dr. heap

A sound and complete proof system for separation logic (part 1)

Hans-Dieter A. Hiep Frank S. de Boer

June 29, 2024

1 Introduction

In this article we have another look at the proof system for separation logic that is introduced in the first author's PhD thesis [5] (publicly defended on Thursday, May 23rd, 2024).

By separation logic we mean the logic behind the assertion language used in Reynolds' logic, the program logic for reasoning about the correctness of pointer programs that was introduced in 2002 by J.C. Reynolds [9]. In that article, Reynolds introduces both his program logic and axiomatizes the logic of separation logic by several axioms, but writes:

> Finally, we give axiom schemata for the predicate $\mapsto$.
> (Regrettably, these are far from complete.)

In 2021, completeness of quantifier-free separation logic was established [3], and three year later completeness of the full language of separation logic [5].

The purpose of this article is to show the novel proof system of [5] in a straightforward way. The new proof system can be used to prove *all* valid formulas, which until now were impossible to prove using existing automatic and interactive tools for separation logic. In Section 2 we quickly revisit the formulas of separation logic, in Section 3 we introduce the proof system, and in Section 4 we have a look at a number of example proofs. We then continue the discussion that motivates the design of the proof system: in Section 5 we discuss referential transparency and the binding structure of separation logic, and in Section 6 we discuss issues such as univalence, well-foundedness, and finiteness.

This article is **part one** of a series of articles about the new proof system for separation logic. In this article, we focus on the syntax of the proof system. In Section 7, the conclusion, we discuss the topics of the next parts of this series, namely semantics and the soundness and completeness of the proof system.

2 Preliminaries

The syntax of formulas of separation logic is defined as follows:

$$\phi, \psi ::= \bot \mid (x \hookrightarrow y) \mid P(x_1, \ldots, x_n) \mid (\phi \to \psi) \mid (\forall x)\phi \mid (\phi * \psi) \mid (\phi \mathrel{-\!\!*} \psi)$$

where we assume there is a countably infinite set of variables V with typical examples x, y, z (possibly with subscripts), and we have a signature which has a countably infinite set of non-logical symbols each assigned to a fixed arity of which P is a typical example with arity n. We have the usual logical symbols: $\bot$ stands for falsity and $(\phi \to \psi)$ stands for logical implication. From these two symbols we can derive all other propositional connectives, such as negation $\neg\phi$, verum $\top$, logical conjunction $(\phi \wedge \psi)$, and logical disjunction $(\phi \vee \psi)$. We have universal quantification $(\forall x)\phi$ where the variable x is bound in the usual way, and we can define existential quantification $(\exists x)\phi$ as the dual $\neg(\forall x)\neg\phi$. By $FV(\phi)$ we mean the set of free variables in ϕ. Quantification is first-order in the sense that quantification ranges over individuals. Finally, we also have equality $(x = y)$ as a non-logical symbol, but with a fixed meaning. (Our treatment of parentheses and resolution of ambiguity is standard: we may leave parentheses out as long as the result is not too ambiguous.)

What is different in separation logic compared to classical first-order logic are the following so-called *separation symbols* (distinguished from the logical and non-logical symbols). The primitive formula $(x \hookrightarrow y)$ is called *points to* (as in 'x points to y') or a *reference* (as in 'x is a reference to y'). As complex formulas, two separating connectives are given: $(\phi * \psi)$ is a *separating conjunction*, and $(\phi \mathrel{-\!\!*} \psi)$ is a *separating implication*. The latter connective is also called the *magic wand* by some authors. Having 'points to' $\hookrightarrow$ as primitive allows us to define 'strict points to' $\mapsto$ as follows, where we take $(x \mapsto y)$ to abbreviate

$$(x \hookrightarrow y) \wedge (\forall z, w.\, (z \hookrightarrow w) \to x = z).$$

The intention is that $(x \hookrightarrow y)$ expresses that location x has value y, whereas $(x \mapsto y)$ expresses furthermore that x is the only location allocated. We also have the abbreviations $(x \hookrightarrow -)$ and $(x \mapsto -)$, where we immediately existentially quantify away the value. These express that x is allocated (possibly among other locations) and, moreover, that only x is allocated. By **emp** we mean nothing is allocated, so it abbreviates $\forall x(\neg(x \hookrightarrow -))$, i.e. every location x does not point to some value, or equivalently $\forall x, y.\, \neg(x \hookrightarrow y)$.

3 Proof system

In this section we introduce a novel proof system for separation logic. In this article we look at the proof system from a purely syntactical point of view. In the next article of this series, we give the standard semantics of separation logic.

The first device we introduce is a special **let** construct, in the following sense:

$$\textbf{let } (x \hookrightarrow y) := \psi(x, y) \textbf{ in } \phi.$$

This construct allows us to change the meaning of 'point to' in ϕ, by assigning it the meaning denoted by $\psi(x, y)$. Intuitively speaking, to evaluate whether **let** $(x \hookrightarrow y) := \psi(x, y)$ **in** ϕ holds, we first consider the heap denoted by ψ (with free variables x and y) and then we evaluate whether ϕ holds in the heap

described by ψ. We must be careful not having a too naïve interpretation of **let**: we cannot just simplify by replacing the occurrences of $(x \hookrightarrow y)$ in ϕ by $\psi(x, y)$, because separating connectives are referentially opaque (this is explained in more detail in Section 5). The purpose of our proof system is to reason about this **let** construct in a formal way.

