New Foundations for Separation Logic

Hiep, H.A.

Chapter 2

Model theory of separation logic

In this chapter, we introduce the syntax and semantics of separation logic. We shall present the standard interpretation of separation logic, but also investigate a new interpretation of separation logic and the relations between the different interpretations of separation logic, and between separation logic and second-order logic. The first result is the inadequacy of the standard of separation logic, by showing it is non-compact. We then introduce a new interpretation, the full interpretation of separation logic based on the possibility of infinite heaps, and show it is inadequate too. We investigate the sufficient and necessary conditions for an embedding of the standard interpretation into the full interpretation, and we introduce relational separation logic to compare separation logic to second-order logic. An interesting result is that the full interpretation of separation logic is close to the standard interpretation of second-order logic, in the sense that the expressivity of a binding operator is sufficient for the two logics to coincide. As such, this chapter is a model theoretic investigation of separation logic.

Informally, the purpose of separation logic is to formalize and allow reasoning about the notion of spatial separation. This intuition can best be elucidated by the following examples, in which we see the different meaning in natural language of the conjunction of two facts:

- "I know the moon orbits the earth."
- "I have a Euro in one of my pockets."

Notice how the meaning of conjunction differs in the following sentences:

- "I know the moon orbits the earth, and I know the moon orbits the earth."
- "I have a Euro in one of my pockets, and I have a Euro in one of my pockets."

In the first sentence, describing the knowledge that the moon orbits the earth twice does not change the way we could interpret the overall sentence. Once you know

something, it does not matter how often you think of it: we have the classical propositional law that states that $A \wedge A$ is equivalent to A for any proposition A. This works for any 'pure' proposition.

However, in the second sentence there is a difference between our classical intuition, and our spatial intuition. Classically, we are able to substitute any sentence for the proposition A, and hence the propositional law should also apply here. But this does not entirely capture our spatial intuition. The second sentence is not precise: is the Euro in the same pocket, or in different pockets? Phrased differently, is the expression 'one of my pockets' in both conjuncts referring to the same pocket or to two different pockets? Classically, to be precise, one would have to explicitly describe this situation: "I have a Euro in one of my pockets, and I have a Euro in another of my pockets."

Notice how the explicit difference in location, which we classically need to describe to be precise, can also be resolved differently: by changing the way we interpret the conjunction—no longer classically, but spatially: "I have a Euro in one of my pockets, and separately, I have a Euro in one of my pockets." Surely the two pockets must now be different pockets, because how could otherwise one fact be separate from the other fact?

We introduce the following symbols to abbreviate our intuition. We write $(x \hookrightarrow y)$ to express that the location x has the value y. If we interpret x as being the location of one of my pockets and y as the value of one Euro, then $(x \hookrightarrow y)$ expresses that "I have a Euro in one of my pockets". Further, we introduce the spatial conjunction, or separating conjunction, by using $*$ as a connective. Symbolically, we can evaluate the following formulas:

1. $\exists x(x \hookrightarrow y)$,
2. $(\exists x(x \hookrightarrow y)) \wedge (\exists x(x \hookrightarrow y))$,
3. $(\exists x(x \hookrightarrow y)) * (\exists x(x \hookrightarrow y))$,
4. $\exists x((x \hookrightarrow y) \wedge \exists z(z \neq x \wedge (z \hookrightarrow y)))$.

Notice how the first two formulas are equivalent (due to the classical law that $A \wedge A$ is equivalent to A). However, the second and third formulas are not equivalent. The second formula expresses that at least one pocket has a Euro in it. The third formula expresses that at least two (different) pockets have a Euro in them. Intuitively, the third and fourth formula are again equivalent, where in the fourth formula we classically express that the second pocket is different from the first pocket.

If we scale up our argument to more and more pockets (for example, imagine the many pockets of a handyman), we observe the following pairs of equivalences:

- $(\exists x(x \hookrightarrow y)) * (\exists x(x \hookrightarrow y)) * (\exists x(x \hookrightarrow y))$,
- $\exists x((x \hookrightarrow y) \wedge \exists z(z \neq x \wedge (z \hookrightarrow y) \wedge \exists w(w \neq x \wedge w \neq z \wedge (w \hookrightarrow y))))$.

and

- $(\exists x(x \hookrightarrow y)) * (\exists x(x \hookrightarrow y)) * (\exists x(x \hookrightarrow y)) * (\exists x(x \hookrightarrow y))$,
- $\exists x((x \hookrightarrow y) \wedge \exists z(z \neq x \wedge (z \hookrightarrow y) \wedge \exists w(w \neq x \wedge w \neq z \wedge (w \hookrightarrow y) \wedge \exists v(v \neq x \wedge v \neq z \wedge v \neq w \wedge (v \hookrightarrow y)))))$.

We see how, classically, we need to describe more and more facts that state that the locations are all pairwise different, whereas with our spatial intuition we simply declare these locations to be separate by the separating conjunction connective. The separation is either described 'bottom up' using explicit equational facts, or 'top down' using separating connectives.

Especially in the setting of programming with pointers, this spatial intuition is natural to reason about. Often, data structures are laid out in separate parts of the memory, and describing explicitly that these parts of the memory are separate quickly grows in complexity. Consider, for example, the circular singly-linked list of Figure 2.1. Whenever we informally reason about the data structure of a linked list, we mentally model the memory state by means of a picture in which each box represents some storage space in memory, and pointers to boxes represent the address value of the location of that memory space. Already by choosing such a picturesque model, we have the graphical intuition that locations in space are separate: drawing two boxes on paper means the two boxes have to be separate. However, classically, one would have to explicitly describe that to be precise.

Consider running through the execution of a program that manipulates the memory states of a linked list:

(a) In the initial state we have one item which is linked back to itself. This is the so-called head of the list. In Figure 2.1 we see this situation, where x is a variable that points to the location of the box, and the value of the box points to itself. Symbolically, we would describe this by stating:

$$(x \hookrightarrow x) \wedge \forall z((z \hookrightarrow -) \rightarrow z = x)$$

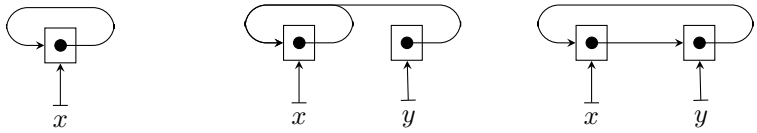
where $(z \hookrightarrow -)$ means $\exists y(z \hookrightarrow y)$. This describes that the value of x represents a location, which is allocated since we see it points to a box, and the value of the location is the address of itself, since we see a pointer from inside the box to the edge of the box itself. The second conjunct expresses that there are no other locations. We can abbreviate this formula, by simply writing:

$$(x \mapsto x)$$

where the symbol $\mapsto$ indicates that the location is the sole location that is allocated.

(b) Next, we allocate new space. So y now points to a location that was previously unallocated, but now it is allocated since it takes up space as a box. We obtain the 'paperclip' state, in which the box to which x points still points to itself, but also the box to which y points points to the box to which x points. Symbolically we have

$$(x \mapsto x) * (y \mapsto x)$$



(a) Initial state (b) After creation (c) Relinking

Figure 2.1: Three different memory states of a circular singly-linked list.

since we can imagine to split the memory in two parts: the box on the left, and the new box on the right. The box on the left still points to itself. The box on the right points to the box on the left.

(c) Finally, we relink the previous box to the new box: the box to which x points itself now points to the box to which y points. Symbolically, we express this situation as:

$$(x \mapsto y) * (y \mapsto x)$$

and this description is quite precise. We know, from the picture, that x and y must point to *different* boxes, because we have drawn these boxes apart from each other. Notice that the two components each point to each other. The components, when viewed isolated from each other, have *dangling pointers*. However, by putting the components in separating conjunction, these former dangling pointers now resolve to the proper location, being the box in each component.

This simple example convincingly shows that pointer structures can be described, component-wise, using the connective of separating conjunction. Separating conjunction directly captures the intuition that the locations to which one refers to are separated among the components of the conjunction. Further examples of cyclic data structures and containers can be given: doubly-linked lists, tree structures with pointers from parents to children but also back links from children to parents. In fact, our intuition of these pointer structures quite naturally transfers to the memory states of object-oriented programs, in which the precise identity of objects is abstracted away, where different pointers coming out of a box represent the different fields of an object.

Another important aspect of spatial intuition is hypothetical space: the question what happens if one would allocate locations according to a description, where those locations are previously not allocated. To describe hypothetical situations we introduce the connective $-\!\!*$ that is called separating implication, or the magic wand. Consider Figure 2.2 in which there are two situations depicted. In the situation on right we are dealing with a hypothetical extension of the situation depicted on the left, as represented by a dashed arrow. We can describe these situations as follows:

(a) This situation is described precisely by the formula

$$(x \mapsto y) \wedge (y \not\mapsto -).$$

(a) Original state (b) Original state and extended state

Figure 2.2: Hypothetical extension of locations.

The location pointed to by y is not allocated, as depicted by a crossed-out box. Hence both the box pointed to by x, and to which y points, are dangling pointers: the location to which is pointed is not allocated.

(b) Now, we could imagine a hypothetical extension of the state, in which the box pointed to by y actually exists and points back to x. This hypothetical situation can be described by the formula

$$(x \mapsto y) \wedge (y \not\mapsto -) \wedge ((y \mapsto x) \mathrel{-\!\!*} ((x \mapsto y) * (y \mapsto x))).$$

The connective $\mathrel{-\!\!*}$ describes on the right of the connective what holds of the resulting, hypothetical memory state after the current memory state is extended by any separate part of the memory for which the left of the connective holds.

In this chapter we shall formally introduce *separation logic*. In separation logic we aim to formally capture our intuition as given above. We introduce the language in which formulas are described, their interpretation by means of structures. In this chapter we focus on the model theoretic development of the semantics of separation logic. Next chapter we develop the proof theory for reasoning about formulas of separation logic.

Technically, we restrict ourselves to classical first-order separation logic. By classical we mean that we interpret the formulas of first-order logic embedded in separation logic classically. Although the language of separation logic extends the language of first-order logic, not all classical laws such as the law of excluded middle hold for all classical separation logic formulas. The embedding of first-order logic in separation logic is also called the 'pure' part of separation logic; it is pure since it does not depend on our spatial intuition. Further, we restrict to first-order logic and do not develop higher-order separation logic, to keep the presentation light. In principle, nothing restrains us from allowing higher-order variables and higher-order quantification in separation logic too. Furthermore, just like in Chapter A that introduced first-order logic, we keep terms orthogonal to our discussion. Although terms can be easily added to the syntax, and we can interpret constant symbols and function symbols specially as individuals and functions, it is not necessary to do so—we do not miss out any expressivity of the logic, but we get simpler definitions by leaving them out.

