necessary, instead considering either the length zero or length two model immediately suffices. So decidability is achieved not through some brute force interference analysis, but by leveraging locality.

The second is a proof-theoretic procedure. It has the advantage of not generating the exponentially-many potential countermodels in every case, as the semantic procedure does. Also, this is the first complete proof theory that has been given for (a fragment of) separation logic. It is a candidate for extension to richer fragments (where we might not insist on decidability).

It is worth remarking on what is left out of the fragment. Although we are asking about the validity of entailments, entailment is not itself internalized with an implication connective; the additive and multiplicative implications (\rightarrow and \multimap) from BI are omitted. A hint of the computational significance of these omissions can be seen in the (easier) problem of model checking assertions (checking satisfaction). In earlier work it was shown that a fragment with points-to and nesting of \multimap and \rightarrow , but no list segment predicate, has model checking complexity PSpace-Complete [5]. Even just wrapping negations around the separating conjunction leads to PSpace-Complete model checking. In contrast, the model checking problem for the fragment of this paper, which goes further in that it considers list segments, is linear.

The fragment of this paper has been used in a prototype tool that checks properties of pointer programs. Typically in tools of this kind, the assertion language is closed under taking weakest preconditions of atomic commands. This is not the case for our fragment. However, it is possible to reduce entailments arising from weakest preconditions to entailments in our fragment, by way of a form of symbolic execution. Here we confine ourselves to the question of decid-

ability for the fragment, and leave a description of the symbolic execution phase to a future paper.

2 Fragment of Separation Logic

The fragment of separation logic we are concerned with is specified by restricting the assertion language to that generated by the following grammar:

$x, y, \dots \in \text{Variables}$	variables
$E ::= \text{nil} \mid x$	Expressions
$P ::= E=E \mid \neg P$	simple Pure formulæ
$\Pi ::= \text{true} \mid \Pi \wedge P$	Pure formulæ
$S ::= E \mapsto E \mid \text{ls}(E, E)$	simple Spatial formulæ
$\Sigma ::= \text{emp} \mid S * \Sigma$	Spatial formulæ
$A ::= P \mid \Pi \mid S \mid \Sigma \mid \Pi \dagger \Sigma$	formulæ

Note that we abbreviate $\neg(E_1=E_2)$ as $E_1 \neq E_2$, and use \equiv to denote “syntactic” equality of formulæ, which are considered up to symmetry of $=$ and permutations across \wedge and $*$, e.g. $\Pi \wedge P \wedge P' \equiv \Pi \wedge P' \wedge P$. We use notation treating formulæ as sets of simple formulæ, e.g., writing $P \in \Pi$ for $\Pi \equiv P \wedge \Pi'$ for some Π' .

Formulæ are interpreted as predicates on program States with a forcing relation, while expressions denote Values and depend only on the stack:³

$s, h \models A$	$\llbracket E \rrbracket \in \text{Stacks} \rightarrow \text{Values}$
$\text{Stacks} \stackrel{\text{def}}{=} \text{Variables} \rightarrow \text{Values}$	$\text{R-values} \stackrel{\text{def}}{=} \text{Values}$
$\text{Heaps} \stackrel{\text{def}}{=} \text{L-values} \stackrel{\text{fin}}{\rightarrow} \text{R-values}$	$\text{L-values} \stackrel{\text{def}}{\subset} \text{Values}$
$\text{States} \stackrel{\text{def}}{=} \text{Stacks} \times \text{Heaps}$	$\text{nil} \stackrel{\text{def}}{\in} \text{Values} \setminus \text{L-values}$

The semantics of the assertion language is shown in Table 1, while below we try to give some intuitive feel for the assertions and what sorts of properties are expressible with a few examples.

As always, a formula $S * \Sigma$ is true in states where the heap can be split into two separate parts (with disjoint domains) such that S is true in one part and Σ is true in the other. The unit of this conjunction is **emp**, which is true only in the empty heap. The only primitive spatial predicate is \mapsto , which describes individual L-values in the heap. So $10 \mapsto 42$ is true in the heap in which L-value 10 contains 42, and nothing else—the domain is the singleton $\{10\}$. Similarly, $x \mapsto 42$ asserts that whichever L-value the stack maps x to contains 42. In addition to the spatial (heap-dependent) part, formulæ also have a pure (heap-independent) part. So extending the last example, with $x=y \dagger x \mapsto 42$ we also assert that the stack maps x and y to equal R-values. Since the conjuncts of a $*$ formula must be true in disjoint heaps, $x=y \dagger x \mapsto \text{nil} * y \mapsto \text{nil}$ is unsatisfiable.

³ For a concrete instance of this model, take $\text{Values} = \mathbb{Z}$, $\text{L-values} = \mathbb{N} \setminus \{0\}$, $\text{nil} = 0$.

Table 1. Semantics of Assertion Language

	$\llbracket x \rrbracket s \stackrel{\text{def}}{=} s(x)$	$\llbracket \text{nil} \rrbracket s \stackrel{\text{def}}{=} \text{nil}$
$s, h \models E_1 = E_2$	$\stackrel{\text{def}}{\text{iff}} \llbracket E_1 \rrbracket s = \llbracket E_2 \rrbracket s$	
$s, h \models \neg P$	$\stackrel{\text{def}}{\text{iff}} s, h \not\models P$	
$s, h \models \text{true}$	always	
$s, h \models \Pi \wedge P$	$\stackrel{\text{def}}{\text{iff}} s, h \models \Pi \text{ and } s, h \models P$	
$s, h \models E_1 \mapsto E_2$	$\stackrel{\text{def}}{\text{iff}} h = [\emptyset \mid \llbracket E_1 \rrbracket s \rightarrow \llbracket E_2 \rrbracket s]$	
$s, h \models \text{ls}(E_1, E_2)$	$\stackrel{\text{def}}{\text{iff}} \text{there exists } n. s, h \models \text{ls}^n(E_1, E_2)$	
$s, h \models \text{ls}^0(E_1, E_2)$	$\stackrel{\text{def}}{\text{iff}} \llbracket E_1 \rrbracket s = \llbracket E_2 \rrbracket s \text{ and } h = \emptyset$	
$s, h \models \text{ls}^{n+1}(E_1, E_2)$	$\stackrel{\text{def}}{\text{iff}} \llbracket E_1 \rrbracket s \neq \llbracket E_2 \rrbracket s \text{ and}$ there exists $v \in \text{Values}. [s \mid x \mapsto v], h \models E_1 \mapsto x * \text{ls}^n(x, E_2)$ for $x \notin \text{fv}(E_1, E_2)$	
$s, h \models \text{emp}$	$\stackrel{\text{def}}{\text{iff}} h = \emptyset$	
$s, h \models S * \Sigma$	$\stackrel{\text{def}}{\text{iff}} \text{there exists } h_1 \perp h_2. h = h_1 * h_2 \text{ and } s, h_1 \models S \text{ and } s, h_2 \models \Sigma$	
$s, h \models \Pi \vdash \Sigma$	$\stackrel{\text{def}}{\text{iff}} s, h \models \Pi \text{ and } s, h \models \Sigma$	

The ls predicate describes segments of linked list structures in the heap: $\text{ls}(x, y)$ describes a list segment starting at the L-value denoted by x whose last link contains the value of y , which is a dangling pointer. That y is dangling is significant, as it precludes cycles. So $\text{ls}(x, x)$ describes the empty list segment, and is equivalent to emp . Were the endpoint not required to be dangling, then $\text{ls}(x, x)$ could describe cyclic lists containing x . Instead, a cyclic list is described for instance with $x \mapsto y * \text{ls}(y, x)$. For some further examples, $\text{ls}(x, \text{nil})$ describes “complete” lists, rather than segments. A list with an intermediate link can be expressed with $\text{ls}(x, y) * \text{ls}(y, \text{nil})$, two non-overlapping lists with $\text{ls}(x, \text{nil}) * \text{ls}(y, \text{nil})$, and two lists with a shared tail with $\text{ls}(x, z) * \text{ls}(y, z) * \text{ls}(z, \text{nil})$.