Working with **let** takes much space, so instead we use the shorthand notation $\phi @_{x,y} \psi$. Thus, the objects of our proof system involve not just the formulas of separation logic, but an extended language (called extended separation logic) in which we add this special construct:

$$\phi, \psi ::= \ldots \mid (\phi @_{x,y} \psi)$$

Next, we introduce a proof system with as objects the formulas of extended separation logic. This proof system allows us to deduce formulas: a deduction is also called a proof, and we shall give a number of example proofs. Recall that we have a signature that has a countable infinite supply of non-logical symbols. For any formula of extended separation logic, its parameters are the predicate symbols of the signature that occur somewhere in the formula. In particular, we shall make use of so-called 'bookkeeping devices', which are binary predicate symbols R (possibly with quotes or subscripts) from the signature. Sometimes we have the side-conditions that our bookkeeping devices are 'fresh', in the sense that they do not appear as parameters of formulas in the context.

We present the proof system as a Hilbert-style axiom system, but nothing prevents us from also giving the proof system in the style of natural deduction. We have the usual proof rules and axioms of classical logic (but instantiated with formulas of extended separation logic), together with the following axioms:

- $\phi \leftrightarrow (\phi @ (x \hookrightarrow y))$ (Lookup)
- $((x' \hookrightarrow y') @ \psi) \leftrightarrow \psi[x, y := x', y']$ (Replace)
- $(\mathbf{false} @ \phi) \leftrightarrow \mathbf{false}$ ($@\bot$)
- $((\phi \to \psi) @ \chi) \leftrightarrow (\phi @ \chi \to (\psi @ \chi))$ ($@\to$)
- $((\forall x \phi) @ \psi) \leftrightarrow (\forall x)(\phi @ \psi)$ if $x \notin FV(\psi)$ ($@\forall$)
- $(\phi @ (\psi @ \chi)) \leftrightarrow ((\phi @ \psi) @ \chi)$ (Assoc)
- $(\forall x, y(\psi \leftrightarrow \chi)) \to ((\phi @ \psi) \leftrightarrow (\phi @ \chi))$ (Extent)
- $((\phi * \psi) @ \chi) \to (\chi = R_1 \uplus R_2 \to (\phi @ R_1) \to (\psi @ R_2) \to \xi) \to \xi$ ($*$E)
- $\chi = \chi_1 \uplus \chi_2 \to (\phi @ \chi_1) \to (\psi @ \chi_2) \to ((\phi * \psi) @ \chi)$ ($*$I)
- $((\phi \mathrel{-\!\!*} \psi) @ \chi) \to (\chi \perp \chi') \to (\phi @ \chi') \to ((\psi @ (\chi \vee \chi')) \to \xi) \to \xi$ ($-\!\!*$E)
- $(\chi \perp R \to (\phi @ R) \to (\psi @ (\chi \vee R(x, y)))) \to ((\phi \mathrel{-\!\!*} \psi) @ \chi)$ ($-\!\!*$I)

We have the side-condition in the rule ($*$E) that the symbols R_1 and R_2 are fresh, i.e. are not parameters of ϕ, ψ, χ, ξ. Similarly, we have the side-condition in the rule ($-\!\!*$I) that the symbol R is fresh, i.e. is not a parameter of ϕ, ψ, χ. We used @ without subscripts instead of $@_{x,y}$ to reduce notational clutter. To avoid confusion, we may use **false** instead of $\bot$ and **true** instead of $\top$.

$\psi[x, y := x', y']$ is the result of simultaneous substitution of x by x' and y by y', respectively. The substitution operator $\phi[x := x']$ is defined compositionally as usual, and has the following specification for the new connective:

$$(\phi @_{x,y} \psi)[z := z'] = (\phi[z := z'] @_{x,y} \psi[z := z'])$$

where x, y, z are all distinct. If z is either the same variable as x or y, then the substitution is not pushed down on the right side. A similar definition can be given for simultaneous substitution of distinct variables.

We let $\chi = \chi_1 \uplus \chi_2$ abbreviate the formula

$$(\chi \equiv \chi_1 \cup \chi_2) \wedge (\chi_1 \perp \chi_2)$$

and let $\chi \equiv \chi_1 \cup \chi_2$ abbreviate the formula

$$\forall x, y (\chi(x, y) \leftrightarrow \chi_1(x, y) \vee \chi_2(x, y))$$

and let $\chi_1 \perp \chi_2$ abbreviate the formula

$$\forall x, y \big(\chi_1(x, y) \rightarrow \forall z. \neg \chi_2(x, z)\big).$$

These abbreviations universally quantify x, y: we let these quantifiers, on purpose, capture the free variables x and y of χ, χ_1, χ_2. When χ_1 and χ_2 are just the binary predicate symbols R_1 and R_2, we mean the formulas $R_1(x, y)$ and $R_2(x, y)$. One can also use set builder notation to make the intention more clear. Note that in the latter abbreviation, $\chi_1 \perp \chi_2$, we require the stronger notion of disjointness of the domains of the relation, not the weaker notion of disjointness of the two sets of pairs representing the pairs that are related by each relations.

Further, a useful result in practical reasoning is that we can replace equivalent subformulas in any formula. Moreover, the deduction theorem also holds for our proof system, hence we can apply the axioms under any context. We furthermore shall use the above proof system in a natural deduction style.

4 Example proofs

Let us now have a look at a number of example proofs. We shall write $\vdash \phi$ to mean that ϕ is demonstrable in the proof system given above without any premises, and $\Gamma \vdash \phi$ to mean that ϕ is demonstrable using the premises in Γ.

The first example is given in Figure 1. The statement we want to prove has the following intuitive meaning: in the heap described by $\bot$ we have that **emp** is satisfied. The argument is the following: the heap described by $\bot$ is the empty graph (no location is mapped to any value), so evaluating **emp** in

Proposition. $\vdash \mathbf{emp}@\bot$.