Throughout this chapter, we will introduce different classes of separation logic formulas. An overview of these classes of formulas is given in Figure 2.3.

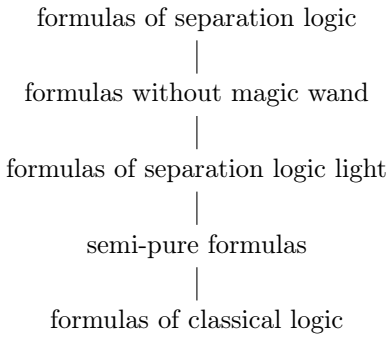


Figure 2.3: Overview of the classes of formulas of separation logic. Formulas are included from bottom to top.

2.1 Syntax of separation logic

The language of separation logic imports many of the concepts already present in classical logic, such as variables and signatures. Formulas, or synonymously assertions, of separation logic extend the formulas of classical logic in two aspects: first, we add a new primitive formula, called *points-to*, denoted by $\hookrightarrow$, and, secondly, we add two new connectives, called *separating conjunction* and *separating implication*, denoted by $*$ and $-\!\!*$, respectively. After extending formulas, some concepts imported from classical logic require some adjustments to the setting of separation logic. We also introduce new concepts that are were not yet present in classical logic.

In our presentation of the assertions of separation logic, we shall not introduce terms at first. This makes a comparison with our presentation of classical logic easier, and also makes the technical development easier to follow. However, by taking this approach, we do not lose any expressive power: later in this section we add terms in an orthogonal way, regardless of whether the assertion language of classical logic or separation logic is used.

Based on a given signature (see Definition A.1.2), we construct the formulas of separation logic. Although we follow [125] in the definition of the syntax (and later also the standard semantics) of the assertion language of separation logic, we now work in this more general setting with signatures and we use a different atomic 'weak points to' formula. For the remainder of this section, we fix a first-order signature Σ that consists of constant symbols, similar to what we did for classical logic. It should be clear from context whether one speaks of formulas of separation logic or formulas from classical logic. In this chapter, 'formula' refers to 'formula of separation logic' unless explicitly mentioned otherwise.

Definition 2.1.1 (Formulas)**.** A *formula* is constructed inductively as follows:

1. $\bot$ is a formula,
2. $(x \doteq y)$ is a formula if x and y are individual variables,

3. $(x \hookrightarrow y)$ is a formula if x and y are individual variables,
4. $C(x_1, \ldots, x_n)$ is a formula if C is a constant symbol of arity n and $x_1, \ldots, x_n$ are individual variables,
5. $(\phi \to \psi)$ is a formula if ϕ and ψ are formulas,
6. $(\forall x \phi)$ is a formula if ϕ is a formula and x an individual variable,
7. $(\phi * \psi)$ is a formula if ϕ and ψ are formulas,
8. $(\phi \mathrel{-\!\!*} \psi)$ is a formula if ϕ and ψ are formulas.

All formulas are constructed by one of these eight clauses. Alternatively, we can define formulas by the following abstract grammar:

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

Note that in separation logic, we prefer using **false** and **true** instead of $\bot$ and $\top$, where **false** abbreviates $\bot$, and **true** abbreviates $(\bot \to \bot)$.

The first four clauses construct primitive formulas, the last four clauses construct complex formulas. Formulas can still be regarded as finite sequences of symbols, but we consider the newly introduced symbols $\hookrightarrow$, $*$, $\mathrel{-\!\!*}$ to be *separation symbols* disjoint from the earlier *logical symbols* and *non-logical symbols*. It is easy to verify that the set of formulas, as defined above, is recursive.

We speak of formulas in the usual way, except for the new clauses. The primitive formula $(x \hookrightarrow y)$ is called *points-to* (as in 'x points 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.

Again, when proving meta-properties of formulas, we proceed by induction on the *complexity* of formulas. There are different obvious measures of complexity: the *height of a formula*, by viewing the formula as a parse tree and taking the height of that tree, or the *length of a formula*, by viewing a formula as a sequence of symbols and taking the length of that sequence.

In a certain sense, Definition 2.1.1 includes all first-order formulas of classical logic (cf. Definition A.1.4). Regarding formulas as sequences of symbols, if in that sequence there is no separation symbols present then the formula must also be a (first-order) formula of classical logic. The latter formulas are also called *pure* formulas or *heap-independent* formulas. If a formula, seen as a sequence of symbols, contains a separation symbol we call it *impure* or *heap-dependent*. If a formula is impure but does not contain separating connectives (i.e. contains primitive points-to constructs) then we call it *semi-pure*.

As such, we may use the usual classical abbreviations, given in Definition A.1.5, too in separation logic. This means we have access to (logical) disjunction, conjunction, bi-implication and existential quantification. Further, we also use these

abbreviations in the case where they are used to compose heap-dependent formulas. Specifically, we may use separating connectives nested under logical connectives or quantifiers.

We further have the separation symbols $\not\hookrightarrow$, **emp**, $\mapsto$ and we use the symbol $-$ as a placeholder, by introducing the following abbreviations:

$$\begin{aligned}
(x \not\hookrightarrow y) &\text{ abbreviates } (\neg(x \hookrightarrow y)) \\
(x \hookrightarrow -) &\text{ abbreviates } (\exists z(x \hookrightarrow z)) \\
(x \not\hookrightarrow -) &\text{ abbreviates } (\neg(x \hookrightarrow -)) \\
\mathbf{emp} &\text{ abbreviates } (\forall x(x \not\hookrightarrow -)) \\
(x \mapsto y) &\text{ abbreviates } ((x \hookrightarrow y) \wedge (\forall z, w((z \hookrightarrow w) \to x \doteq z))) \\
(x \mapsto -) &\text{ abbreviates } (\exists z(x \mapsto z))
\end{aligned}$$

where z is a fresh individual variable (i.e. not the same as x). We may speak of 'locations' being on the left-hand side of either $\hookrightarrow$ or $\mapsto$, and 'values' being on the right-hand side of either $\hookrightarrow$ or $\mapsto$. We may speak about these formulas in the following way:

- for $(x \hookrightarrow y)$ we say 'x points to y' or 'location x has value y',
- for $(x \hookrightarrow -)$ we say '(at least) x is allocated',
- for $(x \mapsto y)$ we say 'strictly x points to y' or we may say 'x points to y and only x is allocated',
- for $(x \mapsto -)$ we say 'x is solely allocated' or 'x is allocated alone',
- for **emp** we say 'nothing is allocated'.

When speaking of the negated forms, some care is needed:

- for $(x \not\hookrightarrow y)$ we say 'x does not point to y' or 'x has not the value y', but note that does not necessarily mean x is not allocated,
- for $(\forall y(x \not\hookrightarrow y))$ we say 'x has no value' or 'x is not allocated',
- equivalently, for $(x \not\hookrightarrow -)$ we say 'x is not allocated' (but that does not say anything about other allocations).

Remark 2.1.2. In some texts on separation logic, the symbols **emp** and $\mapsto$ are taken as primitive formula (instead of $\hookrightarrow$). In this case we recover the same language as we have here by taking $(x \hookrightarrow y)$ as an abbreviation of $(\mathbf{true} * (x \mapsto y))$. In other texts the 'weak' points to is taken as primitive (as in [187] and [70]). The reason we take $\hookrightarrow$ as primitive is to avoid the use of separating connectives in expressing our abbreviations, while still being able to express that an element is not allocated: all abbreviations are semi-pure.

The same conventions for reducing parentheses are employed, with the following two additions to precedence: separating conjunction precedes logical conjunction (and so also disjunction and logical implication), and separating implication precedes logical implication (and so also bi-implication). All separating connectives also associate to the right. For example, $P(x) * Q(x) \wedge P(x)$ disambiguates to $((P(x) * Q(x)) \wedge P(x))$, and $P(x) \to Q(x) \mathrel{-\!\!*} P(x) \to Q(x)$ disambiguates to $(P(x) \to ((Q(x) \mathrel{-\!\!*} P(x)) \to Q(x)))$. However, we shall try to use parentheses, even if not necessary by these disambiguation rules, to present formulas as clearly as possible also for readers less familiar with separation logic.

The concept of variable occurrences in formulas can be imported in a straightforward manner. Formulas in separation logic can also be viewed as a parse tree, in which variables occurs at leaves. We also have the sets $FV(\phi)$, and $BV(\phi)$, that denote the variables that occur free in ϕ, and the variables that occur bound in ϕ, respectively. Revisiting Definition A.1.6 (Free and bound variables), we need to add the following clauses:

- $FV(x \hookrightarrow y) = \{x, y\}$ and $BV(x \hookrightarrow y) = \emptyset$,
- $FV(\phi * \psi) = FV(\phi) \cup FV(\psi)$ and $BV(\phi * \psi) = BV(\phi) \cup BV(\psi)$,
- $FV(\phi \mathrel{-\!\!*} \psi) = FV(\phi) \cup FV(\psi)$ and $BV(\phi \mathrel{-\!\!*} \psi) = BV(\phi) \cup BV(\psi)$.

As such, we also import the following concepts: a formula without free variables is a *sentence*, a *context* is a list of formulas, and a *theory* is a set of sentences.

Similarly, we have also the application $\pi(\phi)$ of a renaming π to a formula ϕ. Revisiting Definition A.1.9 (Variable renaming), we need to add the following:

- $\pi(x \doteq y) = (\pi(x) \doteq \pi(y))$,
- $\pi(\phi * \psi) = (\pi(\phi) * \pi(\psi))$,
- $\pi(\phi \mathrel{-\!\!*} \psi) = (\pi(\phi) \mathrel{-\!\!*} \pi(\psi))$.

Also (capture-avoiding) substitution of variables for variables works in the same way as in classical logic. We can also add terms to separation logic in the same manner as is done in Section A.5.

2.2 Standard semantics

The standard semantics of separation logic formula is given in the style of Tarski, extending the semantics of classical logic. There are two important aspects to consider before we define the satisfaction relation formally.

The first aspect is that we employ the same structures that we used for giving semantics to classical logic: this ensures that the semantics of the heap-independent formulas of separation logic coincides with their classical semantics. Further, terms are evaluated without depending on the heap. So the semantics of terms in separation logic completely coincides with the semantics of terms in classical logic.

The second aspect is the context in which we define the satisfaction relation. In separation logic, we employ another concept, next to structures and valuations, called the *heap*. A heap is represented by a partial function. In the standard semantics we furthermore restrict ourselves to *finitely-based* partial functions, meaning that only finitely many locations are assigned a value by the partial function representing the heap.

