A Decidable Fragment of Separation Logic

Josh Berdine[1], Cristiano Calcagno[2], and Peter W. O'Hearn[1]

[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 $\Pi \mid \Sigma$, 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 $-\!\!*$) 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 $-\!\!*$ 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 decidability 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:

$$\begin{array}{rll}
x, y, \ldots & \in \text{Variables} & \text{variables} \\
E & ::= \mathsf{nil} \mid x & \text{Expressions} \\
P & ::= E{=}E \mid \neg P & \text{simple Pure formulæ} \\
\Pi & ::= \mathsf{true} \mid \Pi \wedge P & \text{Pure formulæ} \\
S & ::= E{\mapsto}E \mid \mathsf{ls}(E, E) & \text{simple Spatial formulæ} \\
\Sigma & ::= \mathsf{emp} \mid S * \Sigma & \text{Spatial formulæ} \\
A & ::= P \mid \Pi \mid S \mid \Sigma \mid \Pi ¦ \Sigma & \text{formulæ}
\end{array}$$

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]

$$\begin{array}{rlrl}
 & s\,, h \vDash A & & [\![E]\!] \in \text{Stacks} \to \text{Values} \\
\text{Stacks} & \overset{\text{def}}{=} \text{Variables} \to \text{Values} & \text{R-values} & \overset{\text{def}}{=} \text{Values} \\
\text{Heaps} & \overset{\text{def}}{=} \text{L-values} \overset{\text{fin}}{\rightharpoonup} \text{R-values} & \text{L-values} & \overset{\text{def}}{\subset} \text{Values} \\
\text{States} & \overset{\text{def}}{=} \text{Stacks} \times \text{Heaps} & \mathit{nil} & \overset{\text{def}}{\in} \text{Values} \smallsetminus \text{L-values}
\end{array}$$

The semantics of the assertion language is shown in Table 1, where $\mathit{fv}(E)$ simply denotes the variables occurring in E. 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 $x{=}y ¦ x{\mapsto}42$ we also assert that the

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

Table 1. Semantics of Assertion Language

$$[\![x]\!]s \overset{\text{def}}{=} s(x) \qquad\qquad [\![\mathsf{nil}]\!]s \overset{\text{def}}{=} \mathit{nil}$$

$$\begin{array}{ll}
s\,,h \vDash E_1{=}E_2 & \overset{\text{def}}{\text{iff}}\ [\![E_1]\!]s = [\![E_2]\!]s \\
s\,,h \vDash \neg P & \overset{\text{def}}{\text{iff}}\ s\,,h \nvDash P \\
 & \\
s\,,h \vDash \mathsf{true} & \phantom{\text{iff}}\ \text{always} \\
s\,,h \vDash \Pi \wedge P & \overset{\text{def}}{\text{iff}}\ s\,,h \vDash \Pi \text{ and } s\,,h \vDash P \\
 & \\
s\,,h \vDash E_1{\mapsto}E_2 & \overset{\text{def}}{\text{iff}}\ h = [\emptyset \mid [\![E_1]\!]s{\to}[\![E_2]\!]s] \\
s\,,h \vDash \mathsf{ls}(E_1, E_2) & \overset{\text{def}}{\text{iff}}\ \text{there exists } n.\ s\,,h \vDash \mathsf{ls}^n(E_1, E_2) \\
s\,,h \vDash \mathsf{ls}^0(E_1, E_2) & \overset{\text{def}}{\text{iff}}\ [\![E_1]\!]s = [\![E_2]\!]s \text{ and } h = \emptyset \\
s\,,h \vDash \mathsf{ls}^{n+1}(E_1, E_2) & \overset{\text{def}}{\text{iff}}\ [\![E_1]\!]s \neq [\![E_2]\!]s \text{ and} \\
 & \phantom{\text{iff}}\ \text{there exists } v \in \text{Values}.\ [s \mid x{\to}v]\,,h \vDash E_1{\mapsto}x * \mathsf{ls}^n(x, E_2) \\
 & \phantom{\text{iff}}\ \text{for } x \notin \mathit{fv}(E_1, E_2) \\
 & \\
s\,,h \vDash \mathsf{emp} & \overset{\text{def}}{\text{iff}}\ h = \emptyset \\
s\,,h \vDash S * \Sigma & \overset{\text{def}}{\text{iff}}\ \text{there exists } h_1 \perp h_2.\ h = h_1{*}h_2 \text{ and } s\,,h_1 \vDash S \text{ and } s\,,h_2 \vDash \Sigma \\
 & \\
s\,,h \vDash \Pi \mathrel{¦} \Sigma & \overset{\text{def}}{\text{iff}}\ s\,,h \vDash \Pi \text{ and } s\,,h \vDash \Sigma
\end{array}$$