Proof. Recall **emp** abbreviates $\forall x, y.\, \neg(x \hookrightarrow y)$, and $\neg\phi$ abbreviates $\phi \to \bot$.
1. $\forall x, y.\, (\bot \to \bot)$ is trivial from first-order reasoning.
2. $\forall x, y.\, (\bot \to (\bot@\bot))$ by rewriting using $(@\bot)$ in 1.
3. $\forall x, y.\, ((x \hookrightarrow y)@\bot \to (\bot@\bot))$ by rewriting using (Replace) in 2.
4. $\forall x, y.\, ((x \hookrightarrow y) \to \bot)@\bot$ by rewriting using $(@\to)$ in 3.
5. $(\forall x, y.\, \neg(x \hookrightarrow y))@\bot$ by applying $(@\forall)$ twice in 4. *Qed.*

Figure 1. Proof of **emp** in the empty heap.

Proposition. $\vdash \phi * \psi \to \psi * \phi$.

Proof.
1. $(\phi * \psi)@(x \hookrightarrow y)$ is assumed.
   2. $\{(x, y) \mid (x \hookrightarrow y)\} = R_1 \uplus R_2$ is assumed. (R_1, R_2 fresh)
   3. $\phi@R_1$ is assumed.
   4. $\psi@R_2$ is assumed.
   5. $\{(x, y) \mid (x \hookrightarrow y)\} = R_2 \uplus R_1$ by using first-order reasoning from 2.
   6. $(\psi * \phi)@(x \hookrightarrow y)$ by using $(*\mathrm{I})$ on 5,4,3.
7. $(\psi * \phi)@(x \hookrightarrow y)$ by using $(*\mathrm{E})$ on 1,(2–6).
8. $(\phi * \psi)@(x \hookrightarrow y) \to ((\psi * \phi)@(x \hookrightarrow y))$ by deduction theorem on (1–7).
9. $(\phi * \psi \to \psi * \phi)@(x \hookrightarrow y)$ by using $(@\to)$ in 8.
10. $\phi * \psi \to \psi * \phi$ by using (Lookup) in 9. *Qed.*

Figure 2. Proof of commutativity of $*$.

that heap indeed yields a true formula. In the proof that follows, we do not explicitly write down how to do classical reasoning, and instead we focus on the application of the new axioms.

The second example is given in Figure 2. We prove that for any (extended) separation logic formulas ϕ and ψ, their separating conjunction is commutative. The proof proceeds in two parts. In step 9, we have shown how to swap the two separated formulas relative to the heap $(x \hookrightarrow y)$. But this heap description has the same extension as the 'outer' heap, hence we obtain the non-relative result in step 10! As such, we can obtain the result simply by putting the given formula under this @-connective. We add formulas to the context by means of opening a box, so at step 6 we have established:

$$(\phi * \psi)@(x \hookrightarrow y), (x \hookrightarrow y) = R_1 \uplus R_2, \phi@R_1, \psi@R_2 \vdash (\psi * \phi)@(x \hookrightarrow y).$$

See Figure 3 and Figure 4 for the third and fourth examples. Figure 3 is a generalization of the result in Figure 1. Note that in step 5 of Figure 4 we use

Proposition. $\vdash (\mathbf{emp}@\phi) \leftrightarrow (\forall x, y.\, \neg\phi(x, y))$.

Proof. Recall **emp** abbreviates $\forall x, y.\, \neg(x \hookrightarrow y)$, and $\neg\phi$ abbreviates $\phi \to \bot$.

1. $\mathbf{emp}@\phi$ is assumed.
   2. $\phi(x_0, y_0)$ is assumed. (x_0, y_0 fresh)
   3. $\forall x, y.\, (\neg(x \hookrightarrow y))@\phi$ by (@∀) in 1.
   4. $(\neg(x_0 \hookrightarrow y_0))@\phi$ from 3.
   5. $(x_0 \hookrightarrow y_0))@\phi$ by (Replace) from 2.
   6. $\bot@\phi$ by (@→) by modus ponens from 4 and 5.
   7. $\bot$ by (@⊥) from 6.
   8. $\forall x, y.\, \phi(x, y) \to \bot$ from (2–7).
9. $\forall x, y.\, \phi(x, y) \to \bot$ is assumed.
   10. $(x_0 \hookrightarrow y_0)@\phi$ is assumed. (x_0, y_0 fresh)
   11. $\phi(x_0, y_0)$ by (Replace) from 10.
   12. $\bot$ by instantiation and modus ponens from 9 and 11.
   13. $\forall x, y.\, ((x \hookrightarrow y)@\phi \to \bot)$ from (10–12)
   14. $\mathbf{emp}@\phi$ by (@∀) and (@→) and (@⊥) in 13.
15. $(\mathbf{emp}@\phi) \leftrightarrow (\forall x, y.\, \neg\phi(x, y))$ from (1–8) and (9–14). *Qed.*

Figure 3. Proof that **emp** holds and only holds in empty heaps.

the result proven in Figure 3. What should be obvious now is that the proofs are not very difficult: we use our set theoretic intuition for dealing with heaps. Both Figure 2 and Figure 4 show that (∗E) simply adds fresh parameters R_1, R_2 and the corresponding assumptions to the context. This shows that separating connectives behave almost like a quantifier, if we compare it with the way first-order quantification works (as in Figure 3).

The reader can now try and write down the proofs for the following formulas:

- $\vdash (\phi \vee \psi) * \chi \leftrightarrow \phi * \chi \vee \psi * \chi$,
- $\vdash (\phi \wedge \psi) * \chi \to \phi * \chi \wedge \psi * \chi$,
- $\vdash (\exists x \phi(x)) * \psi \leftrightarrow \exists x(\phi(x) * \psi)$,
- $\vdash (\forall x \phi(x)) * \psi \to \forall x(\phi(x) * \psi)$,
- $\vdash \phi * (\phi \mathrel{-\!\!*} \psi) \to \psi$.