Our restriction to unary heap cells, and hence lists with links containing nothing but a pointer to the next link, is not significant and need not cause alarm: our development extends straightforwardly, all the formulæ just get longer.⁴

3 Decidability, Model-Theoretically

As mentioned earlier, our primary concern in this paper is deciding *validity* of entailments between formulæ in the fragment. That is, for entailments of the form $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$, we wish to check if for all $s, h. s, h \models \Pi \vdash \Sigma$ implies $s, h \models \Pi' \vdash \Sigma'$. Before getting stuck into decidability, we try to develop some intuition with a few examples.

⁴ While with binary heap cells, unrolling a ls involves generating a fresh variable, this is unproblematic for decidability in part due to Definition 10.

First trivially, anything entails itself, up to equalities: $x=y \wedge E=F \vdash x \mapsto E \vdash y \mapsto F$. As $\text{nil} \notin \text{L-values}$, $x \mapsto E \vdash x \neq \text{nil} \vdash x \mapsto E$. Also, since $*$ guarantees separation, spatial formulæ have implicit non-alias consequences: $x \mapsto E * y \mapsto F \vdash x \neq y \vdash x \mapsto E * y \mapsto F$. Explicit descriptions of list segments entail the inductive descriptions: $x=y \vdash \text{emp} \vdash \text{ls}(x, y)$ for length 0, $x \neq y \vdash x \mapsto y \vdash \text{ls}(x, y)$ for length 1, $x \neq y \wedge z \neq y \vdash x \mapsto z * z \mapsto y \vdash \text{ls}(x, y)$ for length 2, and $x \neq y \vdash x \mapsto z * \text{ls}(z, y) \vdash \text{ls}(x, y)$ for length “ $n + 1$ ”. All the inequalities in these examples are actually necessary: Since the ls predicate prohibits cycles in the consequent, there must be enough inequalities in the antecedent to guarantee acyclicity. Crucially, there are valid entailments which generally require induction to prove, such as appending a list segment and a list: $\text{ls}(x, z) * \text{ls}(z, \text{nil}) \vdash \text{ls}(x, \text{nil})$.

Before attacking entailment validity, we must consider formula satisfaction:

Lemma 1 (Satisfaction Decidable). *For given $s, h, \Pi \vdash \Sigma$, checking the satisfaction $s, h \models \Pi \vdash \Sigma$ is decidable.*

In fact, satisfaction checking is linear in the combined size of the model and the formula. For a given stack and heap, first we check the pure part of the formula against the stack in the obvious way. Then, to check the spatial part we start from the left and proceed as follows. If the first formula is a points-to, we just look to see if the evident singleton is in the heap; if not we report “no” and if so we remove it from the heap (or mark it) and continue. If the formula is a ls we simply try to traverse through the heap from the putative start until we get to the putative end (deleting cells as we go). If the traversal fails we report “no”, otherwise we continue on with the rest of the spatial part. When we get to the empty spatial formula we just check to see if we have the empty heap.

Informally, checking validity of entailments of the form $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$ is decidable because it suffices to consider finitely-many potential models of the antecedent. This small model property is captured primarily by:

Proposition 2. *The following rule is sound:*

$$\frac{\text{UNROLLCOLLAPSE} \quad \begin{array}{l} \Pi \wedge E_1 = E_2 \vdash \Sigma \vdash \Pi' \vdash \Sigma' \\ \Pi \wedge E_1 \neq E_2 \wedge x \neq E_2 \vdash E_1 \mapsto x * x \mapsto E_2 * \Sigma \vdash \Pi' \vdash \Sigma' \end{array}}{\Pi \vdash \text{ls}(E_1, E_2) * \Sigma \vdash \Pi' \vdash \Sigma'} \quad x \notin \text{fv}(\Pi, E_1, E_2, \Sigma, \Pi', \Sigma')$$

This rule says that to prove that a ls entails a formula, it suffices to check if the ls s of lengths zero and two entail the formula. That is, it eliminates ls from the form of antecedents, and allows the conclusion of an inductive property from finitely-many non-inductive premisses. From a different perspective, this rule expresses a form of heap abstraction in that, as far as entailment is concerned, each of all the possible models of the ls is equivalent to either the empty one or the length two one. Pushing this further, we see that the case analysis UNROLLCOLLAPSE performs when read bottom-up effects a sort of symbolic state space exploration.

Before presenting the proof, we show how this result yields decidability.

Lemma 3. *For fixed $\Pi, \Sigma, \Pi', \Sigma'$ such that no subformula of Σ is of form $\text{ls}(E_1, E_2)$, checking $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$ is decidable.*

Proof (Sketch, see also Section A.2). Because the antecedent’s spatial part is a list of points-to facts, any potential model must have a heap whose domain is exactly the size of the antecedent. Furthermore, there is an evident notion of isomorphism, where two states are isomorphic just if one is obtained from the other by L-value renaming. The fragment is closed (semantically) under isomorphism and, up to isomorphism, there are only finitely-many states of any given size. So, we check the antecedent on finitely-many canonical representatives of these equivalence classes, and when the antecedent holds we check the conclusion. \square

Corollary 4 (Validity Decidable). *For fixed $\Pi, \Sigma, \Pi', \Sigma'$, checking $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$ is decidable.*

Proof. Applying UNROLLCOLLAPSE repeatedly yields a set of entailments whose antecedents do not contain `ls`, and so can each be decided due to Lemma 3. \square

The semantic decision procedure gotten from the small model property shows that validity is in coNP; to show invalidity we can guess one of exponentially-many models of a suitably bounded size, and then satisfaction of both antecedent and consequent can easily be checked in polynomial time. We are not sure about hardness. On one side, the absence of negation from the fragment may suggest a polynomial complexity. However, a subtle form of negation is implicit in formulæ like $y \neq z \vdash \text{ls}(x, y) * \text{ls}(x, z)$, which implies that either `ls` is empty, but not both. Preliminary attempts to exploit these implicit disjunctions to reduce one of the standard coNP-complete problems to validity of entailment have failed.