stack maps x and y to equal R-values. Since the conjuncts of a $*$ formula must be true in disjoint heaps, $x{=}y \mathrel{¦} x{\mapsto}\mathsf{nil} * y{\mapsto}\mathsf{nil}$ is unsatisfiable.

The ls predicate describes segments of linked list structures in the heap: $\mathsf{ls}(x, y)$ 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 $\mathsf{ls}(x, x)$ describes the empty list segment, and is equivalent to emp. Were the endpoint not required to be dangling, then $\mathsf{ls}(x, x)$ could describe cyclic lists containing x. Instead, a cyclic list is described for instance with $x{\mapsto}y * \mathsf{ls}(y, x)$. For some further examples, $\mathsf{ls}(x, \mathsf{nil})$ describes "complete" lists, rather than segments. A list with an intermediate link can be expressed with $\mathsf{ls}(x, y){*}\mathsf{ls}(y, \mathsf{nil})$, two non-overlapping lists with $\mathsf{ls}(x, \mathsf{nil}){*}\mathsf{ls}(y, \mathsf{nil})$, and two lists with a shared tail with $\mathsf{ls}(x, z) * \mathsf{ls}(y, z) * \mathsf{ls}(z, \mathsf{nil})$.

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

[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.

form $\Pi \mid \Sigma \vdash \Pi' \mid \Sigma'$, we wish to check if for all $s, h.\, s\,, h \vDash \Pi \mid \Sigma$ implies $s\,, h \vDash \Pi' \mid \Sigma'$. Before getting stuck into decidability, we try to develop some intuition with a few examples.

First trivially, anything entails itself, up to equalities: $x{=}y \wedge E{=}F \mid x{\mapsto}E \vdash y{\mapsto}F$. As $nil \notin$ L-values, $x{\mapsto}E \vdash x{\neq}\mathsf{nil} \mid x{\mapsto}E$. Also, since $*$ guarantees separation, spatial formulæ have implicit non-alias consequences: $x{\mapsto}E * y{\mapsto}F \vdash x{\neq}y \mid x{\mapsto}E * y{\mapsto}F$. Explicit descriptions of list segments entail the inductive descriptions: $x{=}y \mid \mathsf{emp} \vdash \mathsf{ls}(x, y)$ for length 0, $x{\neq}y \mid x{\mapsto}y \vdash \mathsf{ls}(x, y)$ for length 1, $x{\neq}y \wedge z{\neq}y \mid x{\mapsto}z * z{\mapsto}y \vdash \mathsf{ls}(x, y)$ for length 2, and $x{\neq}y \mid x{\mapsto}z * \mathsf{ls}(z, y) \vdash \mathsf{ls}(x, y)$ 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: $\mathsf{ls}(x, z) * \mathsf{ls}(z, \mathsf{nil}) \vdash \mathsf{ls}(x, \mathsf{nil})$.