Let $\mathfrak{A} = (A, \mathcal{I})$ be a structure (see Definition A.2.3). A *finite heap* of $\mathfrak{A}$ is a finitely-based partial function from A to A. A heap thus assigns to finitely many elements of the domain of the structure a *value*, which is also an element of the domain of the structure. Let h be a heap. By $\mathrm{dom}(h)$ we denote the *domain* of the heap h, that is, the set of all elements of A on which h is defined. If a is an element for which h is undefined we also write $h(a) = \bot$ (where we implicitly know $\bot \notin A$ since $\bot$ is some dummy element), and if a is an element for which h is defined we write $h(a) = a'$ for some value $a' \in A$.

We may also speak of *locations* to mean elements that are (possibly) in the domain of a heap. A finite heap thus assigns finitely many locations to values. Note that speaking of just a domain may be unclear: is one speaking of the *domain of a structure*, or the *domain of a heap*? The latter, however, is a (finite) subset of the former.

We define three operations on finite heaps. There is the *empty heap*, denoted by ϵ. The empty heap is undefined on every element of the domain, that is, $\mathrm{dom}(\epsilon) = \emptyset$. Clearly the domain of the empty heap is finite. Given two elements a, a' of A. By $h[a := a']$ we denote the heap obtained after applying a *heap update* operation that sets location a to the element a'. Formally,

$$h[a := a'](e) = \begin{cases} a' & \text{if } a = e \\ h(e) & \text{if } a \neq e \text{ and } h(e) \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

where e ranges over elements of the domain A of our structure. We now thus have $\mathrm{dom}(h[a := a']) = \mathrm{dom}(h) \cup \{a\}$, and so the domain remains finite. By $h[a := \bot]$ we denote the heap obtained after applying a *heap clear* operation that clears location a from the domain. Formally,

$$h[a := \bot](e) = \begin{cases} \text{undefined} & \text{if } a = e \\ h(e) & \text{if } a \neq e \text{ and } h(e) \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

and thus $\mathrm{dom}(h[a := \bot]) = \mathrm{dom}(h) \setminus \{a\}$, and also here the domain remains finite.

We have the property of *heap extensionality*: given two finite heaps h and g, then $h = g$ if and only if $\mathrm{dom}(h) = \mathrm{dom}(g)$ and $h(a) = g(a)$ for every $a \in \mathrm{dom}(h)$.

Intuitively, we could split and merge heaps. A finite heap that has more than one element in its domain can be partitioned into two finite heaps, by selecting for each element in the domain to what partition it should belong after the split. Similarly, given two finite heaps that have a disjoint domain, we can form a new

finite heap by merging the two. After splitting or merging, the values assigned to locations remain the same. To formalize these intuitions, we introduce the concept of a heap partitioning.

Let h_1 and h_2 be finite heaps with disjoint domains, $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \emptyset$. We sometimes write $h_1 \perp h_2$ to abbreviate $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \emptyset$. Now $h_1 \uplus h_2$ denotes a finite heap that can be split into two parts, h_1 and h_2, so has as domain the union of the underlying domains, $\mathrm{dom}(h_1 \uplus h_2) = \mathrm{dom}(h_1) \cup \mathrm{dom}(h_2)$. Every location in the resulting heap has the value of the corresponding underlying heap, $(h_1 \uplus h_2)(e) = h_1(e)$ if $e \in \mathrm{dom}(h_1)$ and $(h_1 \uplus h_2)(e) = h_2(e)$ if $e \in \mathrm{dom}(h_2)$. Locations outside of the domain remain undefined, $(h_1 \uplus h_2)(e) = \perp$ if $e \notin \mathrm{dom}(h_1)$ and $e \notin \mathrm{dom}(h_2)$. Thus one can think of $h_1 \uplus h_2$ as a merged heap. It does not exists when h_1 and h_2 both assign a value to the same location, even when both h_1 and h_2 assign the same value to shared locations. Although the latter makes sense when merging heaps, its fails our intuition in the other direction, when splitting a heap in two parts. Thus, we write $h \equiv h_1 \uplus h_2$ to mean $h_1 \perp h_2$, that h_1 and h_2 have disjoint domains—and so the finite heap $h_1 \uplus h_2$ exists, and $h = h_1 \uplus h_2$, and we say that there is a heap partitioning.

We are now able to give the formal definition of the satisfaction relation. Our definition is an extension of Definition A.2.5 in two ways: we additionally consider a finite heap h, and we have new clauses corresponding to points-to, separating conjunction, and separating implication.

Definition 2.2.1 (Satisfaction relation)**.** Given a structure $\mathfrak{A} = (A, \mathcal{I})$, a valuation ρ of $\mathfrak{A}$, a finite heap h of $\mathfrak{A}$, and a separation logic formula ϕ. The satisfaction relation $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi$ is defined inductively on the structure of ϕ:

- $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \perp$ never holds,
- $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} (x \doteq y)$ iff $\rho(x) = \rho(y)$,
- $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} (x \hookrightarrow y)$ iff $h(\rho(x))$ is defined and $h(\rho(x)) = \rho(y)$,
- $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} C(x_1, \ldots, x_n)$ iff $(\rho(x_1), \ldots, \rho(x_n)) \in C^{\mathcal{I}}$,
- $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi \to \psi$ iff $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi$ implies $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \psi$,
- $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \forall x \phi$ iff $\mathfrak{A}, h, \rho[x := a] \models^{\mathbf{SSL}} \phi$ for every $a \in A$,
- $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi * \psi$ iff $\mathfrak{A}, h_1, \rho \models^{\mathbf{SSL}} \phi$ and $\mathfrak{A}, h_2, \rho \models^{\mathbf{SSL}} \psi$ for some h_1, h_2 such that $h \equiv h_1 \uplus h_2$,
- $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi \mathrel{-\!\!*} \psi$ iff $\mathfrak{A}, h', \rho \models^{\mathbf{SSL}} \phi$ implies $\mathfrak{A}, h'', \rho \models^{\mathbf{SSL}} \psi$ for every h', h'' such that $h'' \equiv h \uplus h'$.

The superscript **SSL** stands for Standard Separation Logic. In the third clause it is superfluous to state that $h(\rho(x))$ is defined, since we know that $\rho(y)$ cannot be the dummy element $\perp$ and hence $\rho(x) \in \mathrm{dom}(h)$. Further, since we restrict ourselves to first-order signatures, the valuation ρ of $\mathfrak{A}$ only assigns individual

variables a value. We shall leave out the discussion how the satisfaction relation is defined for empty structures, since it is similar to classical logic.

Based on this definition it is now also possible to give semantics of abbreviations, similar to what we did in the case of classical logic. Also similar to classical logic, we have the coincidence condition and invariance under renaming. Both propositions are with respect to a fixed heap.

Proposition 2.2.2 (Coincidence condition)**.** *Given that* $\rho[FV(\phi)] = \rho'[FV(\phi)]$*, it follows that* $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi$ *if and only if* $\mathfrak{A}, h, \rho' \models^{\mathbf{SSL}} \phi$*.*

Proposition 2.2.3 (Invariance under renaming)**.** *Given a renaming* π *such that all free variables of* ϕ *stay the same, i.e.* $\pi(v) = v$ *for all* $v \in FV(\phi)$*. It follows that* $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi$ *if and only if* $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \pi(\phi)$*.*

Sometimes, it is more convenient to work with the set of heaps and valuations by which a formula is satisfied given a particular structure.

Definition 2.2.4 (Denotation)**.** The *denotation* of a formula $\mathfrak{A}[\![\phi]\!]^{\mathbf{SSL}}$ is defined:

$$\mathfrak{A}[\![\phi]\!]^{\mathbf{SSL}} = \{(h, \rho) \mid \mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi\}.$$

Similar as before, we may drop **SSL** if clear from context. We write $\phi \equiv_{\mathfrak{A}} \psi$ for $\mathfrak{A}[\![\phi]\!] = \mathfrak{A}[\![\psi]\!]$, and say that ϕ and ψ are equivalent.

Note that the we can also add terms to separation logic, completely analogous to what we did in Section A.5. An important result also holds for separation logic:

Lemma 2.2.5 (Substitution lemma)**.**

$$\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi[x := t] \text{ if and only if } \mathfrak{A}, h, \rho[x := \rho(t)] \models^{\mathbf{SSL}} \phi.$$

We write $\mathfrak{A}, h \models^{\mathbf{SSL}} \phi$ to mean $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi$ for all valuations ρ, and we write $\mathfrak{A} \models^{\mathbf{SSL}} \phi$ to mean $\mathfrak{A}, h \models^{\mathbf{SSL}} \phi$ for all finite heaps h. Given a sentence that is satisfied, using the coincidence condition we can obtain that it is also satisfied by the same structure but with any other valuation: the valuation has no influence on whether a sentence is satisfied by the structure, but the heap does have such influence. So if ϕ is a sentence, $\mathfrak{A}, h \models^{\mathbf{SSL}} \phi$ if and only if $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi$ for some valuation ρ.

Given a sentence ϕ, we write $\models^{\mathbf{SSL}} \phi$ to mean that $\mathfrak{A} \models^{\mathbf{SSL}} \phi$ for all structures $\mathfrak{A}$, and we then say that ϕ is *valid*. Valid sentences in separation logic thus are properties that hold for all finite heaps.

Given a theory, i.e. a set of sentences Γ, we write $\mathfrak{A}, h \models^{\mathbf{SSL}} \Gamma$ to mean that all sentences in Γ are satisfied by $\mathfrak{A}$ and finite heap h, that is, $\mathfrak{A}, h \models^{\mathbf{SSL}} \phi$ for all $\phi \in \Gamma$. We may also speak of 'Γ is satisfied by $\mathfrak{A}$ and h'. A theory Γ is *satisfiable* if there exists a structure $\mathfrak{A}$ and heap h such that $\mathfrak{A}, h \models^{\mathbf{SSL}} \Gamma$. A theory Γ is *finitely satisfiable* if every finite subset of Γ is satisfiable. Note that the finite heap is considered existentially when speaking of (finite) satisfiability in separation logic.

We write $\Gamma \models^{\mathbf{SSL}} \phi$ to mean $\mathfrak{A}, h \models^{\mathbf{SSL}} \phi$ for all structures $\mathfrak{A}$ and finite heaps h such that $\mathfrak{A}, h \models^{\mathbf{SSL}} \Gamma$, and say that ϕ is a *semantic consequence* of Γ.

By $Th^{\mathbf{SSL}}(\mathfrak{A})$ we mean the set of all sentences ϕ such that $\mathfrak{A} \models^{\mathbf{SSL}} \phi$, and we speak of the *separation logic theory of* $\mathfrak{A}$. Note that we have $Th_1^{\mathbf{CL}}(\mathfrak{A}) \subseteq Th^{\mathbf{SSL}}(\mathfrak{A})$, that is, the first-order theory of $\mathfrak{A}$ is included in its separation logic theory. Further, a separation logic theory contains only sentences that are universal in the heap, i.e. sentences that hold for every finite heap.