3.1 Soundness of UNROLLCOLLAPSE

Note that while we are only investigating a fragment, the metatheory uses the whole of separation logic. Section A.1 summarizes the extensions to the semantics, and see [1]. The full logic is used in particular to state the following properties of the `ls` predicate, upon which soundness of UNROLLCOLLAPSE depends:

- The end of a `ls` dangles:

$$\text{ls}(-, E_2) \rightarrow (E_2 \not\leftrightarrow -) \tag{1}$$

- Each L-value reachable in a `ls`, except the end, does not dangle:

$$(E_1 \neq E_2 \wedge \text{ls}(-, E_2) \wedge - \leftrightarrow E_1) \rightarrow (E_1 \not\leftrightarrow -) \tag{2}$$

- Models of `sublss` can be changed provided cycles are not introduced:

$$\begin{aligned} & \text{ls}(E_1, E_4) \wedge (\text{ls}(E_2, E_3) * \text{true}) \\ \leftrightarrow & (\text{ls}(E_2, E_3) \wedge E_4 \not\leftrightarrow -) * ((\text{ls}(E_2, E_3) \wedge E_4 \not\leftrightarrow -) -* \text{ls}(E_1, E_4)) \end{aligned} \tag{3}$$

These can be understood simply as particular properties of `ls`, but there are more elucidating readings. That is, (1) and (2) provide a non-inductive characterization of what L-values are, and are not, in heaps modeling a `ls`. In other words, they characterize the points-to facts about models of `lss`.

Property (3), which is proved in Section A.3, states that heaps containing segments from E_1 to E_4 ($\text{ls}(E_1, E_4)$) via a segment from E_2 to E_3 ($\wedge(\text{ls}(E_2, E_3) * \text{true})$) can be split into a heap containing the subsegment ($\text{ls}(E_2, E_3)$) which, due to acyclicity, must not contain the endpoint ($\wedge E_4 \not\rightarrow -$), and $(*)$ a heap which when augmented with *any* heap containing a segment from E_2 to E_3 without E_4 ($\text{ls}(E_2, E_3) \wedge E_4 \not\rightarrow -$) yields $(-*)$ a segment from E_1 to E_4 ($\text{ls}(E_1, E_4)$). That is, while the semantics in Table 1 specifies how models of a ls are related to models of the inductive occurrence, (3) characterizes how models of a ls are related to *any* submodel which is a ls (which, summarizing the above, is simply that the submodels do not contain the endpoint). In other words, (3) characterizes the ls facts about models of ls s.

The soundness argument for UNROLLCOLLAPSE is largely concerned with analyzing the impact on validity of entailment which changing from one model of a ls to another has. For atomic formulæ, (1)–(3) give us a handle on this impact. For compound formulæ, the local reasoning supported by $*$, and precision of every predicate is essentially all we need. A predicate is *precise* [8] just when for any given stack and heap, there is at most one subheap that satisfies it; and so every predicate cuts out an unambiguous area of storage.

Thus armed, we attack the key lemma:

Lemma 5.

$$\text{If } \Pi \vdash \text{ls}^2(E_2, E_3) * \Sigma \vdash \Pi' \vdash \Sigma' \quad (4)$$

$$\text{and } s, h \models \Pi \wedge E_2 \neq E_3 \wedge E_2 \not\rightarrow - \wedge \Sigma \quad (5)$$

$$\text{then } s, h \models \Pi' \wedge (\text{ls}(E_2, E_3) \multimap \Sigma')$$

Proof. See Section A.4. □

This expresses that the ls predicate is, in some sense, “abstract”; stating, basically, that if a length two ls validates an entailment, then the entailment’s consequent is insensitive to the particular model of the ls . In more detail, any model of the entailment’s antecedent minus the ls ($\Pi \vdash \Sigma$), which is consistent with adding a nonempty ls ($E_2 \neq E_3 \wedge E_2 \not\rightarrow -$), will also model the entailment’s consequent when augmented with *any* model of the ls ($\Pi' \wedge (\text{ls}(E_2, E_3) \multimap \Sigma')$).

It may be useful to note some formulæ that, were they allowed, would cause this result to fail. First are imprecise predicates. Nearly everything breaks in their presence, but in particular, for imprecise A, B such that $s, h \models A * B$, not all subheaps of h which model A need leave or take enough heap for the remainder to model B , and so changing models of A can easily falsify B . Another problematic addition would be existentials in consequents, which would allow consequents to, e.g., impose minimum lengths with formulæ such as $\exists x, y. E_1 \mapsto x * x \mapsto y * \text{ls}(y, E_2)$, which changing models of antecedents could violate. Finally, allowing “unspecified” sharing with formulæ such as $\text{ls}(x, y) \wedge \Sigma$ gives two views of the same heap, one of which may be invalidated when replacing the heap with a different model of the other. Banning unspecified sharing forces the program annotations to explicate sharing; a restriction whose impact is presently unclear.

Once we know that consequents are insensitive to particular models of ls s, we can replace any model with one of either length 0 or 2, depending on whether

or not the pure part of the antecedent forces the endpoints to be equal, making proving soundness of UNROLLCOLLAPSE straightforward:

Proof (Proposition 2). Suppose the premisses are valid:

$$\Pi \wedge E_1 = E_2 \vdash \Sigma \vdash \Pi' \vdash \Sigma' \quad (6)$$

$$\Pi \wedge E_1 \neq E_2 \wedge x \neq E_2 \vdash E_1 \mapsto x * x \mapsto E_2 * \Sigma \vdash \Pi' \vdash \Sigma' \quad (7)$$

for $x \notin \text{fv}(\Pi, E_1, E_2, \Sigma, \Pi', \Sigma')$. Fix s, h and assume the antecedent of the conclusion: $s, h \models \Pi \vdash \text{ls}(E_1, E_2) * \Sigma$. Proceed by cases:

$\llbracket E_1 \rrbracket s = \llbracket E_2 \rrbracket s$: Hence $s, h \models \Pi \wedge E_1 = E_2 \vdash \Sigma$, and so by (6), $s, h \models \Pi' \vdash \Sigma'$.

$\llbracket E_1 \rrbracket s \neq \llbracket E_2 \rrbracket s$: Hence $h = h_{12} * h_\Sigma$ and there exists $l, s', h_{12} \models E_1 \mapsto x * \text{ls}(x, E_2)$ and $s', h_\Sigma \models \Pi \wedge E_1 \neq E_2 \vdash \Sigma$ where $s' = [s \mid x \rightarrow l]$ for x fresh. Therefore by (7), Lemma 5 ensures $s', h_\Sigma \models \Pi' \vdash (\text{ls}(E_1, E_2) * \Sigma')$, and hence $s, h \models \Pi' \vdash \Sigma'$. \square

4 Proof Theory

In the previous section we saw how UNROLLCOLLAPSE yields decidability of the fragment model-theoretically. We now see that it also forms the basis of a sound and complete proof theory, and a decision procedure based on proof-search.

The rules of the proof system are shown in Table 2. Since there is no CUT rule, the rules have a rather odd form. What we have, essentially, is a collection of axioms for the semantic properties of the assertion language, each of which has been CUT with an arbitrary formula. A noteworthy point is that the rules generally have only one premiss, so proof-search is largely simply rewriting.

Proposition 6 (Soundness). *Every derivable entailment is valid.*

Proof. The result follows from validity of each axiom's conclusion, and validity of each rule's premisses implies validity of its conclusion. The UNROLLCOLLAPSE case is Proposition 2, and the others are straightforward calculations. \square

4.1 Decidability and Completeness

While we have already shown model-theoretically that entailment between formulæ is decidable, the proof system provides another argument for decidability. We prove decidability directly by demonstrating a decision procedure.