Before attacking entailment validity, we must consider formula satisfaction:

Lemma 1 (Satisfaction Decidable). *For given $s\,, h, \Pi \mid \Sigma$, checking the satisfaction $s\,, h \vDash \Pi \mid \Sigma$ 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 remove the evident singleton from the heap (if present) and continue; if the sigleton is not present we report "no". 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 $\Pi \mid \Sigma \vdash \Pi' \mid \Sigma'$ 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:*

$$\textsc{UnrollCollapse}\qquad \frac{\begin{array}{c}\Pi \wedge E_1{=}E_2 \mid \Sigma \vdash \Pi' \mid \Sigma' \\ \Pi \wedge E_1{\neq}E_2 \wedge x{\neq}E_2 \mid E_1{\mapsto}x * x{\mapsto}E_2 * \Sigma \vdash \Pi' \mid \Sigma'\end{array}}{\Pi \mid \mathsf{ls}(E_1, E_2) * \Sigma \vdash \Pi' \mid \Sigma'}\; x \notin fv(\Pi, E_1, E_2, \Sigma, \Pi', \Sigma')$$

This rule says that to prove that a ls entails a formula, it suffices to check if the lss of lengths zero and two[5] 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

[5] There is no need to consider length one because if the right-hand side accepts a list of length two then it also accepts a list of length one. The converse does not hold because of $\mapsto$.

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 $\mathsf{ls}(E_1, E_2)$, checking $\Pi \mid \Sigma \vdash \Pi' \mid \Sigma'$ is decidable.*

Proof (Sketch). 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. □

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

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 $y{\neq}z \mid \mathsf{ls}(x, y) * \mathsf{ls}(x, z)$, 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. 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:

$$\mathsf{ls}(-, E_2) \to (E_2 \not\hookrightarrow -) \tag{1}$$

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

$$(E_1{\neq}E_2 \land \mathsf{ls}(-, E_2) \land - \hookrightarrow E_1) \to (E_1 \hookrightarrow -) \tag{2}$$

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

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

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) states that heaps containing segments from E_1 to E_4 ($\mathsf{ls}(E_1, E_4)$) via a segment from E_2 to E_3 ($\wedge(\mathsf{ls}(E_2, E_3) * \mathsf{true})$) can be split into a heap containing the subsegment ($\mathsf{ls}(E_2, E_3)$) which, due to acyclicity, must not contain the endpoint ($\wedge E_4 \not\hookrightarrow -$), and ($*$) a heap which when augmented with *any* heap containing a segment from E_2 to E_3 without E_4 ($\mathsf{ls}(E_2, E_3) \wedge E_4 \not\hookrightarrow -$) yields ($-\!\!*$) a segment from E_1 to E_4 ($\mathsf{ls}(E_1, E_4)$). 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 lss.

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* [6] 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.

The general property we need is expressed in the following key lemma:

Lemma 5.

$$\begin{aligned} &\textit{If} && \Pi \mathrel{¦} \mathsf{ls}^2(E_2, E_3) * \Sigma \vdash \Pi' \mathrel{¦} \Sigma' && (4) \\ &\textit{and} && s\,, h \vDash \Pi \wedge E_2{\neq}E_3 \wedge E_2 \not\hookrightarrow - \wedge \Sigma && (5) \\ &\textit{then} && s\,, h \vDash \Pi' \wedge (\mathsf{ls}(E_2, E_3) -\!\!* \ \Sigma') && \end{aligned}$$

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. The proof of this lemma is omitted for space reasons. But 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 $s\,, h \vDash A * B$, 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 $\exists x, y.\, E_1{\mapsto}x * x{\mapsto}y * \mathsf{ls}(y, E_2)$, which changing models of antecedents could violate. Finally, allowing "unspecified" sharing with formulæ such as $\mathsf{ls}(x, y) \wedge \Sigma$ 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 lss, 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:

$$\Pi \land E_1{=}E_2 \mid \Sigma \vdash \Pi' \mid \Sigma' \tag{6}$$

$$\Pi \land E_1{\neq}E_2 \land x{\neq}E_2 \mid E_1{\mapsto}x * x{\mapsto}E_2 * \Sigma \vdash \Pi' \mid \Sigma' \tag{7}$$

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

$[\llbracket E_1 \rrbracket s = \llbracket E_2 \rrbracket s]$: Hence $s\,,h \vDash \Pi \land E_1{=}E_2 \mid \Sigma$, and so by (6), $s\,,h \vDash \Pi' \mid \Sigma'$.