Proposition 2.2.6. *For any sentence ϕ we have $\mathfrak{A}, h \models^{\mathbf{SSL}} \phi$ or $\mathfrak{A}, h \models^{\mathbf{SSL}} \neg\phi$.*

However, contrasting to the first-order theory of a structure, which is necessarily complete, we do not have that the separation theory of a structure is necessarily complete. To see why, consider the counter-example used in the following proof.

Proposition 2.2.7. *$Th^{\mathbf{SSL}}(\mathfrak{A})$ is complete if and only if the domain of $\mathfrak{A}$ is empty.*

Proof. Assume the domain of $\mathfrak{A}$ is not empty. It is sufficient to show there is a sentence ϕ such that there is a heap h_1 such that $\mathfrak{A}, h_1 \models^{\mathbf{SSL}} \phi$, and there is a heap h_2 such that $\mathfrak{A}, h_2 \models^{\mathbf{SSL}} \neg\phi$. Take ϕ to be $\mathbf{emp}$. Now h_1 can be simply ϵ, the empty heap. And h_2 is any non-empty heap (which exists since the domain of our structure is non-empty).

Assume $Th^{\mathbf{SSL}}(\mathfrak{A})$ is not complete. Thus there is a sentence ϕ such that it is not the case that $\mathfrak{A} \models^{\mathbf{SSL}} \phi$ or $\mathfrak{A} \models^{\mathbf{SSL}} \neg\phi$. So there exists h_1, h_2 such that $\mathfrak{A}, h_1 \models^{\mathbf{SSL}} \neg\phi$ and $\mathfrak{A}, h_2 \models^{\mathbf{SSL}} \phi$. Now h_1 and h_2 are not equal due to Proposition 2.2.6. If, however, $\mathfrak{A}$ is empty then there is only one heap and h_1 and h_2 thus must be equal. So $\mathfrak{A}$ must be non-empty. □

Another difference is that the standard semantics of separation logic is not compact, in contrast to classical logic (see Theorem A.2.10). So see why, consider the following counter-example: every finite subset of an infinite set of sentences expressing that the domain of the heap contains at least so many elements is satisfiable, but clearly no finite heap satisfies the entire set.

Lemma 2.2.8 (Non-compactness standard semantics)**.** *It is not the case that Γ is finitely satisfiable implies that Γ is satisfiable.*

Proof. Let $x_0, x_1, x_2, \ldots$ be individual variables. We construct a set of sentences Γ which is finitely satisfiable, but not satisfiable:

$$\Gamma = \{\phi_n \mid n \in \mathbb{N}\}$$

where ϕ_n expresses that there are at least $n+1$ allocated and distinct elements:

$$\phi_n = \exists x_0, \ldots, x_n.\ (\bigwedge_{0 \leq i \leq n} (x_i \hookrightarrow -)) \wedge \bigwedge_{0 \leq i < j \leq n} x_i \neq x_j.$$

Clearly, Γ is infinite. For any finite subset of Γ, there must exist a maximum m such that $\phi_m \in \Gamma$. Then we take a structure with as domain $\mathbb{N}$ and a finite heap in which the locations $0, \ldots, m$ are allocated (their value does not matter). Every formula ϕ_i for $0 \leq i \leq m$ is satisfied. This construction works for every finite subset of Γ, so Γ is finitely satisfiable. However, Γ is not satisfiable, since for every $\phi_i \in \Gamma$ there always exists $\phi_j \in \Gamma$ with $j > i$, and so we cannot construct a finite heap that satisfies all sentences. □

Above it is also possible to give ϕ_n using separating conjunctions, namely by

$$\phi_n = \exists x_0, \ldots, x_n.\ (x_0 \hookrightarrow -) * \ldots * (x_n \hookrightarrow -).$$

From the semantics of separating conjunction, it follows that $x_i \neq x_j$ for every $0 \leq i < j \leq n$: if it were the case that $x_i = x_j$ for $i \neq j$ then one cannot split the heap into disjoint parts that satisfies all components of the conjunction.

The failure of compactness has important ramifications to the design of a proof system for separation logic.

Corollary 2.2.9. *There is no sound, complete, finitary proof system for* **SSL**.

Proof. Suppose there would be a finitary proof system **SSL** that allows us to define what it means that a sentence ϕ is a syntactic consequence of a theory Γ, denoted $\Gamma \vdash^{\mathbf{SSL}} \phi$, and suppose that it is complete with respect to the standard semantics:

$$\Gamma \models^{\mathbf{SSL}} \phi \text{ implies } \Gamma \vdash^{\mathbf{SSL}} \phi.$$

Now we have that compactness follows from it: Γ is finitely satisfiable if and only if Γ is satisfiable. To see why, it is sufficient to show that if Γ is not satisfiable then Γ is not finitely satisfiable. Suppose some theory Γ is not satisfiable, then $\Gamma \models^{\mathbf{SSL}} \bot$ is the case. By completeness, we then have $\Gamma \vdash^{\mathbf{SSL}} \bot$. By the finitary nature of our proof system, there must only be finitely many sentences in Γ on which the deduction is based. Let Γ_0 denote that finite subset of Γ such that the same deduction can be used to witness $\Gamma_0 \vdash^{\mathbf{SSL}} \bot$. Thus Γ is not finitely satisfiable. Hence, the existence of a complete finitary proof system is in contradiction with the above non-compactness proposition. □

2.3 Full semantics

One cause of non-compactness in the previous section is the assumption that we deal with finite heaps only. In this section, we consider a more liberal semantics: the *full semantics* of separation logic. In the full semantics, we leave out the restriction that we only consider heaps that are finite. Recall the discussion in the introduction chapter that motivates our choice (see Section 1.3). Does this modification resolve the problem of non-compactness, or will the resulting semantics also be non-compact? That is the main question we answer in this section.

Again, let $\mathfrak{A} = (A, \mathcal{I})$ be a structure. A *heap* of $\mathfrak{A}$ is a partial function from A to A. Every finite heap is a heap, and also every function from A to A is a heap. If A is infinite, then there are heaps that are not finite heaps. In the case A is finite, then every heap is a finite heap. A heap that is not a finite heap is called an infinite heap.

The concepts we have introduced earlier are easily adapted to the new situation. Let h be a (finite or infinite) heap. The domain $\mathrm{dom}(h)$ is the set of locations for which h is defined. For infinite heaps, $\mathrm{dom}(h)$ is an infinite set. The empty heap remains. The two operations of heap update $h[a := a']$ and heap clear $h[a := \bot]$ can be extended to infinite heaps: their definition remains the same. However,

$\mathrm{dom}(h[a := a'])$ and $\mathrm{dom}(h[a := \bot])$ remain infinite if $\mathrm{dom}(h)$ is infinite. Also the concept of heap partitioning can be extended to infinite heaps: we have that either h_1 or h_2 is an infinite heap if $h \equiv h_1 \uplus h_2$ and h is an infinite heap.

Similarly, we can adapt the satisfaction relation, which for the full semantics we denote by $\mathfrak{A}, h, \rho \models^{\mathbf{FSL}} \phi$, where the superscript **FSL** stands for Full Separation Logic. Revisiting Definition 2.2.1, we now have the following adapted clauses:

Definition 2.3.1 (Satisfaction relation)**.** Given a structure $\mathfrak{A} = (A, \mathcal{I})$, a valuation ρ of $\mathfrak{A}$, a heap h of $\mathfrak{A}$, and a separation logic formula ϕ. The satisfaction relation $\mathfrak{A}, h, \rho \models^{\mathbf{FSL}} \phi$ is defined inductively on the structure of ϕ:

- ...
- $\mathfrak{A}, h, \rho \models^{\mathbf{FSL}} \phi * \psi$ iff $\mathfrak{A}, h_1, \rho \models^{\mathbf{FSL}} \phi$ and $\mathfrak{A}, h_2, \rho \models^{\mathbf{FSL}} \psi$ for some h_1, h_2 such that $h \equiv h_1 \uplus h_2$,
- $\mathfrak{A}, h, \rho \models^{\mathbf{FSL}} \phi \mathrel{-\!\!*} \psi$ iff $\mathfrak{A}, h', \rho \models^{\mathbf{FSL}} \phi$ implies $\mathfrak{A}, h'', \rho \models^{\mathbf{FSL}} \psi$ for every h', h'' such that $h'' \equiv h \uplus h'$,
- ...

where the heaps h, h_1, h_2, h', h'' range over (finite or infinite) heaps, not only finite heaps as in the standard semantics.

With this satisfaction relation, we can also introduce the usual no(ta)tions of validity, semantic consequence, and theories, but we have to make sure that we consider all, finite or infinite, heaps universally. For example, we write $\mathfrak{A} \models^{\mathbf{FSL}} \phi$ to mean $\mathfrak{A}, h \models^{\mathbf{FSL}} \phi$ for all (finite or infinite) heaps h.

Comparing the standard semantics and the full semantics with respect to the satisfaction relation, we can see some obvious connections. For structures, the full semantics also has the first-order theory included in its separation theory: we have $Th_1^{\mathbf{CL}}(\mathfrak{A}) \subseteq Th^{\mathbf{FSL}}(\mathfrak{A})$. If the domain of our structure is finite, every heap is also a finite heap: so there is no distinction between the two semantics. However, there are structures with an infinite domain in which the full semantics and standard semantics differ in the sentences they satisfy.

Consider the sentence $\phi = \exists x.(x \hookrightarrow -) \mathrel{-\!\!*} \bot$, and take a structure $\mathfrak{N}$ with the naturals $\mathbb{N}$ as domain. The sentence is considered false with respect to the standard semantics, but the sentence is satisfiable with respect to the full semantics.

- (**SSL**) Let h be an arbitrary finite heap. Let m be the maximum location of h, or 0 if h is empty. Then surely $\epsilon[m + 1 := 0]$ is disjoint from h. However, $\mathfrak{N}, h[m + 1 := 0] \not\models^{\mathbf{SSL}} \bot$.
- (**FSL**) To show satisfiability, we give a heap h such that $\mathfrak{N}, h \models^{\mathbf{FSL}} \phi$. Take any function for h such that $\mathrm{dom}(h) = \mathbb{N}$, i.e. all locations are allocated. Now there is no disjoint h' that is not empty. Hence, for any choice of value for x, the formula $(x \hookrightarrow -) \mathrel{-\!\!*} \bot$ is vacuously satisfied.