At last, we have the following non-trivial properties:

- $\vdash (x \hookrightarrow y) \leftrightarrow (x \mapsto y) * \top$,
- $\vdash \neg(x \hookrightarrow -) \to (((x \mapsto y) \mathrel{-\!\!*} (x \mapsto y) * \phi) \leftrightarrow \phi)$,
- $\vdash ((\exists x(x \hookrightarrow y)) * (\exists x(x \hookrightarrow y))) \leftrightarrow (\exists x((x \hookrightarrow y) \wedge \exists z(z \neq x \wedge (z \hookrightarrow y))))$.

Proposition. $\vdash \phi * \mathbf{emp} \leftrightarrow \phi$.

Proof. Recall that $\chi = \chi_1 \uplus \chi_2$ abbreviates $(\chi \equiv \chi_1 \cup \chi_2) \wedge (\chi_1 \perp \chi_2)$.

1. $(\phi * \mathbf{emp})@(x \hookrightarrow y)$ is assumed.
   2. $\{(x,y) \mid (x \hookrightarrow y)\} = R_1 \uplus R_2$ is assumed. (R_1, R_2 fresh)
   3. $\phi@R_1$ is assumed.
   4. $\mathbf{emp}@R_2$ is assumed.
   5. $\forall x, y.\, \neg R_2(x,y)$ from applying previous proposition to 4.
   6. $\forall x, y.\, R_1(x,y) \leftrightarrow (x \hookrightarrow y)$ from 2 and 5.
   7. $\phi@(x \hookrightarrow y)$ by (Extent) from 3 and 6.
   8. $\phi@(x \hookrightarrow y)$ by ($*$E) from 1,(2–7).
   9. $\phi@(x \hookrightarrow y)$ is assumed.
   10. $\mathbf{emp}@\perp$ by previous proposition.
   11. $\{(x,y) \mid (x \hookrightarrow y)\} = \{(x,y) \mid (x \hookrightarrow y)\} \uplus \perp$ is a first-order tautology.
   12. $(\phi * \mathbf{emp})@(x \hookrightarrow y)$ by ($*$I) from 11,9,10.
   13. $(\phi * \mathbf{emp})@(x \hookrightarrow y) \leftrightarrow (\phi@(x \hookrightarrow y))$ from (1–8),(9–12).
   14. $(\phi * \mathbf{emp}) \leftrightarrow \phi$ by ($@\perp$) and ($@\rightarrow$) from 13. *Qed.*

Figure 4. Proof that **emp** is a unit of separating conjunction.

The last property is very important. It shows that separation logic can be used to express cardinality properties of the universe. The last property shows the separation logic equivalent of the classical expression of the property 'there are at least two elements'. When we scale this property, to 'there are at least n elements', one will see that the separation logic formula grows linearly but the classical logic equivalent grows drastically faster: quadratically! This is the essence of the scalability argument motivating the use of separation logic.

Our proof system is able to prove this equivalence. However, existing proof systems for separation logic still lack the ability to prove this elementary fact. We have investigated whether the equivalence of these formulas can be proven in an interactive tool for reasoning about separation logic: the Iris project [6]. In current versions of that system, it is not possible to show the equivalence of these assertions, at least not without adding additional axioms.

The last example is a demonstration of the following equivalence:

$$\begin{array}{c}(x \hookrightarrow -) \wedge ((x = y \wedge z = w) \vee (x \neq y \wedge (y \hookrightarrow z)))\\ \leftrightarrow \\ (x \mapsto -) * ((x \mapsto w) \mathrel{-\!\!*} (y \hookrightarrow z)).\end{array}$$

This equivalence is expressed in quantifier-free separation logic, for which a complete axiomatization was already known [3]. We can also give a proof, see Figure 5. Surprisingly, this already exceeds the capability of all the automated separation logic provers in the benchmark competition SL-COMP. In fact, only

the CVC4-SL tool [8] supports the fragment of separation logic that includes the separating implication. However, from our own experiments with that tool, we have that it produces an incorrect counter-example and reported this as a bug to one of the maintainers of the project. In fact, the latest version, CVC5-SL, reports the same input as 'unknown', indicating that the tool is incomplete.

So far, we have seen several valid formulas of separation logic, which in the novel proof system for separation logic we are actually able to prove. This alone already shows our proof system goes beyond the ability of existing tools for reasoning about separation logic! The novelty of this proof system lies in the fact of adding a **let** binding construct, which in shorthand is written using the @-connective, that relativates the heap with respect to which a formula is interpreted.

5 Referential transparency

In this section we discuss the binding structure of separation logic, and how the concept of referential transparency applies. Referential transparancy is a general concept in formal languages and as such applies to both logical and programming languages. Although Whitehead and Russell already speak of it, Quine is often credited for introducing the term in his book *Word and Object* [7] originally published in 1960. In the case of separation logic, we shall see that the separating connectives fail referential transparency!

Separating connectives capture references, the 'points to', that occur in subformulas. In the binding structure of first-order logic, one could resolve unintentional capturing by means of a so-called 'capture avoiding' substitution operator that renames quantified variables before actually performing a substitution. However in separation logic, one cannot define such a capture avoiding substitution operator since in separation logic there is only a single heap in scope that can not be renamed.

First, we shall make some general remarks about the binding structure of separation logic formulas. A formula is *pure* if no separation symbol $\hookrightarrow, *, -\!\!*, @$ occurs in it. In that case the meaning of a formula does not depend on the heap, viz. the interpretation of $\hookrightarrow$. Otherwise, a formula is *semi-pure* if only the separation symbol $\hookrightarrow$ occurs in it. A formula in which one of the separating connectives occur is called a *separating* formula. We have the usual notions of free variable occurrence and bound variable occurrence, as our notion of variable binding is the same as in first-order logic. But, in separation logic, we also have another binding structure, namely that of references: the meaning of 'points to' is different under the separating connectives.