$[\llbracket E_1 \rrbracket s \neq \llbracket E_2 \rrbracket s]$: Hence $h = h_{12}{*}h_\Sigma$ and there exists $l.\, s'\,, h_{12} \vDash E_1{\mapsto}x{*}\mathsf{ls}(x, E_2)$ and $s'\,, h_\Sigma \vDash \Pi \land E_1{\neq}E_2 \mid \Sigma$ where $s' = [s \mid x{\to}l]$ for x fresh. Therefore by (7), Lemma 5 ensures $s'\,, h_\Sigma \vDash \Pi' \mid (\mathsf{ls}(E_1, E_2) \mathrel{-\!\!*} \Sigma')$, and hence $s\,,h \vDash \Pi' \mid \Sigma'$. □

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. □

4.1 Decidability and Completeness

The proof-search algorithm makes use of a class of formulæ which are "maximally explicit". The primary characteristic of these formulæ, discussed later, is that the Frame rule is complete for entailments with such formulæ as antecedents.

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

$$\begin{aligned}\Pi \mid \Sigma \equiv\ & (x_i{\neq}x_j)_{1\le i\neq j\le n} \land (x_i{\neq}\mathsf{nil})_{1\le i\le n} \land (E_i{\neq}E'_i)_{1\le i\le m} \land \mathsf{true} \\ & \mid x_1{\mapsto}E''_1 * \cdots * x_n{\mapsto}E''_n * \mathsf{emp}\end{aligned}$$

for some n, m and where $x_i \not\equiv x_j$ for $i \neq j$ and $E_i \not\equiv E'_i$.

Table 2. Proof System

Axiom

$$\frac{}{\Pi ¦ \mathsf{emp} \vdash \mathsf{true} ¦ \mathsf{emp}}$$

Inconsistent

$$\frac{}{\Pi \wedge E{\neq}E ¦ \Sigma \vdash \Pi' ¦ \Sigma'}$$

Substitution

$$\frac{\Pi[E/x] ¦ \Sigma[E/x] \vdash \Pi'[E/x] ¦ \Sigma'[E/x]}{\Pi \wedge x{=}E ¦ \Sigma \vdash \Pi' ¦ \Sigma'}$$

=ReflexiveL

$$\frac{\Pi ¦ \Sigma \vdash \Pi' ¦ \Sigma'}{\Pi \wedge E{=}E ¦ \Sigma \vdash \Pi' ¦ \Sigma'}$$

nilNotLval

$$\frac{\Pi \wedge E_1{\neq}\mathsf{nil} ¦ E_1{\mapsto}E_2 * \Sigma \vdash \Pi' ¦ \Sigma'}{\Pi ¦ E_1{\mapsto}E_2 * \Sigma \vdash \Pi' ¦ \Sigma'}$$

$*$Partial

$$\frac{\Pi \wedge E_1{\neq}E_3 ¦ E_1{\mapsto}E_2 * E_3{\mapsto}E_4 * \Sigma \vdash \Pi' ¦ \Sigma'}{\Pi ¦ E_1{\mapsto}E_2 * E_3{\mapsto}E_4 * \Sigma \vdash \Pi' ¦ \Sigma'}$$

UnrollCollapse

$$\frac{\begin{array}{c}\Pi \wedge E_1{=}E_2 ¦ \Sigma \vdash \Pi' ¦ \Sigma' \\ \Pi \wedge E_1{\neq}E_2 \wedge x{\neq}E_2 ¦ E_1{\mapsto}x * x{\mapsto}E_2 * \Sigma \vdash \Pi' ¦ \Sigma'\end{array}}{\Pi ¦ \mathsf{ls}(E_1, E_2) * \Sigma \vdash \Pi' ¦ \Sigma'}\ x \notin \mathit{fv}(\Pi, E_1, E_2, \Sigma, \Pi', \Sigma')$$

=ReflexiveR

$$\frac{\Pi ¦ \Sigma \vdash \Pi' ¦ \Sigma'}{\Pi ¦ \Sigma \vdash \Pi' \wedge E{=}E ¦ \Sigma'}$$