To further explore the semantics of separation logic, we introduce the following abbreviation (also called the *box modality*) that we characterize below:

$$\blacksquare\phi \text{ abbreviates } \top * (\mathbf{emp} \wedge (\top \mathrel{-\!\!*} \phi))$$

Formulas placed directly within the context of the box modality are interpreted with respect to an arbitrary heap. Thus, this modality expresses a limited form of second-order universal quantification. One may think of the box modality acting as a binder of the points-to construct.

Proposition 2.3.2. *The following holds:*

- $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \blacksquare\phi$ *if and only if* $\mathfrak{A}, h', \rho \models^{\mathbf{SSL}} \phi$ *for every finite heap* h'*,*
- $\mathfrak{A}, h, \rho \models^{\mathbf{FSL}} \blacksquare\phi$ *if and only if* $\mathfrak{A}, h', \rho \models^{\mathbf{FSL}} \phi$ *for every heap* h'*.*

Proof. We show it for the standard semantics first.
$\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \top * (\mathbf{emp} \wedge (\top \mathrel{-\!\!*} \phi))$
if and only if
$\mathfrak{A}, \epsilon, \rho \models^{\mathbf{SSL}} \top \mathrel{-\!\!*} \phi$
if and only if
$\mathfrak{A}, h', \rho \models^{\mathbf{SSL}} \phi$ for every finite heap h'.
The proof for the full semantics is similar, but quantifying over all heaps h'. □

Dually, we introduce the *diamond modality*

$$\blacklozenge\phi \text{ abbreviates } \neg\blacksquare\neg\phi$$

which expresses a limited form of second-order existential quantification.

Corollary 2.3.3. *The following holds:*

- $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \blacklozenge\phi$ *if and only if* $\mathfrak{A}, h', \rho \models^{\mathbf{SSL}} \phi$ *for some finite heap* h'*,*
- $\mathfrak{A}, h, \rho \models^{\mathbf{FSL}} \blacklozenge\phi$ *if and only if* $\mathfrak{A}, h', \rho \models^{\mathbf{FSL}} \phi$ *for some heap* h'*.*

The above box modality can be used to characterize finiteness of the domain of the structure in the case of full separation logic. Let *fin* abbreviate

$$\blacksquare(\mathit{tot}(\hookrightarrow) \wedge \mathit{inj}(\hookrightarrow) \rightarrow \mathit{surj}(\hookrightarrow))$$

where the abbreviations *tot*, *inj*, *surj* of Proposition A.2.13 can be reused, but applied to points-to by considering the primitive separation logic formula $(x \hookrightarrow y)$ to be obtained as if $\hookrightarrow$ were a 2-ary relation (cf. the abbreviations below).

Proposition 2.3.4. $\mathfrak{A} \models^{\mathbf{FSL}}$ *fin if and only if the domain of* $\mathfrak{A}$ *is finite.*

Proof. Recall Proposition A.2.12, that a set is finite if and only if every injective total function on that set is a surjection. In contrast to Proposition A.2.13, we do not need $\mathit{fun}(\hookrightarrow)$ because in the semantics of separation logic we already have that heaps are partial functions from which this property holds for every heap. □

The above characterization fails for standard separation logic. In that semantics, the formula is valid: if the domain of the structure is finite then every injective total heap is already a surjection, and if the domain of the structure is infinite then there is no finite heap that satisfies $tot(\hookrightarrow)$.

Going further, we can give a sufficient condition for the finiteness of the domain of the heap. Surely, for the standard semantics, this condition is useless since every heap already is finite. However, in the full semantics, the condition can be useful in certain contexts to restrict attention to heaps with a finite domain. We extend our signature with an additional 1-ary predicate symbol, P. We then have the following sentence:

$$\forall x.\ P(x) \leftrightarrow (x \hookrightarrow -) \tag{2.1}$$

which expresses that the extension of the predicate P coincides with the domain of the current interpretation of the heap. This sentence can not be valid if the structure has at least one element in its domain, since each structure gives an *a priori* interpretation of P independently of the heap, which may satisfy the formula for one heap (say, the empty heap) but not for another (say, the non-empty heap).

However, we are still able to use the formula to capture the domain of the heap at top-level. How this can be used will become clear later. To express that the extension of P is finite we adapt our previous formula in the following way:

$$\blacksquare(tot^P(\hookrightarrow) \wedge inj^P(\hookrightarrow) \rightarrow surj^P(\hookrightarrow)) \tag{2.2}$$

where we introduce the following relativized abbreviations:

- $inj^P(\hookrightarrow)$ abbreviates $\forall x, y, z.\ P(x) \wedge P(y) \wedge P(z) \wedge (x \hookrightarrow z) \wedge (y \hookrightarrow z) \rightarrow x \doteq y$,
- $tot^P(\hookrightarrow)$ abbreviates $\forall x.\ P(x) \rightarrow \exists y.\ P(y) \wedge (x \hookrightarrow y)$,
- $surj^P(\hookrightarrow)$ abbreviates $\forall y.\ P(y) \rightarrow \exists x.\ P(x) \wedge (x \hookrightarrow y)$.

Even in case the domain of our structure is infinite and in the full semantics the box modality universally quantifies over all heaps, the above formula expresses that all total injective functions on P must be surjective on P, and thus that P is finite (for the same reason as in Proposition A.2.12).

Given that we can capture the domain of the heap at the top-level by a predicate P, and that we can express that the extension of P is finite, we have also non-compactness for the full semantics.

Lemma 2.3.5 (Non-compactness full semantics)**.** *It is not the case that* Γ *is finitely satisfiable implies that* Γ *is satisfiable.*

Proof. The set of sentences Γ is finitely satisfiable but not satisfiable:

$$\Gamma = \{\phi_n \mid n \in \mathbb{N}\} \cup \{\psi\}$$

where ϕ_n expresses that there are at least $n+1$ allocated and distinct elements as in Lemma 2.2.8, and ψ is the conjunction of the sentences of Equations (2.1) and (2.2). Each finite subset of Γ is satisfiable: we can construct structures in the same

way as before, but now also select as interpretation for P the domain of the finite heap used. However, Γ is not satisfiable: due to ψ we need to choose a finite heap h and then not all ϕ_n can be satisfied. □

Corollary 2.3.6. *There is no sound, complete, finitary proof system for* **FSL**.

The above argument of non-compactness relies on the presence of separating implication (in the box modality), whereas in Lemma 2.2.8 we need not rely on any separating connective. Can we also show non-compactness without relying on separating implication?

We introduce the following abbreviation (also called the *sub-heap modality*):

$$\Box\phi \text{ abbreviates } \neg(\top * \neg\phi)$$

that expresses that ϕ holds for every sub-heap of the current heap. Bannister and others have also introduced a related connective, called *separating coimplication*, in [15], of which this sub-heap modality is an instance. Formally, a heap h' is a sub-heap of heap h, denoted $h' \subseteq h$, if $\mathrm{dom}(h') \subseteq \mathrm{dom}(h)$ and $h'(a) = h(a)$ for all $a \in \mathrm{dom}(h')$. We have that $h \equiv h_1 \uplus h_2$ implies $h_1 \subseteq h$ and $h_2 \subseteq h$. Conversely, if $h_2 \subseteq h$ then there exists a heap h_1 such that $h \equiv h_1 \uplus h_2$.

Proposition 2.3.7. *The following holds:*

- $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \Box\phi$ *if and only if* $\mathfrak{A}, h', \rho \models^{\mathbf{SSL}} \phi$ *for every finite heap* $h' \subseteq h$,
- $\mathfrak{A}, h, \rho \models^{\mathbf{FSL}} \Box\phi$ *if and only if* $\mathfrak{A}, h', \rho \models^{\mathbf{FSL}} \phi$ *for every heap* $h' \subseteq h$.

Proof. We show it for the standard semantics first.
$\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \neg(\top * \neg\phi)$
if and only if
$\mathfrak{A}, h_1, \rho \models^{\mathbf{SSL}} \bot$ or $\mathfrak{A}, h_2, \rho \models^{\mathbf{SSL}} \phi$ for all h_1, h_2 such that $h \equiv h_1 \uplus h_2$
if and only if
$\mathfrak{A}, h', \rho \models^{\mathbf{SSL}} \phi$ for every $h' \subseteq h$.
The proof for the full semantics is similar. □

Since a heap is a partial function on the domain of a structure, the values that are assigned to locations can themselves be used as locations. Intuitively, if $(x \hookrightarrow y)$ and $(y \hookrightarrow z)$ are satisfied, we may think of traversing the pointer at location x to obtain the value y, which is used as a location and can be traversed to obtain the value z. Such traversal is denoted by $(x \hookrightarrow y \hookrightarrow z)$. In general, we can form a *chain* of traversals

$$(x_0 \hookrightarrow x_1 \hookrightarrow \ldots \hookrightarrow x_n)$$

which means $(x_i \hookrightarrow x_{i+1})$ for every $0 \leq i < n$. We traverse n locations to end up with the value x_n. We say one can *reach* value x_n by traversing n locations. A heap with cycles allows the same location to be revisited in a traversal:

$$x_0 \hookrightarrow \ldots \hookrightarrow x_n \hookrightarrow x_0$$

where we end up back at the starting location x_0 and we can infinitely continue traversing along the same locations of the chain. Although the cycle comprises finitely many locations, the chain can be extended infinitely long.

A *dead-end* is a location that is not allocated: we are unable to continue the traversal through that location. Formally, x is a dead-end if $\forall y(x \not\rightsquigarrow y)$ (or, equivalently, $(x \not\rightsquigarrow -)$). If a location is not a dead-end, we may progress our traversal from that location. Conversely, an *unreachable* value (in the sense that it is unreachable from the heap) is a value which is not pointed to by any location allocated on the heap. Formally, x is unreachable if $\forall y(y \not\rightsquigarrow x)$. A value is reachable (from the heap) if it is not unreachable, so there exists a location which points to that value. One can think of the set of all values that are reachable from the heap as comprising the contents of the heap. Thus, a value unreachable from some heap is a value that is not contained in that heap.

In both the standard semantics and the full semantics, there is a heap in which every location is a dead-end: the empty heap. The empty heap has no contents. In the full semantics, we can also have heaps in which all values of the domain of the structure are reachable. Then the contents of the heap is the same as the domain of the structure. This is not possible in the standard semantics for structures with an infinite domain.

A heap h is *well-founded* if for every non-empty sub-heap $h' \subseteq h$ there is a value not contained in h'. Alternatively, a heap h is well-founded if there exists no infinite sequence of locations $a_0, a_1, \ldots$ such that $h(a_{n+1}) = a_n$. There are non-well-founded heaps, for example, a heap which has a cycle.