First we define a class of formulæ which are “maximally explicit”. The primary characteristic of these formulæ, which we discuss later, is that the FRAME rule is complete for entailments with such formulæ as antecedents.

Definition 7 (Normal Form). *A formula $\Pi \vdash \Sigma$ is in normal form if*

$$\begin{aligned} \Pi \vdash \Sigma \equiv & (x_i \neq x_j)_{1 \leq i \neq j \leq n} \wedge (x_i \neq \text{nil})_{1 \leq i \leq n} \wedge (E_i \neq E'_i)_{1 \leq i \leq m} \wedge \text{true} \\ & \vdash x_1 \mapsto E''_1 * \dots * x_n \mapsto E''_n * \text{emp} \end{aligned}$$

for some n, m and where $x_i \neq x_j$ for $i \neq j$ and $E_i \neq E'_i$.

Table 2. Proof System

<p>AXIOM</p> $\frac{}{\Pi \vdash \text{emp} \vdash \text{true} \vdash \text{emp}}$	<p>INCONSISTENT</p> $\frac{}{\Pi \wedge E \neq E \vdash \Sigma \vdash \Pi' \vdash \Sigma'}$	
<p>SUBSTITUTION</p> $\frac{\Pi[E/x] \vdash \Sigma[E/x] \vdash \Pi'[E/x] \vdash \Sigma'[E/x]}{\Pi \wedge x = E \vdash \Sigma \vdash \Pi' \vdash \Sigma'}$	<p>=REFLEXIVEL</p> $\frac{\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'}{\Pi \wedge E = E \vdash \Sigma \vdash \Pi' \vdash \Sigma'}$	
<p>nilNOTLVAL</p> $\frac{\Pi \wedge E_1 \neq \text{nil} \vdash E_1 \mapsto E_2 * \Sigma \vdash \Pi' \vdash \Sigma'}{\Pi \vdash E_1 \mapsto E_2 * \Sigma \vdash \Pi' \vdash \Sigma'}$	<p>*PARTIAL</p> $\frac{\Pi \wedge E_1 \neq E_3 \vdash E_1 \mapsto E_2 * E_3 \mapsto E_4 * \Sigma \vdash \Pi' \vdash \Sigma'}{\Pi \vdash E_1 \mapsto E_2 * E_3 \mapsto E_4 * \Sigma \vdash \Pi' \vdash \Sigma'}$	
<p>UNROLLCOLLAPSE</p> $\frac{\Pi \wedge E_1 = E_2 \vdash \Sigma \vdash \Pi' \vdash \Sigma' \quad \Pi \wedge E_1 \neq E_2 \wedge x \neq E_2 \vdash E_1 \mapsto x * x \mapsto E_2 * \Sigma \vdash \Pi' \vdash \Sigma'}{\Pi \vdash \text{ls}(E_1, E_2) * \Sigma \vdash \Pi' \vdash \Sigma'} \quad x \notin \text{fv}(\Pi, E_1, E_2, \Sigma, \Pi', \Sigma')$		
<p>=REFLEXIVER</p> $\frac{\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'}{\Pi \vdash \Sigma \vdash \Pi' \wedge E = E \vdash \Sigma'}$	<p>HYPOTHESIS</p> $\frac{\Pi \wedge P \vdash \Sigma \vdash \Pi' \vdash \Sigma'}{\Pi \wedge P \vdash \Sigma \vdash \Pi' \wedge P \vdash \Sigma'}$	<p>EMPTYls</p> $\frac{\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'}{\Pi \vdash \Sigma \vdash \Pi' \vdash \text{ls}(E, E) * \Sigma'}$
<p>FRAME</p> $\frac{\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'}{\Pi \vdash S * \Sigma \vdash \Pi' \vdash S * \Sigma'}$	<p>NONEMPTYls</p> $\frac{\Pi \wedge E_1 \neq E_3 \vdash \Sigma \vdash \Pi' \vdash \text{ls}(E_2, E_3) * \Sigma'}{\Pi \wedge E_1 \neq E_3 \vdash E_1 \mapsto E_2 * \Sigma \vdash \Pi' \vdash \text{ls}(E_1, E_3) * \Sigma'}$	

We will be concerned with the following proof-search algorithm:

Algorithm 8. For goal entailment g , $\text{ps}(g)$ either fails or returns a proof of g :

$\text{ps}(g)$ = nondeterministically select a rule r such that:

g unifies with the conclusion of r , via some substitution s
and if r is nilNOTLVAL, then $E_1 \neq \text{nil} \notin \Pi$ (8)

and if r is *PARTIAL, then $E_1 \neq E_3 \notin \Pi$ (9)

and if r is FRAME or NONEMPTYls, (10)

then the antecedent of g is in normal form

if no such rule exists, then fail

else if r is an axiom, then return r

else let p_0, \dots, p_n for some n be the premisses of r after applying s
in return $r(\text{ps}(p_0), \dots, \text{ps}(p_n))$

Here we consider axioms in the proof system to be proof constants, and rules to be functions from proofs of their premisses to proofs of their conclusions.

A point to note about this algorithm is that as long as the additional sideconditions (8)–(10) are met, the order in which the rules are applied is inconsequential. Also note that when a stuck entailment is encountered the algorithm fails, without backtracking as would be standard for a proof-search algorithm.

The first step toward showing that ps is a decision procedure is termination:

Lemma 9 (Termination). *For any goal entailment, ps terminates.*

Proof. Termination of ps is established by observing that, with additional side-conditions (8) and (9), applying any rule makes progress: the size of each premiss of any rule application is lexicographically less than the size of the conclusion, where size is defined by:

Definition 10 (Size). *The size of an entailment $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$ is a triple of:*

1. *the number of lss occurring in $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$,*
2. *the number of inequalities missing from Π , that is, $|\{E_0 \neq E_1 \mid E_0, E_1 \in \text{fv}(\Pi \vdash \Sigma, \Pi' \vdash \Sigma') \cup \{\text{nil}\}\} \setminus \Pi|$,*
3. *the length of $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$, where length is defined in the obvious way taking all simple formulæ to have length 1.*

□

When ps fails, the short story is that it has found a disproof of the goal. We begin explaining this by analyzing entailments with antecedents in normal form.

Observation 11. *The antecedent of every entailment to which no rule applies, except possibly FRAME and NONEMPTYs, is in normal form.*

For a more intuitive characterization of normal form, note that formulæ $\Pi \vdash \Sigma$ in normal form satisfy the following properties:

1. No equalities $E=E'$ (other than reflexive $E=E$) are guaranteed to hold.
2. The only inequalities $E \neq E'$ guaranteed to hold appear explicitly in Π .
3. The only expressions E guaranteed to be in the domain of the heap appear explicitly as $E \mapsto E'$ in Σ .