To see why separation logic fails referential transparency, consider the reference to 'the value of location y' in the proposition 'the value of location y has property P'. To avoid that 'the value of location y' is ill-defined, when speaking of *the* value one implicitly intends there exists a unique value. Moreover, linguistically speaking, a reference is *free* if we can replace it by any other expression that is equal to it, without affecting the truth value of the proposi-

Proposition. $\vdash ((x \hookrightarrow -) \wedge (x = y \wedge z = w) \vee (x \neq y \wedge (y \hookrightarrow z))) \leftrightarrow ((x \mapsto -) * ((x \mapsto w) \mathrel{-\!\!*} (y \hookrightarrow z)))$.

Proof.

1. $(x \hookrightarrow -)$ is assumed.
2. $(x = y \wedge z = w) \vee (x \neq y \wedge (y \hookrightarrow z))$ is assumed.

3. $\{(x_0, y_0) \mid x_0 \neq x \wedge (x_0 \hookrightarrow y_0)\} \perp R$ is assumed. (R fresh)
4. $(x \mapsto w)@R$ is assumed.

5. $(x = y \wedge z = w)$ is assumed.
6. $R(y, z)$ from 4 and 5.
7. $(y \neq x \wedge (y \hookrightarrow z)) \vee R(y, z)$.

8. $(x \neq y \wedge (y \hookrightarrow z))$ is assumed.
9. $(y \neq x \wedge (y \hookrightarrow z)) \vee R(y, z)$.

10. $(y \neq x \wedge (y \hookrightarrow z)) \vee R(y, z)$ from 2,(5–7),(8–9).
11. $(y \hookrightarrow z)@_{x_0,y_0}((x_0 \neq x \wedge (x_0 \hookrightarrow y_0)) \vee R(x_0, y_0))$ from 10.

12. $((x \mapsto w) \mathrel{-\!\!*} (y \hookrightarrow z))@_{x_0,y_0}(x_0 \neq x \wedge (x_0 \hookrightarrow y_0))$ by ($-\!\!*$I) fr. (3–11).
13. $(x \mapsto -)@_{x_0,y_0}(x_0 = x \wedge (x_0 \hookrightarrow y_0))$ from 1.
14. $\hookrightarrow = \{(x_0, y_0) \mid x_0 = x \wedge (x_0 \hookrightarrow y_0)\} \uplus \{(x_0, y_0) \mid x_0 \neq x \wedge (x_0 \hookrightarrow y_0)\}$
15. $((x \mapsto -) * ((x \mapsto w) \mathrel{-\!\!*} (y \hookrightarrow z)))@(x \hookrightarrow y)$ by ($*$I) from 14,13,12.
16. $(x \mapsto -) * ((x \mapsto w) \mathrel{-\!\!*} (y \hookrightarrow z))$ by (Lookup) from 14.

17. $(x \mapsto -) * ((x \mapsto w) \mathrel{-\!\!*} (y \hookrightarrow z))$ is assumed.
18. $((x \mapsto -) * ((x \mapsto w) \mathrel{-\!\!*} (y \hookrightarrow z)))@(x \hookrightarrow y)$ by (Lookup) from 17.

19. $\{(x, y) \mid (x \hookrightarrow y)\} = R_1 \uplus R_2$ is assumed. (R_1, R_2 fresh)
20. $(x \mapsto -)@R_1$ is assumed.
21. $((x \mapsto w) \mathrel{-\!\!*} (y \hookrightarrow z))@R_2$ is assumed.
22. $\{(x, w)\} \perp R_2$ from 19 and 20.
23. $(x \mapsto w)@_{x_0,y_0}(x_0 = x \wedge y_0 = w)$.

24. $(y \hookrightarrow z)@_{x_0,y_0}(R_2(x_0, y_0) \vee (x_0 = x \wedge y_0 = w))$ is assumed.
25. $R_2(y, z) \vee (y = x \wedge z = w)$ by (Replace) from 24.
26. $R_2(y, z) \rightarrow y \neq x \wedge (y \hookrightarrow z)$ from 19 and 20.
27. $((x = y \wedge z = w) \vee (x \neq y \wedge (y \hookrightarrow z)))$ from 25 and 26.

28. $((x = y \wedge z = w) \vee (x \neq y \wedge (y \hookrightarrow z)))$ by ($-\!\!*$E) fr. 21,22,23,(24–27).
29. $(x \hookrightarrow -)$ from 19 and 20.
30. $(x \hookrightarrow -) \wedge ((x = y \wedge z = w) \vee (x \neq y \wedge (y \hookrightarrow z)))$ from 28,29.

31. $(x \hookrightarrow -) \wedge ((x = y \wedge z = w) \vee (x \neq y \wedge (y \hookrightarrow z)))$ by ($*$E) fr. 18,(19–30).

Qed.

Figure 5. Proof of an equivalence between a semi-pure and separating formula.

tion after replacement compared to the proposition before replacement. Often this is called the principle of substitutivity[1]. For example, given that 'the value of location y' equals 'the value of location z', when we replace a reference of the former by the latter in the expression 'the value of location y has property P' to obtain 'the value of location z has property P', we obtain an equivalent proposition: so we have that the reference 'the value of location y' occurs free. A context is said to be *referentially transparent* whenever it preserves the free references: every free reference remains a free reference under the given context. Otherwise, the context is *referentially opaque*.

In classical logic all propositional connectives are referentially transparent. The only referentially opaque connectives are the quantifiers under specific circumstances. This is easy to see for a given formula $P(x)$ with a free variable x. Suppose $x = 5$, then by substitutivity we know that $P(5)$ is equivalent to $P(x)$. However, some quantifiers fail referential transparency, since for example in the formula $\exists x(P(x))$ we can no longer naïvely replace x with 5 when we know $x = 5$. If the quantified variable is not the same as one of the free variables (either in the subformula or in the expression being substituted), we do maintain referential transparency. To ensure referential transparency there is the convention of keeping bound and free variables separate, analogous to the so-called Barendregt variable convention [4, Sect. 5.2].