With the sub modality we can express the heap is *well-founded*, by the sentence

$$\Box(\mathbf{emp} \vee \exists x((x \hookrightarrow -) \wedge \forall y((y \hookrightarrow -) \rightarrow (y \not\rightsquigarrow x)))). \tag{2.3}$$

Lemma 2.3.8 (Non-compactness full semantics)**.** *It is not the case that* Γ *is finitely satisfiable implies* Γ *is satisfiable for theories* Γ *in which the separating implication connective does not occur.*

Proof. Let $c_0, c_1, \ldots$ be countably many individual constant symbols (these can be encoded using unary predicate symbols with the appropriate property of existence and uniqueness). The set of sentences Γ is finitely satisfiable but not satisfiable:

$$\Gamma = \{c_{n+1} \hookrightarrow c_n \mid n \geq 0\} \cup \{\psi\}$$

where ψ is Equation (2.3). Every finite subset of Γ is satisfiable: take as domain of our structure $\mathbb{N}$, interpret the individual constants $c_0, \ldots$ by unique naturals, and construct a finite heap that satisfies the (finitely many) points-to constraints. For every sub-heap of the constructed finite heap there exists a value not contained in it, so ψ is also satisfied. However, Γ is not satisfiable, as that would imply there is an infinite sequence of locations $a_0, a_1, \ldots$ such that $h(a_{n+1}) = a_n$. □

Note that we did not need to add to Γ any sentence expressing that $c_i \neq c_j$ for $i \neq j$, since this possibility is already ruled out by ψ: if $c_i = c_j$ for some $i \neq j$ then there is a cycle in the heap and thus is the heap non-well-founded.

Corollary 2.3.9. *There is no sound, complete, finitary proof system restricted to formulas without separating implication for* **FSL**.

The diamond and sub modality together can be used to express that the domain of the structure is countably infinite. The main idea is that we can express that there is a smallest heap that has the same structure as the natural numbers, and that every element of the domain is reachable in that heap. For notational convenience, we introduce the following abbreviations: let *zero* abbreviate the sentence

$$\exists x((x \hookrightarrow -) \wedge \forall y(y \not\hookrightarrow x))$$

which expresses that there is some unreachable value in the domain of the heap, and let *nat* abbreviate the sentence

$$tot(\hookrightarrow) \wedge inj(\hookrightarrow) \wedge zero.$$

If $\mathfrak{A}, h \models^{\mathbf{FSL}} nat$ then h encodes some copy of the natural numbers. We can then define $f : \mathbb{N} \to A$ inductively by $f(0) = a_0$, where

$$\mathfrak{A}, h, \rho[x := a_0] \models^{\mathbf{FSL}} \forall y((x \hookrightarrow -) \wedge y \not\hookrightarrow x))$$

for some valuation ρ (i.e. a_0 is some 'minimal' element of h), and $f(n+1) = a'$, where $h(f(n)) = a'$. The latter is defined on $f(n)$ because of $tot(\hookrightarrow)$, since $\mathfrak{A}, h \models^{\mathbf{FSL}} \forall x \exists y(x \hookrightarrow y)$ means that the domain of h equals D. Note that the encoding is not necessarily unique: there could be multiple 'minimal' elements.

Next, let *ind* we denote the formula

$$(zero \wedge (\forall x, y.\ (x \hookrightarrow y) \to (y \hookrightarrow -))) \to \forall x(x \hookrightarrow -)$$

which expresses that the domain of the heap and the structure coincide, if there is some unreachable value in the domain of the heap and every reachable value is also in the domain of the heap. The formula $\blacklozenge(nat \wedge \Box ind)$ then characterizes the class of countably infinite structures.

Proposition 2.3.10. $\mathfrak{A} \models^{\mathbf{FSL}} \blacklozenge(nat \wedge \Box ind)$ *if and only if the domain of* $\mathfrak{A}$ *is countably infinite.*

Proof. Let $\mathfrak{A} \models^{\mathbf{FSL}} \blacklozenge(nat \wedge \Box ind)$. From Corollary 2.3.3, there exists a heap h' such that $\mathfrak{A}, h' \models^{\mathbf{FSL}} nat \wedge \Box ind$. From the definition of *nat*, as explained above, it follows that h' contains a copy of the natural numbers. From Proposition 2.3.7 and that $\boxdot ind$ is satisfied, it follows that the sub-heap h'' of h' satisfies *ind*. Now choose a particular sub-heap h''_0 of h' which contains precisely the copy of the natural numbers, and nothing else. Thus h''_0 satisfies $zero \wedge \forall x, y((x \hookrightarrow y) \to (y \hookrightarrow -))$, and so we obtain that $\mathfrak{A}, h''_0 \models^{\mathbf{FSL}} \forall x(x \hookrightarrow -)$, that is, the domain of h''_0 (which is $\mathbb{N}$) equals the domain of $\mathfrak{A}$.

Conversely, let A be countably infinite. For any enumeration of the domain of $\mathfrak{A}$, we can construct a corresponding heap h which encodes this enumeration and thus satisfies $\mathfrak{A}, h \models^{\mathbf{FSL}} nat \wedge \Box ind$. □

Corollary 2.3.11. $\mathfrak{A} \models^{\mathbf{FSL}} \neg \mathit{fin} \wedge \blacksquare(\mathit{nat} \to \neg\Box \mathit{ind})$ *if and only if the domain of* $\mathfrak{A}$ *is uncountably infinite.*

Thus, we are able to distinguish whether a structure is countably infinite or not. Consequently, full separation logic cannot satisfy the Löwenheim-Skolem theorem.

2.4 Embeddings

In the previous sections we defined the standard semantics and full semantics for separation logic. But what is the precise relation between the validity $\models^{\mathbf{FSL}} \phi$ and $\models^{\mathbf{SSL}} \phi$ for formulas ϕ of separation logic? It seems possible to embed the valid sentences of standard separation logic in full separation logic, if there is a formula *finh* that expresses the finiteness of the domain of the (current) heap in full separation logic. Consider the following translation $T(\phi)$ defined by induction on the separation logic formula ϕ:

- $T(\bot) = \bot$,
- $T(x \doteq y) = (x \doteq y)$,
- $T(x \hookrightarrow y) = (x \hookrightarrow y)$,
- $T(C(x_1, \ldots, x_n)) = C(x_1, \ldots, x_n)$,
- $T(\phi \to \psi) = T(\phi) \to T(\psi)$,
- $T(\forall x \phi) = \forall x (T(\phi))$,
- $T(\phi * \psi) = T(\phi) * T(\psi)$,
- $T(\phi \mathrel{-\!\!*} \psi) = (\mathit{finh} \wedge T(\phi)) \mathrel{-\!\!*} T(\psi)$.

Proposition 2.4.1. $\mathfrak{A}, h, \rho \models^{\mathbf{SSL}} \phi$ *if and only if* $\mathfrak{A}, h, \rho \models^{\mathbf{FSL}} \mathit{finh} \wedge T(\phi)$.

Proof. Note that h necessarily is a finite heap. A finite heap can only be split into finite subheaps. In the translation of separating implication we ensure that in the full semantics we only quantify over finite heaps. □

Recall that we have already seen a *sufficient* condition for finiteness of the domain of the heap, the conjunction of Equations (2.1) and (2.2). However, that condition only works at the top-level and makes use of an additional predicate symbol added to the signature that can only be given an interpretation *a priori*, and not depending on the current interpretation of the heap as modified by the separating connectives. However, we still lack a *necessary* condition and thus have the following open problem:

Problem 2.1. *Is there a sentence in separation logic finh such that* $\mathfrak{A}, h \models^{\mathbf{FSL}} \mathit{finh}$ *if and only if* $\mathrm{dom}(h)$ *is finite?*

Next, we then turn our attention to the relation between separation logic and classical logic. Firstly, we embed a subset of separation logic, called *separation logic light*, into first-order logic. We argue that the resulting embedding preserves semantics, but does not give us a compact semantic consequence relation. Secondly, we show that it is possible to translate a formula of separation logic into a formula of second-order logic that preserves its semantics.

In the previous section, we have seen there is no sound, complete, finitary proof system for full separation logic, even when we restrict to formulas without separating implication. Was the problem of non-compactness caused by the fact that, in the modality $\Box\phi$, that is defined as $\neg(\top * \neg\phi)$, we use negation outside of separating conjunction? We now study a restricted set of separation logic formulas, in which we restrict the occurrences of the separating conjunction. Are we able to show compactness whenever we disallow using separating conjunction under negation?

Recall that in a pure formula of separation logic we do not have any occurrences of points-to, separating conjunction, or separating implication. We now consider *semi-pure* formulas: in a semi-pure formulas there are no occurrences of separating conjunction or separating implication, but we are allowed to use points-to. We then define a restricted set of formulas of separation logic, called the formulas of separating logic light (SLL), as follows:

- a semi-pure formula of separation logic is a formula of separation logic light,
- $(\phi * \psi)$ is a formula of separation logic light given that ϕ and ψ are formulas of separation logic light.

We thus restrict the use of separating conjunction to the 'top level' of the formula. For the purposes of our exposition, we use only the classical connectives of conjunction, disjunction and negation: in this case, implication $(\phi \to \psi)$ is interpreted as material implication $(\neg\phi \vee \psi)$. Note that the use of negation in a semi-pure formula can be pushed to the leaves of the formula, by the classical equivalences:

- $\neg(\forall x\phi)$ reduces to $\exists x(\neg\phi)$,
- $\neg(\exists x\phi)$ reduces to $\forall x(\neg\phi)$,
- $\neg(\phi \wedge \psi)$ reduces to $(\neg\phi) \vee (\neg\psi)$,
- $\neg(\phi \vee \psi)$ reduces to $(\neg\phi) \wedge (\neg\psi)$.

In fact, any semi-pure formula can first be normalized into prenex normal form, and then the negation in the quantifier-free part can be pushed to the leaves. Such formulas of separation logic light are said to be in *normal form*.

We now consider whether satisfiability of separation logic light formulas is compact, that is, whether a theory of separation logic light is satisfiable if and only if that theory is finitely satisfiable.