A key property of normal forms is satisfiability. Later we will make use of two different types of model of such formulæ:

Definition 12 (Bad Model). *For $\Pi \vdash \Sigma$ in normal form:*

1. *A bad model of $\Pi \vdash \Sigma$ is a state $s, h \models \Pi \vdash \Sigma$ where $\text{nil} \notin \text{rng}(s)$ and s is one-one on $\text{fv}(\Pi \vdash \Sigma)$, and h is uniquely determined by s (as in Lemma 24).*
2. *A bad model of $\Pi \vdash \Sigma$ with $x=E$ is a state $s, h \models \Pi \wedge x=E \vdash \Sigma$ where, for s', h' a bad model of $\Pi \vdash \Sigma$, $s = [s' \mid x \mapsto \llbracket E \rrbracket s']$, and h is uniquely determined by s .*

Lemma 13. *For any formula $\Pi \vdash \Sigma$ in normal form:*

1. *There exists a bad model of $\Pi \vdash \Sigma$.*
2. *For any $x \neq E \notin \Pi$, there exists a bad model of $\Pi \vdash \Sigma$ with $x=E$.*

Now for the crux of correctness of ps in the failure case, and completeness of the proof system: when ps reaches a stuck entailment, it is invalid, and invalidity is preserved throughout the path of rule applications ps made from the goal to the stuck entailment.⁵

⁵ Furthermore, countermodels of stuck entailments could be computed, and countermodels of a rule's conclusion could be computed from a countermodel of one the rule's premisses. So ps could be defined so as to either return a proof or a countermodel of the goal.

Lemma 14 (Stuck Invalidity). *Every entailment stuck for ps is invalid.*

Proof (Sketch). Consider a stuck entailment $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$, whose antecedent, by Observation 11, is in normal form. Proceed by cases:

- [$\Sigma' \equiv \text{emp}$ and $\Pi' \equiv \Pi'' \wedge E=E'$]: Note $E \neq E'$ since $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$ is stuck. Therefore a bad model of $\Pi \vdash \Sigma$ is a countermodel.
- [$\Sigma' \equiv \text{emp}$ and $\Pi' \equiv \Pi'' \wedge E \neq E'$]: Note $E \neq E' \notin \Pi$ since $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$ is stuck. Therefore a bad model of $\Pi \vdash \Sigma$ with $E=E'$ is a countermodel.
- [$\Sigma' \equiv E \mapsto E' * \Sigma''$]: Therefore since $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$ is stuck, $E \mapsto E' \notin \Sigma$. Hence, s, h a bad model of $\Pi \vdash \Sigma$ is a countermodel, since either $\llbracket E \rrbracket s \notin \text{dom}(h)$ or $h(\llbracket E \rrbracket s) \neq \llbracket E' \rrbracket s$.
- [$\Sigma' \equiv \text{ls}(\text{nil}, E) * \Sigma''$]: Therefore s, h a bad model of $\Pi \vdash \Sigma$ is a countermodel, since $\text{nil} \neq \llbracket E \rrbracket s$.
- [$\Sigma' \equiv \text{ls}(x, E) * \Sigma''$ and for all $E'. x \mapsto E' \notin \Sigma$]: Therefore s, h a bad model of $\Pi \vdash \Sigma$ is a countermodel, since $\llbracket x \rrbracket s \neq \llbracket E \rrbracket s$ and $\llbracket x \rrbracket s \notin \text{dom}(h)$.
- [$\Sigma \equiv x \mapsto E * \Sigma_0$ and $\Sigma' \equiv \text{ls}(x, E') * \Sigma_1$]: Note that Σ_1 contains only lss, since the other cases have already been covered. Let s, h be a bad model of $\Pi \vdash \Sigma$ with $x=E'$ ($x \neq E' \notin \Pi$ since $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$ is stuck). Therefore $s, h \models \Pi \vdash x \mapsto x * \Sigma_0$ and $s, h \not\models \Pi' \vdash \text{ls}(x, E') * \Sigma_1$, since no ls contains a nonempty cycle, see Lemma 28. Therefore s, h is a countermodel. \square

Lemma 15 (Invalidity Preservation). *For all rule applications satisfying sidecondition (10) of Algorithm 8, invalidity of any of the rule's premisses implies invalidity of the rule's conclusion.*

Proof. See Section A.5. \square

Proposition 16 (Decidability). *Validity of entailment is decidable, in particular, ps is a decision procedure.*

Proof. Lemma 9 establishes termination. For correctness, in case ps returns normally with a proof, correctness is immediate from Proposition 6. Otherwise ps has failed after reaching a stuck entailment. We argue that this implies invalidity of the goal entailment, and hence correctness, by noting that each stuck entailment is itself invalid, due to Lemma 14, and that each rule application in the path from the goal preserves invalidity, due to Lemma 15. Transitively, this shows that when a stuck entailment is encountered, it is invalid, and all the entailments down to the goal are also invalid. \square

Unsurprisingly, completeness is immediate from decidability:

Corollary 17 (Completeness). *Every underivable entailment is invalid.*

But note that arguing completeness of a proof theory in the contrapositive is somewhat odd. That is, rather than an argument based on showing existence of proofs of all valid entailments, we have given an argument based on disproof. A more conventional proof of completeness may be possible, but the sparseness of the fragment could be an obstacle to the usual techniques.

5 Conclusions

In this paper we have proven a decidability result for a logic for just one kind of pointer data structure: linked lists. And it was not easy work. There have been other results as well in this territory (e.g., [6, 7, ?]) but, frankly, we are not sure if it is possible to obtain a canonical decidable fragment that covers a large variety of structures.

Although the main focus in this paper was decidability, the fragment appears to be of some interest in itself. Crucially, its proof theory is extremely deterministic. In particular, there is no need to attempt many different splittings of a context as is usually the case in proof-search for substructural logics. This is a reflection of a semantic property enjoyed by the fragment: every assertion is precise. This then implies that there can be at most one heap splitting used to satisfy a $*$ formula. The absence of (general) disjunction in the fragment is crucial for precision. It is, however, possible to incorporate restricted, disjoint, forms of disjunction, corresponding to if-then-else, without sacrificing precision. These forms are useful in playing the role of guards for inductive definitions, and one of them is implicitly present in the ls predicate.

In future work we plan to add a mechanism for inductive definitions to the fragment. At present we can see how some definitions (e.g., trees) preserve decidability, but we are not sure how far we can go in this direction. Even if decidability cannot be maintained, the computational nature of the proof theory of precise predicates should give a way to selectively consider how deep to go in inductions in a way that gives strong control over proof-search.

References

1. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: *Logic in Computer Science (LICS'02)*, IEEE (2002) 55–74
2. Reynolds, J.C.: Intuitionistic reasoning about shared mutable data structure. In Davies, J., Roscoe, B., Woodcock, J., eds.: *Millennial Perspectives in Computer Science*, Houndsmill, Hampshire, Palgrave (2000) 303–321
3. Isthiaq, S., O’Hearn, P.: BI as an assertion language for mutable data structures. In: *Conference Record of the Twenty-Eighth Annual ACM Symposium on Principles of Programming Languages*, London (2001) 39–46
4. O’Hearn, P., Reynolds, J., Yang, H.: Local reasoning about programs that alter data structures. In: *Computer Science Logic (CSL'01)*. Volume 2142 of *Lecture Notes in Computer Science*, Springer (2001) 1–19
5. Calcagno, C., Yang, H., O’Hearn, P.: Computability and complexity results for a spatial assertion language for data structures. In: *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01)*. Volume 2245 of *Lecture Notes in Computer Science*, Springer (2001) 108–119
6. Jenson, J., Jorgensen, M., Klarkund, N., Schwartzback, M.: Automatic verification of pointer programs using monadic second-order logic. In: *Proceedings of the ACM SIGPLAN'97 Conference on Programming Language Design and Implementation*. (1997) 225–236 *SIGPLAN Notices* 32(5).