In separation logic, however, many contexts involving separating connectives are referentially opaque. For example, in the context of a separating conjunction it is not always the case that we can freely replace references by equivalent expressions. An example is where the value of location y is equal to the value of location z, and where we also separate the locations y and z. Formally, we have the equality on the left, and the separation on the right:

$$(\forall x_1.\,(y \hookrightarrow x_1) \to \forall x_2.\,(z \hookrightarrow x_2) \to x_1 = x_2) \wedge ((\exists x.\,(y \hookrightarrow x)) * (\exists x.\,(z \hookrightarrow x))).$$

Although we know that locations y and z have the same value, we cannot literally replace y for z in the left component of the separating conjunction, without also doing the reverse replacement (replacing z for y) in the other component of the separating connective. Thus we no longer have that the reference 'the value of location y' is free when it is nested under a separating conjunction: separating conjunction is referentially opaque!

To understand the binding structure of separation logic, we introduce the notions of *direct* and *indirect* binding. A reference (a 'points to' construct) or a separating connective is directly bound to the separating connective under which it is nested, without any other separating connective in between. Thus, here by nesting we only have to look at separating connectives in the immediate context, not at the logical connectives. A reference or a separating connective is said to be *free* whenever it is not directly bound. A reference or a separating connective is indirectly bound to all the separating connectives under which it is

[1] What is called a 'free reference' here comes from Quine's 'purely referential position'. But, we already use the word 'pure' in a different sense, namely that any formula that involves the 'points to' construct itself is not pure. Hence we use instead the term 'free reference'.

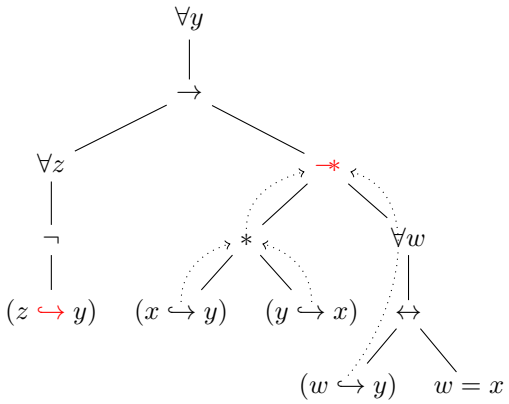

Figure 6. A parse tree showing the binding structure of separation logic. Direct bindings are shown with dotted lines pointing to a separating connective. Free references and free separating connectives are shown in red.

nested, but not immediately nested. In a sense, indirect binding is the transitive but irreflexive closure of direct binding.

Another example is the following formula involving magic wand:

$$\forall y.(\forall z(z \not\hookrightarrow y)) \to ((x \hookrightarrow y) * (y \hookrightarrow x) -\!\!* (\forall w.(w \hookrightarrow y) \leftrightarrow w = x))$$

which expresses the following concept: for every value y that the heap does not refer to, if we were to extend the heap with a cycle between the locations x and y, then in the resulting heap the location x is the only location which has value y. So how does the binding structure of this formula look like? Syntactically, there are four references ('points to' constructs) in this formula and two separating connectives. Each of these entities are either bound or free. The left-most reference is free, and the other three references are bound. These three references are nested under the magic wand, so directly or indirectly bound to that magic wand. The magic wand itself is free. The right-most reference is directly bound to the magic wand. The other two references are directly bound to the separating conjunction. See Figure 6 for a graphical depiction of the parse tree and the binding structure of references and separating connectives to separating connectives.

There is a difference with the variable binding structure of first-order logic: if a variable is bound to a quantifier, then it no longer necessarily has a relationship with the free variables of the same name. Quantifiers thus introduce a so-called *scope* for each variable. This is different for separation logic: although a reference can be directly bound to a separating connective, there still can be a necessary relationship with references that occur outside the connective to which it is bound. For example, in Figure 6 we have that both the free reference and the magic wand speak about the same heap (the 'outer' heap), but also the right-most reference under the magic wands speaks about (part of) that outer

heap: namely, for every $z \neq x$ we also have $\neg(z \hookrightarrow y)$ due to the equivalence on the right-side of the magic wand.

The moral is that separation logic has 'leaky scopes'. But it is also possible to define constructions in separation logic that have proper scopes. For example, the formula $\blacksquare\phi$ has the intuitive meaning that ϕ holds for all heaps (its formal definition is given in the next section). It thus acts as a universal quantifier for heaps. And we can also define $\blacklozenge\phi$ as the dual $\neg\blacksquare\neg\phi$, that acts as an existential quantifier for heaps. Just like quantifiers in first-order logic, we have that $\blacksquare\phi$ and $\blacklozenge\phi$ introduce a proper scope of the 'points to' construct inside ϕ, which is different from the 'points to' construct outside.

The formula $\blacksquare\phi$ is a so-called *heap independent* formula. A heap independent formula is a formula for which its truth value does not depend on the 'current' heap in which it is evaluated. For example, the pure fragment of separation logic, comprising no separation symbols, is heap independent. But also $\blacksquare\phi$ is heap independent, even when it contains 'points to' constructs and separating connectives in ϕ. All references and connectives under $\blacksquare$ are bound and the scope is closed: no 'leaky scope' for the black box.

6 Univalence, well-foundedness and finiteness

We now introduce the modality $\blacksquare\phi$ as the abbreviation

$$\mathbf{true} * (\mathbf{emp} \wedge (\mathbf{true} \mathrel{-\!\!*} \phi)).$$

We also have the dual $\blacklozenge\phi$ defined as $\neg\blacksquare\neg\phi$. We have that both modalities have the same binding strength as classical negation. The intuitive reading of the modal operators is that $\blacksquare\phi$ holds in a given 'current' heap whenever ϕ holds for all heaps (including the current heap), and $\blacklozenge\phi$ holds in a given current heap whenever ϕ holds in some heap (which may be different from the current heap). As such, these modal operators change the heap with respect to which a formula is evaluated.