Hypothesis

$$\frac{\Pi \wedge P ¦ \Sigma \vdash \Pi' ¦ \Sigma'}{\Pi \wedge P ¦ \Sigma \vdash \Pi' \wedge P ¦ \Sigma'}$$

Emptyls

$$\frac{\Pi ¦ \Sigma \vdash \Pi' ¦ \Sigma'}{\Pi ¦ \Sigma \vdash \Pi' ¦ \mathsf{ls}(E, E) * \Sigma'}$$

Frame

$$\frac{\Pi ¦ \Sigma \vdash \Pi' ¦ \Sigma'}{\Pi ¦ S * \Sigma \vdash \Pi' ¦ S * \Sigma'}$$

NonEmptyls

$$\frac{\Pi \wedge E_1{\neq}E_3 ¦ \Sigma \vdash \Pi' ¦ \mathsf{ls}(E_2, E_3) * \Sigma'}{\Pi \wedge E_1{\neq}E_3 ¦ E_1{\mapsto}E_2 * \Sigma \vdash \Pi' ¦ \mathsf{ls}(E_1, E_3) * \Sigma'}$$

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

Algorithm 8. *For goal entailment g, PS(g) either fails or returns a proof of g:*

```
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 $E_1{\neq}\mathsf{nil} \notin \Pi$            (8)
              and if $r$ is $*$Partial, then $E_1{\neq}E_3 \notin \Pi$             (9)
              and if $r$ is Frame or NonEmptyls,
                  then the antecedent of $g$ is in normal form            (10)
          if no such rule exists, then fail
          else if $r$ is an axiom, then return $r$
          else let $p_0, \ldots, p_n$ for some $n$ be the premisses of $r$ after applying $s$
              in return $r(\text{PS}(p_0), \ldots, \text{PS}(p_n))$
```

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. 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 $\Pi ¦ \Sigma \vdash \Pi' ¦ \Sigma'$ is a triple of:*

1. *the number of* ls*s occurring in* $\Pi ¦ \Sigma \vdash \Pi' ¦ \Sigma'$*,*
2. *the number of inequalities missing from* Π*, that is,* $|\{E_0{\neq}E_1 \mid E_0, E_1 \in fv(\Pi ¦ \Sigma, \Pi' ¦ \Sigma') \cup \{\mathsf{nil}\}\} \setminus \Pi|$*,*
3. *the length of* $\Pi ¦ \Sigma \vdash \Pi' ¦ \Sigma'$*, 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* NonEmptyls*, is in normal form.*

For a more intuitive characterization of normal form, note that formulæ $\Pi ¦ \Sigma$ 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 $\Pi ¦ \Sigma$ in normal form:*

1. *A* bad model of $\Pi ¦ \Sigma$ *is a state $s\,, h \vDash \Pi ¦ \Sigma$ where $nil \notin range(s)$ and s is one-one on $fv(\Pi ¦ \Sigma)$, and h is uniquely determined by s .*
2. *A* bad model of $\Pi ¦ \Sigma$ with $x{=}E$ *is a state $s\,, h \vDash \Pi{\wedge}x{=}E ¦ \Sigma$ where, for $s'\,, h'$ a bad model of $\Pi ¦ \Sigma$, $s = [s' \mid x{\to}[\![E]\!]s']$, and h is uniquely determined by s.*

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

1. *There exists a bad model of $\Pi ¦ \Sigma$.*
2. *For any $x{\neq}E \notin \Pi$, there exists a bad model of $\Pi ¦ \Sigma$ 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.[6]

[6] 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 $\Pi \mid \Sigma \vdash \Pi' \mid \Sigma'$, whose antecedent, by Observation 11, is in normal form. Proceed by cases:

[$\Sigma' \equiv \mathsf{emp}$ and $\Pi' \equiv \Pi'' \wedge E{=}E'$]: Note $E \not\equiv E'$ since $\Pi \mid \Sigma \vdash \Pi' \mid \Sigma'$ is stuck. Therefore a bad model of $\Pi \mid \Sigma$ is a countermodel.