7. Benedikt, M., Reps, T., Sagiv, M.: A decidable logic for describing linked data structures. In: Proceedings of the 8th European Symposium on Programming. Volume 1576 of Lecture Notes in Computer Science., Springer (1999) 2–19
8. O’Hearn, P.W., Yang, H., Reynolds, J.C.: Separation and information hiding. In: 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Venice (2004) 268–280

A Appendix: Selected Proofs

A.1 Full Separation Logic

The full semantics of the separation logic connectives used in the metatheory extends Table 1 as follows:

$$\begin{aligned}
s, h \models A \rightarrow B &\stackrel{\text{def}}{\text{iff}} s, h \models A \text{ implies } s, h \models B \\
s, h \models A \multimap B &\stackrel{\text{def}}{\text{iff}} \text{for all } h' \perp h. s, h' \models A \text{ implies } s, h * h' \models B \\
s, h \models \exists x. A &\stackrel{\text{def}}{\text{iff}} \text{there exists } v \in \text{Values}. [s \mid x \rightarrow v], h \models A
\end{aligned}$$

Also, note that $E \leftrightarrow E'$ abbreviates $E \mapsto E' * \text{true}$, $E \not\leftrightarrow E'$ abbreviates $\neg(E \leftrightarrow E')$. We sometimes write formulæ with a blank $A[-]$ as a shorthand for $\exists x. A[x]$ for x fresh. In these cases, the scope of the new variable is taken to be as narrow as possible: for example $E \not\leftrightarrow -$ abbreviates $\neg(\exists x. E \leftrightarrow x)$.

A.2 Proof of Lemma 3 from Section 3

The proof proceeds by an argument similar to that in [5]:

First we define an equivalence which equates states up to renaming L-values, and arbitrary differences on variables other than those in a given set:

Definition 18. $s, h \approx_X s', h'$ if and only if there exists a bijection $r \in \text{Values} \rightarrow \text{Values}$ such that $r(\text{nil}) = \text{nil}$, $r(s(x)) = s'(x)$ for all $x \in X$, and either $l \notin \text{dom}(h)$ and $r(l) \notin \text{dom}(h')$ or $r(h(l)) = h'(r(l))$ for all $l \in \text{L-values}$.

The main property of this relation is that formulæ are insensitive to differences between equivalent states:

Lemma 19. If $s, h \approx_{fv(A)} s', h'$, then $s, h \models A$ if and only if $s', h' \models A$.

We now identify a finite set of *canonical* stacks:

Definition 20. The set $\mathbf{S}_{\Pi \upharpoonright \Sigma}$ of canonical stacks with respect to $\Pi \upharpoonright \Sigma$ is defined as follows: $s \in \mathbf{S}_{\Pi \upharpoonright \Sigma}$ iff $s(\text{Variables} \setminus fv(\Pi \upharpoonright \Sigma)) \subseteq \{\text{nil}\}$, and $s(fv(\Pi \upharpoonright \Sigma)) \subseteq L \cup \{\text{nil}\}$ where L consists of the first $|fv(\Pi \upharpoonright \Sigma)|$ -many L-values.

Observation 21. The set $\mathbf{S}_{\Pi \upharpoonright \Sigma}$ is finite for any $\Pi \upharpoonright \Sigma$.

There are enough canonical stacks that every state is equivalent to one whose stack is canonical:

Lemma 22. For $\Pi \upharpoonright \Sigma$ and all $s, h \in \text{States}$, there exist $s' \in \mathbf{S}_{\Pi \upharpoonright \Sigma}, h' \in \text{Heaps}$ such that $s, h \approx_{fv(\Pi \upharpoonright \Sigma)} s', h'$.

It is then immediate that considering only finitely-many stacks suffices:

Corollary 23. For $\Pi \upharpoonright \Sigma, \Pi' \upharpoonright \Sigma'$

$$\begin{aligned}
&\text{for all } s, h \in \text{States}. \quad s, h \models \Pi \upharpoonright \Sigma \text{ implies } s, h \models \Pi' \upharpoonright \Sigma' \\
&\text{iff for all } s \in \mathbf{S}_{\Pi \upharpoonright \Sigma}, h \in \text{Heaps}. s, h \models \Pi \upharpoonright \Sigma \text{ implies } s, h \models \Pi' \upharpoonright \Sigma'
\end{aligned}$$

Now since the antecedent does not contain ls , it suffices to consider a single heap for each stack:

Lemma 24. *For $\Pi \vdash \Sigma, \Pi' \vdash \Sigma'$ such that no subformula of Σ is of form $\text{ls}(E_1, E_2)$, and $s \in \text{Stacks}$, checking for all $h \in \text{Heaps}$. $s, h \models \Pi \vdash \Sigma$ implies $s, h \models \Pi' \vdash \Sigma'$ is decidable.*

Proof. Since ls does not appear in Σ , $\Sigma \equiv E_1 \mapsto E'_1 * \dots * E_n \mapsto E'_n * \text{emp}$ for some n . Therefore, since s is fixed, this determines a unique heap h which may model $\Pi \vdash \Sigma$; for all others the desired implication holds vacuously. The result then follows from Lemma 1. \square

Lemma 3. *For fixed $\Pi, \Sigma, \Pi', \Sigma'$ such that no subformula of Σ is of form $\text{ls}(E_1, E_2)$, checking $\Pi \vdash \Sigma \vdash \Pi' \vdash \Sigma'$ is decidable.*

Proof. Compose Corollary 23 and Lemma 24. \square

A.3 Proof of (3) from Section 3.1

The proof makes use of the following additional lemmas:

Lemma 25. *Adjacent ls s whose end dangles can be appended:*

$$((\text{ls}(E_1, E_2) \wedge E_3 \not\mapsto -) * \text{ls}(E_2, E_3)) \rightarrow \text{ls}(E_1, E_3)$$

Lemma 26. *The end point of a nonempty ls is pointed to:*

$$(E_1 \neq E_2 \wedge \text{ls}(E_1, E_2)) \rightarrow - \hookrightarrow E_2$$

Lemma 27. *A ls with an internal link can be split into two separate ls s at the link:*

$$(\text{ls}(E_1, E_3) \wedge E_2 \hookrightarrow -) \rightarrow (\text{ls}(E_1, E_2) * \text{ls}(E_2, E_3))$$

Lemma 28. *A ls cannot contain (nonempty) cycles:*

$$\text{ls}(E_1, E_4) \rightarrow \left(E_2 \not\mapsto E_2 \wedge \neg(E_2 \neq E_3 \wedge (\text{ls}(E_2, E_3) * \text{true}) \wedge (\text{ls}(E_3, E_2) * \text{true})) \right)$$

Lemma 29. *A ls with a nonempty subls can be split into three separate ls s:*

$$\begin{aligned} & \text{ls}(E_1, E_4) \wedge (E_2 \neq E_3 \wedge \text{ls}(E_2, E_3) * \text{true}) \\ & \rightarrow \text{ls}(E_1, E_2) * \text{ls}(E_2, E_3) * \text{ls}(E_3, E_4) \end{aligned}$$