In fact, we have the following valid formulas involving these modalities:

- $\vdash \blacksquare(\phi \to \psi) \to \blacksquare\phi \to \blacksquare\psi$,
- $\vdash \blacksquare\phi \to \phi$,
- $\vdash \blacksquare\phi \to (\phi @ \psi)$,
- $\vdash \blacksquare(\phi \to \phi') \to \blacksquare(\psi \to \psi') \to (\phi * \psi) \to (\phi' * \psi')$,
- $\vdash \blacksquare(\phi' \to \phi) \to \blacksquare(\psi \to \psi') \to (\phi \mathrel{-\!\!*} \psi) \to (\phi' \mathrel{-\!\!*} \psi')$.

These formulas can all be proven in our novel proof system (their proofs are great exercises for the reader). We also have that the rule of necessitation is admissible, in the sense that $\vdash \phi$ implies $\vdash \blacksquare\phi$, but whether this rule is also effectively derivable is not known to us.

We now consider an example of using the black box modality ■. In our treatment of separation logic, we do not necessarily impose so-called 'functionality' or 'univalence' of the heap. This means that it is possible that

$$(x \hookrightarrow y) \wedge (x \hookrightarrow z) \wedge y \neq z$$

is true in some situation. We thus treat $\hookrightarrow$ as a relation symbol. One intuitive way to interpret the 'points to' relation would be from object-oriented programming, where the object x has some reference to the object y by one of its fields, but we abstract away through which field object x references object y. It is not difficult to obtain univalence by restricting ourselves to those situations where there is at most one value, by means of the property:

$$\forall x, y, z.\, (x \hookrightarrow y) \wedge (x \hookrightarrow z) \to y = z.$$

That *all* heaps are univalent can be simply expressed by:

$$\blacksquare(\forall x, y, z.\, (x \hookrightarrow y) \wedge (x \hookrightarrow z) \to y = z).$$

We also have the following modality $\Box\phi$, introduced as the abbreviation

$$\neg(\top * \neg\phi).$$

Also the dual $\Diamond\phi$ is defined as $\neg\Box\neg\phi$. The intuitive reading of these modal operators is different, in the sense that $\Box\phi$ holds in a given heap whenever ϕ holds for all subheaps of the given heap. Similarly, $\Diamond\phi$ holds in a given heap whenever ϕ holds for some subheap of the given heap.

An example of the $\Box$ modality is the following. We say that a value x is *reachable* if there is a location y which refers to it, so $\exists y.(y \hookrightarrow x)$. Conversely, a location y is *allocated* whenever it refers to a value, so $\exists x.(y \hookrightarrow x)$. Consider that allocated locations can also be used as values, so we can have an allocated and reachable location. This way, we can form chains of so-called traversals:

$$x_0 \hookrightarrow x_1 \hookrightarrow x_2 \hookrightarrow \ldots \hookrightarrow x_n$$

which abbreviates the conjunction of $x_i \hookrightarrow x_{i+1}$ for $0 \leq i < n$. Whenever x_n is not allocated, the traversal has reached a dead-end. However, whenever in a traversal the first and last location are the same, we have a cycle: it is then possible to keep on traversing the heap indefinitely.

We say that a heap is *well-founded* whenever for every non-empty subheap there is some allocated but unreachable location. This is expressed formally as:

$$\Box(\neg\mathbf{emp} \to \exists x.(x \hookrightarrow -) \wedge \forall y.(y \not\hookrightarrow x)).$$

The claim is now that there are no cycles in a well-founded heap. To see why, suppose towards contradiction we have a well-founded heap (in which the above formula is true) in which there exists a cycle

$$x_0 \hookrightarrow x_1 \hookrightarrow x_2 \hookrightarrow \ldots \hookrightarrow x_n \hookrightarrow x_0.$$

Then take the subheap which consists precisely of the locations $\{x_0, \ldots, x_n\}$, that is, we ignore all the locations not visited as part of the cycle. This subheap is non-empty. But we can not take any x_i as witness, since every location is reachable! This is a contradiction.

When speaking of modal operators, it is useful to speak of the 'current' heap (with respect to which any formula in separation logic is evaluated), the 'outer' heap (which is the heap with respect to which an enclosing formula is evaluated) and the 'inner' heap (which is the current heap while evaluating a subformula). This terminology is also useful when speaking about the separating connective $(\phi * \psi)$, where we would speak of the 'outer' heap with respect to which the entire formula is evaluated, and two 'inner' heaps corresponding to the evaluation of ϕ and ψ.

The point of the discussion above is that we can now understand more clearly what happens with the @-connective. Suppose now that ψ is pure, so it does not have any (free) references. Then we have that $(\phi @ \psi)$ and the formula

$$\blacksquare((\forall x, y.\, (x \hookrightarrow y) \leftrightarrow \psi(x, y)) \to \phi)$$

are equivalent. (We discuss this and related formulas in more detail below.) Clearly, this is a heap independent formula, due to the black box! However, when ψ is not pure, the formula $(\phi @ \psi)$ is not heap independent. In the @-connective, the crux is that the 'points to' symbol in ψ is relevant and its meaning depends on the 'outer' heap, whereas the 'points to' symbol in ϕ is intentionally captured by the @-connective where its denotation is described by ψ. The @-connective thus changes what is the 'current' heap when evaluating ϕ. This is similar to what the modal operator $\Box\phi$ does, in which also we have an 'inner' and 'outer' heap, but where the former is a subheap of the latter heap. In the @-connective the 'inner' heap is described by ψ, which may depend on the 'outer' heap when it is not a heap independent formula.