Proposition 2.4.2. *Given a set Γ of separation logic light formulas. There exists a structure $\mathfrak{A}$ and heap h such that $\mathfrak{A}, h \models^{\mathbf{FSL}} \Gamma$, if and only if, for every finite subset $\Gamma_0 \subseteq \Gamma$ there exists a structure $\mathfrak{A}$ and heap h such that $\mathfrak{A}, h \models^{\mathbf{FSL}} \Gamma_0$.*

Proof. We introduce the following first-order translation $\phi@R$ of separation logic light formulas in normal form, where R is a binary relation symbol of the signature (so necessarily different from $\hookrightarrow$):

- $(x \doteq y)@R = (x \doteq y)$,
- $(x \hookrightarrow y)@R = R(x, y)$,
- $C(x_1, \ldots, x_n)@R = C(x_1, \ldots, x_n)$,
- $(\neg\phi)@R = \neg(\phi@R)$,
- $(\phi \wedge \psi)@R = \phi@R \wedge \psi@R$ and $(\phi \vee \psi)@R = \phi@R \vee \psi@R$,
- $(\forall x\phi)@R = \forall x(\phi@R)$ and $(\exists x\phi)@R = \exists x(\phi@R)$,
- $(\phi * \psi)@R = R \equiv R_1 \uplus R_2 \wedge \phi@R_1 \wedge \psi@R_2$,

where by $R \equiv R_1 \uplus R_2$ we denote the formula

$$\forall x, y.\, (R(x, y) \leftrightarrow R_1(x, y) \vee R_2(x, y)) \wedge \neg(R_1(x, y) \wedge R_2(x, y))$$

that expresses that R is the union of R_1 and R_1 and that R_1 and R_2 are disjoint. The binary relation symbols R, R_1 and R_2 are 'fresh'. It is sufficient to show that ϕ is satisfiable (in full separation logic) if and only if $\mathit{fun}(R) \wedge \phi@R$ is satisfiable (in classical logic). More precisely, $\mathfrak{A}, h, \rho \models^{\mathbf{FSL}} \phi$ for some $\mathfrak{A}, h, \rho$ if and only $\mathfrak{B}, \rho' \models^{\mathbf{CL}} \mathit{fun}(R) \wedge \phi@R$ for some $\mathfrak{B}, \rho'$. Note that the interpretation of R is the graph of the heap h.

Consequently, compactness of first-order logic implies compactness of separation logic light. Let Γ be an infinite set of formulas of SLL and construct a corresponding set $\Gamma' = \{\mathit{fun}(R) \wedge \phi@R \mid \phi \in \Gamma\}$ for some fixed binary relation symbol R. Note that Γ' may require the introduction of a countably infinite number of fresh binary relation symbols. This is however no problem because first-order logic allows for the addition of a countably infinite set of relation symbols to the signature without affecting satisfiability. Now, given that every finite subset of Γ is satisfiable, so is every finite subset of Γ'. By the compactness of first-order logic, we then have that Γ' is also satisfiable. From the structure witnessing satisfiability of Γ' we can show the satisfiability of Γ in full separation logic. Conversely, assume that Γ is satisfiable in full separation logic, then we can use the same structure as witness for the satisfiability of every finite subset of Γ. □

A similar result can be obtained for standard separation logic.

Although we are able to show compactness of the satisfiability relation, we have not yet reached the conclusion that the semantics can be useful for an adequate proof theory. This is because compactness of the satisfiability relation does not imply that the semantic consequence relation is compact. Non-compactness of the consequence relation for separation logic light follows directly from the above argument involving well-founded relations.

Lemma 2.4.3. *Given a set Γ of separation logic light formulas, and a formula ϕ of separation logic light. There is a counter-example to the claim: $\Gamma \models^{\mathbf{FSL}} \phi$ implies there exists a finite subset $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \models^{\mathbf{FSL}} \phi$.*

Proof. Let Γ denote the set of separation logic light formulas

$$\{(c_{n+1} \hookrightarrow c_n) \mid n \geq 0\}$$

where $c_0, c_1, \ldots$ are individual constant symbols (these can be again encoded). It follows that

$$\Gamma \models^{\mathbf{FSL}} \top * (\neg \mathbf{emp} \wedge \forall x((x \hookrightarrow -) \to \exists y(y \hookrightarrow x))),$$

where also the latter is a separation logic light formula. The formula expresses that there exists some non-empty sub-heap, in which every location in the domain is reachable. Clearly, this is the case if we have a cycle in the heap. However, this is also the case when we have an infinite chain in the heap. But, importantly, there does not exist a finite subset Γ_0 of Γ such that

$$\Gamma_0 \models^{\mathbf{FSL}} \top * (\neg \mathbf{emp} \wedge \forall x((x \hookrightarrow -) \to \exists y(y \hookrightarrow x))).$$

As soon as we restrict ourselves to a finite subset Γ_0 of Γ, we know we only have to interpret finitely many values for c_i. In that case, we no longer necessarily have a non-empty subheap in which every location in the domain is reachable. □

This failure of compactness of the semantic consequence relation is important for the design of a finitary proof system (see also Theorem A.4.3): it is not possible to have a sound and complete, finitary proof system for **FSL** even when we restrict to the formulas of separation logic light.

We now turn to the relation between separation logic and second-order classical logic. In particular, we focus on dyadic second-order logic, where we restrict second-order quantification to variables of arity 2. It is possible to embed separation logic in dyadic second-order logic, meaning that second-order logic is at least as expressive as separation logic.

Given a first-order signature Σ. We define the following translation of formulas of separation logic into formulas of second-order logic. The translation is parameterized by a higher-order variable of arity 2, which intuitively takes the place of the points-to construct. This translation makes obvious how separating conjunction and separating implication can be read as particular second-order quantifications.

Definition 2.4.4. Let R, R_1, R_2, R', R'' be variables of arity 2. The function $[-](R)$ translates formulas of separation logic to formulas of dyadic second-order logic, and is defined inductively as follows:

- $[\bot](R) = \bot$,
- $[x \doteq y](R) = (x \doteq y)$,
- $[x \hookrightarrow y](R) = R(x, y)$,

- $[C(x_1, \ldots, x_n)](R) = C(x_1, \ldots, x_n)$,
- $[\phi \to \psi](R) = [\phi](R) \to [\psi](R)$,
- $[\forall x \phi](R) = \forall x([\phi](R))$,
- $[\phi * \psi](R) = \exists R_1 \exists R_2 (R \equiv R_1 \uplus R_2 \wedge [\phi](R_1) \wedge [\psi](R_2))$,
- $[\phi \mathrel{-\!\!*} \psi](R) = \forall R'' \forall R' (\mathit{fun}(R'') \wedge R'' \equiv R' \uplus R \to [\phi](R') \to [\psi](R''))$,

where $R \equiv R_1 \uplus R_2$ denotes the formula

$$\forall x, y.\, (R(x, y) \leftrightarrow R_1(x, y) \vee R_2(x, y)) \wedge \neg(R_1(x, y) \wedge R_2(x, y))$$

where R_1 and R_2 are variables of arity 2.

Note that, in second-order logic, we have the following properties:

- $\mathit{fun}(R) \wedge R \equiv R_1 \uplus R_2 \to \mathit{fun}(R_1) \wedge \mathit{fun}(R_2)$,
- $\mathit{fun}(R) \wedge \mathit{fun}(R'') \wedge R'' \equiv R' \uplus R \to \mathit{fun}(R')$,

hence we need not explicitly mention that all quantified variables are functional.

Proposition 2.4.5. *Given structure $\mathfrak{A} = (A, \mathcal{I})$ and formula ϕ of separation logic. We have that $\mathfrak{A} \models^{\mathbf{FSL}} \phi$ if and only if $\mathfrak{A} \models^{\mathbf{CL}} \forall R(\mathit{fun}(R) \to [\phi](R))$.*

Proof. By unfolding the semantics of separation logic, and induction on the structure of the formula ϕ, where the parameter R represents the current heap in the semantics of separation logic. □

2.5 Relational separation logic

We now ask ourselves the question: is separation logic also at least as expressive as dyadic second-order logic? Although this question is still open for full separation logic at the point of this writing, we can formulate a sufficient condition for having that separation logic is at least as expressive as dyadic second-order logic. That condition is the expressivity of the *binding operator* which captures the current interpretation of the heap in a second-order variable. However, before we are able to introduce the binding operator, we need to discuss two differences between separation logic and dyadic second-order logic.

The first difference between separation logic and dyadic second-order logic is that in our presentation, separation logic speaks about heaps as being *partial functions* whereas in dyadic second-order logic the second-order variables denote *binary relations*. We could thus generalize the semantics of separation logic, and remove the restriction to partial functions at the semantic level; instead consider a separation logic of binary relations called *relational separation logic*. In relational separation logic, we treat the primitive points-to $(x \hookrightarrow y)$ as a binary relation that is *not necessarily* functional, in contrast to the semantics of separation logic as presented in the previous chapter.

Let A be a set (say, the domain of a structure). A relation $\mathcal{R}$ is a subset of the Cartesian product $A \times A$, that is, $\mathcal{R} \subseteq A \times A$. This is in contrast to heaps, which are partial functions from A to A. Every heap can be seen as a relation, where we take the graph of the partial function. We introduce the following notions on relations. The domain of a relation $\mathrm{dom}(\mathcal{R})$ is the set $\{a \mid (a, a') \in \mathcal{R} \text{ for some } a'\}$. By $\mathcal{R} \perp \mathcal{R}'$ we denote that the domains of the relations $\mathcal{R}$ and $\mathcal{R}'$ are disjoint. A relation $\mathcal{R}$ is functional if for every element $a \in \mathrm{dom}(\mathcal{R})$ there is a unique a' such that $(a, a') \in \mathcal{R}$. If $\mathcal{R}$ and $\mathcal{R}'$ are functional, $\mathcal{R} \cap \mathcal{R}' = \emptyset$ if and only if $\mathcal{R} \perp \mathcal{R}'$.

Similar to our previous discussion regarding the semantics of *standard* and *full* separation logic, we can introduce the following relational separation logic semantics:

- **WRSL** is *weak relational separation logic* where second-order quantification of the separating connectives ranges over *all finite* binary relations,
- **FRSL** is *full relational separation logic* where second-order quantification of the separating connectives ranges over *all* binary relations.

We can now give the definitions of weak relational separation logic, **WRSL**, and full relational separation logic, **FRSL**. Only the definition for **FRSL** is shown in full below, that of **WRSL** is easily obtained by restricting to finite relations.

Definition 2.5.1 (Satisfaction relation)**.** Given a structure $\mathfrak{A} = (A, \mathcal{I})$, a valuation ρ of $\mathfrak{A}$, a finite binary relation $\mathcal{R} \subseteq A \times A$, and a separation logic formula ϕ. The satisfaction relation $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{WRSL}} \phi$ is defined inductively on ϕ:

- $\ldots$
- $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{WRSL}} \phi * \psi$ iff $\mathfrak{A}, \mathcal{R}_1, \rho \models^{\mathbf{WRSL}} \phi$ and $\mathfrak{A}, \mathcal{R}_2, \rho \models^{\mathbf{WRSL}} \psi$ for some finite $\mathcal{R}_1, \mathcal{R}_2 \subseteq A \times A$ such that $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ and $\mathcal{R}_1 \perp \mathcal{R}_2$,
- $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{WRSL}} \phi \mathrel{-\!\!*} \psi$ iff $\mathfrak{A}, \mathcal{R}', \rho \models^{\mathbf{WRSL}} \phi$ implies $\mathfrak{A}, \mathcal{R} \cup \mathcal{R}', \rho \models^{\mathbf{WRSL}} \psi$ for every finite $\mathcal{R}' \subseteq A \times A$ such that $\mathcal{R} \perp \mathcal{R}'$.