Proof. Fix s, h and assume the antecedent: $s, h \models \text{ls}(E_1, E_4) \wedge (E_2 \neq E_3 \wedge \text{ls}(E_2, E_3) * \text{true})$. Therefore

$$s, h \models \text{ls}(E_1, E_4) \tag{11}$$

$$s, h_{23} \models E_2 \neq E_3 \wedge \text{ls}(E_2, E_3) \tag{12}$$

for $h_{23} \subseteq h$. Therefore $s, h_{2-} \models E_2 \mapsto -$ for $h_{2-} \subseteq h_{23}$, and hence $s, h \models E_2 \hookrightarrow -$, and hence $s, h \models \text{ls}(E_1, E_4) \wedge E_2 \hookrightarrow -$. Thus Lemma 27 applies, and we have

$$s, h \models \text{ls}(E_1, E_2) * \text{ls}(E_2, E_4) \tag{13}$$

Proceed by cases:

$\llbracket E_3 \rrbracket s = \llbracket E_4 \rrbracket s$: Therefore $s, \emptyset \models \text{ls}(E_3, E_4)$. Therefore, by (13) and the case equality, $s, h \models \text{ls}(E_1, E_2) * \text{ls}(E_2, E_3)$, and hence $s, h \models \text{ls}(E_1, E_2) * \text{ls}(E_2, E_3) * \text{ls}(E_3, E_4)$.

$\llbracket E_3 \rrbracket s \neq \llbracket E_4 \rrbracket s$: By (12) and Lemma 26, $s, h_{23} \models -\hookrightarrow E_3$, and hence $s, h \models -\hookrightarrow E_3$. Therefore by the case inequality, (11), and (2), $s, h \models E_3 \hookrightarrow -$.

Now by (13), $h = h_{12} * h_{24}$ such that

$$s, h_{12} \models \text{ls}(E_1, E_2) \quad (14)$$

and

$$s, h_{24} \models \text{ls}(E_2, E_4) \quad (15)$$

So proceed by cases:

$[s, h_{12} \models E_3 \hookrightarrow -]$: Therefore by (14) and Lemma 27, $s, h_{12} \models \text{ls}(E_1, E_3) * \text{ls}(E_3, E_2)$. Therefore by (12), $s, h \models (\text{ls}(E_2, E_3) * \text{true}) \wedge (\text{ls}(E_3, E_2) * \text{true})$, which with (11), contradicts Lemma 28.

$[s, h_{24} \models E_3 \hookrightarrow -]$: Therefore, by (15), $s, h_{24} \models \text{ls}(E_2, E_4) \wedge E_3 \hookrightarrow -$, and hence by Lemma 27, $s, h_{24} \models \text{ls}(E_2, E_3) * \text{ls}(E_3, E_4)$. Therefore by (14), $s, h \models \text{ls}(E_1, E_2) * \text{ls}(E_2, E_3) * \text{ls}(E_3, E_4)$. \square

Lemma 30 ((3)). *Models of subls can be changed provided cycles are not introduced:*

$$\begin{aligned} & \text{ls}(E_1, E_4) \wedge (\text{ls}(E_2, E_3) * \text{true}) \\ \leftrightarrow & (\text{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -) * ((\text{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -) \multimap \text{ls}(E_1, E_4)) \end{aligned} \quad (3)$$

Proof. $[\rightarrow]$: Fix s, h , assume the antecedent: $s, h \models \text{ls}(E_1, E_4) \wedge (\text{ls}(E_2, E_3) * \text{true})$, and proceed by cases:

$\llbracket E_2 \rrbracket s = \llbracket E_3 \rrbracket s$: Immediate.

$\llbracket E_2 \rrbracket s \neq \llbracket E_3 \rrbracket s$: Therefore $h = h_{23} * h'$ such that $s, h_{23} \models \text{ls}(E_2, E_3)$. Also, by (1), $s, h \models E_4 \not\hookrightarrow -$, and hence, $s, h_{23} \models E_4 \not\hookrightarrow -$. Furthermore, with the case assumption, by Lemma 29, $s, h \models \text{ls}(E_1, E_2) * \text{ls}(E_2, E_3) * \text{ls}(E_3, E_4)$. Therefore, by precision of $\text{ls}(E_2, E_3)$, $s, h' \models \text{ls}(E_1, E_2) * \text{ls}(E_3, E_4)$. Hence $s, h_{12} \models \text{ls}(E_1, E_2)$ for $h_{12} \subseteq h'$, and so $h_{12} \models E_4 \not\hookrightarrow -$. Now fix $h_{\multimap} \perp h'$ such that $s, h_{\multimap} \models \text{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -$. Therefore $s, h_{\multimap} * h' \models (\text{ls}(E_1, E_2) \wedge E_4 \not\hookrightarrow -) * (\text{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -) * \text{ls}(E_3, E_4)$. Hence, by Lemma 25 twice, $s, h_{\multimap} * h' \models \text{ls}(E_1, E_4)$. Therefore $s, h' \models (\text{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -) \multimap \text{ls}(E_1, E_4)$, and hence, $s, h \models (\text{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -) * ((\text{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -) \multimap \text{ls}(E_1, E_4))$.

$[\leftarrow]$: Immediate. \square

A.4 Proof of Lemma 5 from Section 3.1

The proof makes use of the following consequence of the fact that all formulæ are precise:

Lemma 31. *For any singleton subheap of a model of a spatial formula, there is a simple spatial formula modeled by an extension of the subheap:*

*If $s, h_0 * h_1 \models \Sigma$ for h_0 a singleton heap, then there exists S such that:*

1. $\Sigma \equiv S * \Sigma'$ for some Σ' , and
2. $s, h_0 * h'_1 \models S$ for some $h'_1 \subseteq h_1$.

Lemma 5.

$$\text{If } \Pi \vdash \text{ls}^2(E_2, E_3) * \Sigma \vdash \Pi' \vdash \Sigma' \quad (4)$$

$$\text{and } s, h \models \Pi \wedge E_2 \neq E_3 \wedge E_2 \not\leftrightarrow - \wedge \Sigma \quad (5)$$

$$\text{then } s, h \models \Pi' \wedge (\text{ls}(E_2, E_3) \multimap \Sigma')$$

Proof. Choose x fresh and

$$l \in \text{L-values} \setminus s(\text{fv}(\Pi, E_2, E_3, \Sigma, \Pi', \Sigma')) \setminus \text{dom}(h) \quad (16)$$

and define $s' = [s \mid x \rightarrow l]$. Now define

$$h_2 = [[E_2]]s' \rightarrow l \quad (17)$$

$$\text{and } h_l = [l \rightarrow [E_3]]s' \quad (18)$$

and so

$$s, h_2 * h_l \models \text{ls}^2(E_2, E_3) \quad (19)$$

Also, by (5) and (16), $h_2 * h_l \perp h$.