Existence of the empty heap, where nothing is allocated, is expressed by:

$$\blacklozenge(\forall x, y.\, (x \not\hookrightarrow y)).$$

But what about the opposite, the existence of a heap in which every location is allocated? Could the formula

$$\blacklozenge(\forall x \exists y.\, (x \hookrightarrow y))$$

be true? Or what about the existence of a heap in which every value is reachable? Could the formula

$$\blacklozenge(\forall y \exists x.\, (x \hookrightarrow y))$$

be true? No, in the standard interpretation of separation logic, based on the integers, these formulas are false because heaps are finitely-based partial functions!

Suppose we work with the standard integers $\mathbb{Z}$, and we have in our signature the usual arithmetical symbols. If we want to ensure we only deal with finite,

univalent heaps, then we should take the following formulas as axioms:

$$\blacksquare(\forall x, y, z.\, (x \hookrightarrow y) \wedge (x \hookrightarrow z) \to y = z)$$
$$\blacksquare(\exists x_0, x_1.\, \forall x, y.\, (x \hookrightarrow y) \to x_0 \leq x \leq x_1)$$

The first axiom expresses univalence. The second axiom expresses boundedness, that is, for every heap there is a bound on the domain, that is, there is a maximum and minimum location. Every finitely-based partial function satisfies these property (a finitely-based partial function can be seen as a finite list of location-value associations, and the maximum and minimum can be computed). Conversely, every heap that satisfies both axioms can be represented by a finitely-based partial function: there are only finitely many locations between the minimum and maximum location (due to boundedness) that can be assigned at most one value (due to univalence).

Note that in the standard interpretation of separation logic on the integers, we never treat the heap as a total map, where every location must have a value. It thus always remains a possibility for a location to be unallocated, i.e. the location x is unallocated in a situation whenever

$$\forall z(x \not\hookrightarrow z)$$

holds—which expresses that there is no value to which x points. In non-standard interpretations of separation logic, we do have the possibility of an infinite heap.

7 Conclusion

The proof system we introduce makes use of a new @-connective which allows to interpret the points-to relation in terms of a logical description. It bears some relation with hybrid logic [1] which features so-called nominals and satisfaction operators. Temporally, the nominals describe when is 'now', and the satisfaction operator allows to evaluate a formula with respect to a given nominal, thereby changing when is 'now'. As such, hybrid logic allows to express more than modal logic: an example is "At 6 o'clock, the church bells ring six times." This sentence is more time-specific than the usual modal operators for expressing 'always' or 'sometimes'. Comparing with the @-connective, we see that @ is even more general notion than what a satisfaction operator provides, since we introduce it as a connective between formulas. This means that formulas can now also take the place of the nominals in hybrid logic, and this allow us to describe a situation, that is, the 'current' heap, by means of a formula.

An important result is that our new proof system allows us to show many more equivalences than existing proof systems for separation logic. Thus we go beyond the capability of many existing tools for (automatic or interactive) reasoning about separation logic! It is quite surprising that none of the existing tools can verify some of our particular equivalences. We think this is due to the abstract description of the separating connectives in terms of *cancellative partially commutative monoids* (cf. separation algebras [2]). How to combine

this abstract description with a set-theoretical interpretation of the points-to relation is problematic. This seems to suggest we should start developing new kinds of tools for automatic or interactive reasoning about separation logic, or adjust the existing tools to be able to work around current limitations.

The presented proof system is sound and complete. This will be elaborated upon in following blog posts. In **part two** we study standard and non-standard interpretations of separation logic, and give the main argument of relative completeness of the novel proof system. Relative completeness is a completeness argument relative to an oracle. This approach is necessary since absolute completeness for standard separation logic is not possible due to failure of compactness. Other topics that we will discuss in this series of articles concern the impact on Reynolds' program logic [9], expressivity of separation logic and separation logic as an intermediate logic between first-order logic and second-order logic, and intuitionistic separation logic.

How to cite?

You can cite this article using BibTeX:

```
@misc{drheap2024sl1,
    title={A sound and complete proof system for separation logic (part 1)},
    author={Hiep, {Hans-Dieter} A. and de Boer, Frank S.},
    howpublished={{\bf dr.\,heap}},
    month={June},
    year={2024},
    DOI={10.59350/2gkd1-c0k49}
}
```

Or by including the following attribution:

> Hans-Dieter A. Hiep and Frank S. de Boer. A sound and complete proof system for separation logic (part 1). **dr. heap**, June 2024. DOI: 10.59350/2gkd1-c0k49.

References

[1] Torben Braüner. *Hybrid Logic and its Proof-Theory*. Springer, 2010. doi:10.1007/978-94-007-0002-4.

[2] Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. Local action and abstract separation logic. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 366–378. IEEE, 2007. doi:10.1109/LICS.2007.30.

[3] Stéphane Demri, Étienne Lozes, and Alessio Mansutti. A complete axiomatisation for quantifier-free separation logic. *Log. Methods Comput. Sci.*, 17(3), 2021. doi:10.46298/lmcs-17(3:17)2021.

[4] David Herman and Mitchell Wand. A theory of hygienic macros. In *Programming Languages and Systems: 17th European Symposium on Programming*, pages 48–62. Springer, 2008. doi:10.1007/978-3-540-78739-6_4.

[5] Hans-Dieter A. Hiep. *New Foundations for Separation Logic*. PhD thesis, Leiden University, 2024. URL: https://hdl.handle.net/1887/3754463.

[6] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming*, 28, 2018. doi:10.1017/S0956796818000151.

[7] Willard Van Orman Quine. *Word and object*. MIT press, 2013. doi:10.7551/mitpress/9636.001.0001.

[8] Andrew Reynolds, Radu Iosif, Cristina Serban, and Tim King. A decision procedure for separation logic in SMT. In *International Symposium on Automated Technology for Verification and Analysis*, pages 244–261. Springer, 2016. doi:10.1007/978-3-319-46520-3_16.

[9] J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55–74, 2002. doi:10.1109/LICS.2002.1029817.