[$\Sigma' \equiv \mathsf{emp}$ and $\Pi' \equiv \Pi'' \wedge E{\neq}E'$]: Note $E{\neq}E' \notin \Pi$ since $\Pi \mid \Sigma \vdash \Pi' \mid \Sigma'$ is stuck. Therefore a bad model of $\Pi \mid \Sigma$ with $E{=}E'$ is a countermodel.

[$\Sigma' \equiv E{\mapsto}E' * \Sigma''$]: Therefore since $\Pi \mid \Sigma \vdash \Pi' \mid \Sigma'$ is stuck, $E{\mapsto}E' \notin \Sigma$. Hence, $s\,, h$ a bad model of $\Pi \mid \Sigma$ is a countermodel, since either $[\![E]\!]s \notin \mathit{dom}(h)$ or $h([\![E]\!]s) \neq [\![E']\!]s$.

[$\Sigma' \equiv \mathsf{ls}(\mathsf{nil}, E) * \Sigma''$]: Therefore $s\,, h$ a bad model of $\Pi \mid \Sigma$ is a countermodel, since $\mathit{nil} \neq [\![E]\!]s$.

[$\Sigma' \equiv \mathsf{ls}(x, E) * \Sigma''$ and for all $E'.\, x{\mapsto}E' \notin \Sigma$]: Therefore $s\,, h$ a bad model of $\Pi \mid \Sigma$ is a countermodel, since $[\![x]\!]s \neq [\![E]\!]s$ and $[\![x]\!]s \notin \mathit{dom}(h)$.

[$\Sigma \equiv x{\mapsto}E * \Sigma_0$ and $\Sigma' \equiv \mathsf{ls}(x, E') * \Sigma_1$]: Note that Σ_1 contains only lss, since the other cases have already been covered. Let $s\,, h$ be a bad model of $\Pi \mid \Sigma$ with $x{=}E'$ ($x{\neq}E' \notin \Pi$ since $\Pi \mid \Sigma \vdash \Pi' \mid \Sigma'$ is stuck). Therefore $s\,, h \vDash \Pi \mid x{\mapsto}x * \Sigma_0$ and $s\,, h \nvDash \Pi' \mid \mathsf{ls}(x, E') * \Sigma_1$, since no ls contains a nonempty cycle. Therefore $s\,, h$ is a countermodel. □

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.*

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, all the entailments down to the goal are invalid. □

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

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., [7–10]) but, frankly, we are not sure if it is possible to obtain a canonical decidable fragment that covers a large variety of structures. For example, decidability of monadic second-order logic with a unary function symbol [7] implies decidability of our fragment. However, that result is only applicable because we used unary heap cells, while our techniques generalize to n-ary heap cells (necessary for binary trees for example).

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.

ACKNOWLEDGEMENTS. We are grateful to the anonymous referees for helpful comments. During Berdine's stay at Carnegie Mellon University, this research was sponsored in part by National Science Foundation Grant CCR-0204242. All three authors were supported by the EPSRC.

References

1. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: LICS, 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: POPL, London (2001) 39–46
4. O'Hearn, P., Reynolds, J., Yang, H.: Local reasoning about programs that alter data structures. In: CSL. Volume 2142 of LNCS., 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: FSTTCS. Volume 2245 of LNCS., Springer (2001) 108–119
6. O'Hearn, P.W., Yang, H., Reynolds, J.C.: Separation and information hiding. In: POPL, Venice (2004) 268–280
7. Rabin, M.O.: Decidability of secon-order theories and automata on infinite trees. Trans. of American Math. Society **141** (1969) 1–35
8. Jenson, J., Jorgensen, M., Klarkund, N., Schwartzback, M.: Automatic verification of pointer programs using monadic second-order logic. In: PLDI. (1997) 225–236 SIGPLAN Notices 32(5).
9. Benedikt, M., Reps, T., Sagiv, M.: A decidable logic for describing linked data structures. In: ESOP. Volume 1576 of LNCS., Springer (1999) 2–19
10. Immerman, N., Rabinovich, A., Reps, T., Sagiv, M., Yorsh, G.: Verification via structure simulation. In: CAV. Volume 3114 of LNCS. (2004)