Therefore by (5), $s', h_2 * h_l * h \models \Pi \vdash \text{ls}^2(E_2, E_3) * \Sigma$. Therefore (4) applies and we have

$$s', h_2 * h_l * h \models \Pi' \vdash \Sigma' \quad (20)$$

Therefore Lemma 31 applies with $[s'/s \mid h_2/h_0 \mid h_l * h/h_1 \mid \Sigma'/\Sigma]$, so let S be the simple spatial formula whose existence it ensures. Therefore

$$\Sigma' \equiv S * \Sigma''$$

$$\text{and } h_l * h = h_S * h_{\Sigma''} \quad (21)$$

$$\text{such that } s', h_2 * h_S \models S \quad (22)$$

Therefore by (20) and precision of S

$$s', h_{\Sigma''} \models \Pi' \vdash \Sigma'' \quad (23)$$

Proceed by cases on the syntax of S :

$[S \equiv E_1 \mapsto E_4]$: Impossible: Therefore by (22) and (17), $[[E_1]]s' = [[E_2]]s'$, since $h_2 * h_S$ must be a singleton. Hence, $[[E_4]]s' = l$. But, by (16), $[[E_4]]s' = [[E_4]]s \neq l$, contradiction.

$[S \equiv \text{ls}(E_1, E_4)]$: Note that, by (17), $s', h_2 \vDash -\hookrightarrow x$. Then, by (16), $\llbracket E_4 \rrbracket s' = \llbracket E_4 \rrbracket s \neq l$, and so $s', h_2 \vDash x \neq E_4 \wedge -\hookrightarrow x$. Therefore by (22), $s', h_2 * h_S \vDash x \neq E_4 \wedge \text{ls}(-, E_4) \wedge -\hookrightarrow x$, and by (2), $s', h_2 * h_S \vDash x \hookrightarrow -$. Since by (16), $\llbracket E_2 \rrbracket s' = \llbracket E_2 \rrbracket s \neq l$, and by (18), $h_S = h_l * h'$ for some h' . Therefore by (21)

$$h = h' * h_{\Sigma''} \quad (24)$$

and by (22)

$$s, h_2 * h_l * h' \vDash \text{ls}(E_1, E_4) \quad (25)$$

Therefore by (19) and (3), $s, h_2 * h_l * h' \vDash (\text{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -) * ((\text{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -) * \text{ls}(E_1, E_4))$. Therefore by (19) and precision of $\text{ls}(E_2, E_3)$

$$s, h' \vDash (\text{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -) * \text{ls}(E_1, E_4) \quad (26)$$

In order to prove $s, h \vDash \text{ls}(E_2, E_3) * \Sigma'$, fix:

$[h_{-*} \perp h$ such that $s, h_{-*} \vDash \text{ls}(E_2, E_3)]$: Proceed by cases:

$[s, h_{-*} \vDash E_4 \hookrightarrow -]$: Proceed by cases:

$\llbracket E_3 \rrbracket s = \llbracket E_4 \rrbracket s$: Impossible: Therefore $s, h_{-*} \vDash \text{ls}(E_2, E_4)$. Therefore by (1), $s, h_{-*} \vDash E_4 \not\hookrightarrow -$, which contradicts the once removed case assumption.

$\llbracket E_3 \rrbracket s \neq \llbracket E_4 \rrbracket s$: Proceed by cases:

$\llbracket E_2 \rrbracket s = \llbracket E_4 \rrbracket s$: Impossible: Therefore by (25), $s, h_2 * h_l * h' \vDash \text{ls}(E_1, E_2)$. Hence by (1), $s, h_2 * h_l * h' \vDash E_2 \not\hookrightarrow -$, which contradicts (17).

$\llbracket E_2 \rrbracket s \neq \llbracket E_4 \rrbracket s$: Impossible: Define

$$h_{243} = \llbracket E_2 \rrbracket s \rightarrow \llbracket E_4 \rrbracket s \mid \llbracket E_4 \rrbracket s \rightarrow \llbracket E_3 \rrbracket s \quad (27)$$

Note h_{243} is well-defined by the case assumption, and $h_{243} \perp h$ since, by (5) and the thrice removed case assumption, $\llbracket E_2 \rrbracket s \notin \text{dom}(h)$, and also by the twice and thrice removed case assumptions, $\llbracket E_4 \rrbracket s \notin \text{dom}(h)$. Therefore by (5) and the once removed case assumption, $s, h_{243} \vDash \text{ls}^2(E_2, E_3)$, and hence by (5), $s, h_{243} * h \vDash \Pi \wedge \text{ls}^2(E_2, E_3) * \Sigma$, and by (4), $s, h_{243} * h \vDash \Sigma'$, and hence $s, h_{243} * h \vDash S * \Sigma''$. Therefore by (24), (23), and precision of Σ'' , $s, h_{243} * h' \vDash S$, that is $s, h_{243} * h' \vDash \text{ls}(E_1, E_4)$. Therefore by (1), $s, h_{243} * h' \vDash E_4 \not\hookrightarrow -$, which contradicts (27).

$[s, h_{-*} \vDash E_4 \not\hookrightarrow -]$: Therefore $s, h_{-*} \vDash \text{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -$. Therefore by (26), $s, h_{-*} * h' \vDash \text{ls}(E_1, E_4)$, and hence by (23), $s, h_{-*} * h' * h_{\Sigma''} \vDash \text{ls}(E_1, E_4) * \Sigma''$, and by (24), $s, h_{-*} * h \vDash \text{ls}(E_1, E_4) * \Sigma''$.

Therefore $s, h \vDash \text{ls}(E_2, E_3) * \Sigma'$, and hence by (23), $s, h \vDash \Pi' \wedge (\text{ls}(E_2, E_3) * \Sigma')$. \square

A.5 Proof of Lemma 15 from Section 4.1

Lemma 15 (Invalidity Preservation). *For all rule applications satisfying sidecondition (10) of Algorithm 8, invalidity of any of the rule’s premisses implies invalidity of the rule’s conclusion.*

Proof. We prove the contrapositive:

[UNROLLCOLLAPSE]: Any model of either premiss’s antecedent also models the conclusion’s antecedent, and hence, by validity of the conclusion, models the conclusion’s consequent, which is also the premiss’s consequent.

[FRAME]: Suppose $\Pi \upharpoonright S * \Sigma$ is in normal form, by (10), and $\Pi \upharpoonright S * \Sigma \vdash \Pi' \upharpoonright S * \Sigma'$ is valid, and fix s, h such that $s, h \models \Pi \upharpoonright \Sigma$. Therefore $S \equiv E_0 \mapsto E'_0$, $\Sigma \equiv E_1 \mapsto E'_1 * \dots * E_n \mapsto E'_n * \text{emp}$ for some n , and $E_0 \neq E_i \in \Pi$ for all $1 \leq i \leq n$. Hence $\llbracket E_0 \rrbracket s \notin \{\llbracket E_i \rrbracket s \mid 1 \leq i \leq n\} = \text{dom}(h)$. Therefore $h' = [h \mid \llbracket E_0 \rrbracket s \rightarrow \llbracket E'_0 \rrbracket s]$ is well-defined, and $s, h' \models \Pi \upharpoonright S * \Sigma$. Hence by validity of the conclusion, $s, h' \models \Pi' \upharpoonright S * \Sigma'$, and by precision of S , $s, h \models \Pi' \upharpoonright \Sigma'$.

[NONEMPTYls]: Similar, since NONEMPTYls simply unrolls the ls once (which is an identity in the model) and then applies FRAME.

We omit the other cases, which are straightforward calculations in the model. \square