Definition 2.5.2 (Satisfaction relation)**.** Given a structure $\mathfrak{A} = (A, \mathcal{I})$, a valuation ρ of $\mathfrak{A}$, a binary relation $\mathcal{R} \subseteq A \times A$, and a separation logic formula ϕ. The satisfaction relation $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}} \phi$ is defined inductively on the structure of ϕ:

- $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}} \bot$ never holds,
- $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}} (x \doteq y)$ iff $\rho(x) = \rho(y)$,
- $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}} (x \hookrightarrow y)$ iff $(\rho(x), \rho(y)) \in \mathcal{R}$,
- $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}} C(x_1, \ldots, x_n)$ iff $(\rho(x_1), \ldots, \rho(x_n)) \in C^{\mathcal{I}}$,
- $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}} \phi \to \psi$ iff $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}} \phi$ implies $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}} \psi$,
- $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}} \forall x \phi$ iff $\mathfrak{A}, \mathcal{R}, \rho[x := a] \models^{\mathbf{FRSL}} \phi$ for every $a \in A$,

- $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}} \phi * \psi$ iff $\mathfrak{A}, \mathcal{R}_1, \rho \models^{\mathbf{FRSL}} \phi$ and $\mathfrak{A}, \mathcal{R}_2, \rho \models^{\mathbf{FRSL}} \psi$ for some $\mathcal{R}_1, \mathcal{R}_2 \subseteq A \times A$ such that $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ and $\mathcal{R}_1 \perp \mathcal{R}_2$,
- $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}} \phi \mathrel{-\!\!*} \psi$ iff $\mathfrak{A}, \mathcal{R}', \rho \models^{\mathbf{FRSL}} \phi$ implies $\mathfrak{A}, \mathcal{R} \cup \mathcal{R}', \rho \models^{\mathbf{FRSL}} \psi$ for every $\mathcal{R}' \subseteq A \times A$ such that $\mathcal{R} \perp \mathcal{R}'$.

Note that in the semantics of the separating connectives we require disjointedness of the *domains* of the relations, not the disjointedness of the relations themselves. This design choice is discussed in more detail later. In the definition of **WRSL**, it is not strictly necessary to require that $\mathcal{R}_1$ and $\mathcal{R}_2$ are finite, since this follows from the fact that their union must be finite too.

We can embed (functional) separation logic in relational separation logic as follows (we here only sketch the main ideas, and leave out the technical details). Let ϕ be a formula of separation logic. We then define the translation $T(\phi)$ by induction on the separation logic formula ϕ:

- $T(\bot) = \bot$,
- $T(x \doteq y) = (x \doteq y)$,
- $T(x \hookrightarrow y) = (x \hookrightarrow y)$,
- $T(C(x_1, \ldots, x_n)) = C(x_1, \ldots, x_n)$,
- $T(\phi \to \psi) = T(\phi) \to T(\psi)$,
- $T(\forall x \phi) = \forall x (T(\phi))$,
- $T(\phi * \psi) = T(\phi) * T(\psi)$,
- $T(\phi \mathrel{-\!\!*} \psi) = (\mathit{fun}(\hookrightarrow) \wedge T(\phi)) \mathrel{-\!\!*} T(\psi)$.

We have that this translation preserves the semantics of full separation logic:

$$\mathfrak{A}, h, \rho \models^{\mathbf{FSL}} \phi \text{ if and only if } \mathfrak{A}, h, \rho \models^{\mathbf{FSL}} T(\phi)$$

because $\mathit{fun}(\hookrightarrow)$ already holds with respect to an interpretation based on heaps.

Proposition 2.5.3.

$$\mathfrak{A}, h, \rho \models^{\mathbf{FSL}} \phi \textit{ if and only if } \mathfrak{A}, \mathit{Graph}(h), \rho \models^{\mathbf{FRSL}} T(\phi).$$

Conversely, if we also want to embed relational separation logic in (functional) separation logic, we need to encode binary relations in heaps by the introduction of multiple sorts and a binary relation representing *elementhood*: one sort corresponds to the elements, and another sort corresponds to non-empty sets of elements, and the binary relation is interpreted to represent the elements of non-empty sets of elements. In relational separation logic a location can be related to zero or more values. In our two-sorted separation logic we restrict locations to the sort of elements and values to the sort of non-empty sets of elements: then a heap either leaves a location not allocated or it is allocated and is assigned to a set consisting

of one or more elements. The translation T' maps every connective in an identical manner, but only needs to reinterpret the points-to primitive:

$$T'(x \hookrightarrow y) = \exists S((x \hookrightarrow S) \wedge (y \in S))$$

where S is of the non-empty set sort, and $(y \in S)$ indicates that value y is a member of the set S.

To make it possible to embed relational separation logic in separation logic in this straightforward manner forms the technical motivation for the disjointedness of the domains of relations in the semantics of the separating connectives of relational separation logic. Note that it is also possible to embed one-sorted separation logic into this two-sorted separation logic, where a heap that maps a location to a single value is represented by mapping that same location to the singleton set consisting of the sole value.

The second difference between separation logic and second-order logic is, intuitively, the *local perspective* of separation logic, which is determined by the 'current' heap. Separation logic has a restricted form of quantification over heaps by the modalities introduced earlier:

- $\blacksquare\phi$ holds in a heap h if ϕ holds in *every* heap regardless of h,
- $\blacklozenge\phi$ holds in a heap h if ϕ holds in *some* heap regardless of h.

In dyadic second-order logic one can also quantify over binary variables (variables of arity 2), but multiple different such variables can be in scope. In contrast, in the semantics of separation logic we consider formulas with respect to a single heap.

To illustrate how subtle this difference is, we extend the syntax of separation logic with the binding operator $(\downarrow R\phi)$ which binds the binary second-order variable R in the evaluation of ϕ to the current interpretation of the points-to relation. The binding operator acts in a similar way as a quantifier: R is bound in ϕ in the formula $(\downarrow R\phi)$. For this extension, we need to extend the language of separation logic, in the sense that it is allowed to apply binary variables to individual variables to form primitive formulas, as in second-order logic. Note that this extension of the language still lacks quantification over second-order variables. Further, we need to adapt the semantics of relational separation logic, which we call **FRSL**$\downarrow$, since now valuations include assignments of relations to binary second-order variables.

We then give the following semantics to the binding operator:

- $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}\downarrow} (\downarrow R\phi)$ if and only if $\mathfrak{A}, \mathcal{R}, \rho[R := \mathcal{R}] \models^{\mathbf{FRSL}\downarrow} \phi$.

Consider, for example, that for formulas of the form $(\downarrow R(\phi \mathrel{-\!\!*} \psi))$ in the place of the subformulas ϕ and ψ we can still refer to the (outer) heap. Specifically, in the place of ψ we can express the locations that are in the extended part of the heap: $(x \hookrightarrow -) \wedge \neg\exists y R(x, y)$ holds for those locations x in the domain of the heap with which ϕ was evaluated, but were not allocated in the outer heap.

Alternatively, if we would translate this extended language to dyadic second order logic (as in Definition 2.4.4), we would let the binding operator correspond to bounded (second-order) quantification

$$[\downarrow R\phi](R') = \exists R((R \equiv \hookrightarrow) \wedge [\phi](R')),$$

where R and R' are different variables, and $(R \equiv \hookrightarrow)$ abbreviates the first-order formula $\forall x, y(R(x, y) \leftrightarrow (x \hookrightarrow y))$ that ensures the valuation of R coincides with the current interpretation of the relation. Here we see why introducing relational separation logic is useful, since in dyadic second-order logic binary variables range over *arbitrary* relations. Note that, just like quantifiers, it is possible to perform variable renaming of the bound variable R in case it clashes with the chosen variable R' used for translation, but we leave these technical details to the reader.

The expressive power of this binding operator lies in that it allows to 'break the spell' of the local perspective since the bound second-order variable allows, in the local context of the current interpretation of the points-to relation, to refer to the 'outer' interpretations that have generated it (by the separating connectives).

This extension of separation logic consequently allows for a simple, compositional translation of dyadic second-order logic. For notational convenience, let $(\forall R\phi)$ denote the (extended) separation logic formula $\blacksquare(\downarrow R\phi)$. Now we have that

$$\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}\downarrow} \forall R\phi$$

if and only if

$$\mathfrak{A}, \mathcal{R}, \rho[R := \mathcal{R}'] \models^{\mathbf{FRSL}\downarrow} \phi,$$

for every relation $\mathcal{R}'$. To translate every dyadic second-order formula into a corresponding formula of separation logic, we first translate it into separation logic (extended with a binding operator). Let ϕ be a dyadic second-order formula (which is assumed not to contain occurrences of the points-to relation of separation logic). We then define $T(\phi)$ by induction on the structure of the dyadic second-order formula ϕ:

- $T(\bot) = \bot$,
- $T(R(x_1, x_2)) = R(x_1, x_2)$,
- $T(C(x_1, \ldots, x_n)) = C(x_1, \ldots, x_n)$,
- $T(\phi \to \psi) = T(\phi) \to T(\psi)$,
- $T(\forall x(\phi)) = \forall x T(\phi)$,
- $T(\forall R(\phi)) = \forall R(T(\phi))$.

Note that in the last clause we use our abbreviation introduced above. It now follows that $\mathfrak{A}, \rho \models^{\mathbf{CL}} \phi$ if and only if $\mathfrak{A}, \mathcal{R}, \rho \models^{\mathbf{FRSL}\downarrow} T(\phi)$ for an arbitrary binary relation $\mathcal{R}$. Note that $T(\phi)$ does not depend on the interpretation of the points-to relation. The resulting formula $T(\phi)$ is interpreted with respect to relational separation logic, but that can again be embedded in full (functional) separation logic by the introduction of the two sorts as mentioned above.

We end our analysis by stating two open problems:

Problem 2.2. *Is the binding operator $(\downarrow R\phi)$ definable by a standard formula of separation logic in the semantics* **FRSL**$\downarrow$*?*

Problem 2.3. *Are* **FRSL** *and* **FRSL**$\downarrow$ *equally expressive?*

If the answers to both questions are affirmative, then it is possible to express the binding operator in **FRSL** too. We conjecture that this is not possible: in the first problem we have in the extended language and semantics **FRSL**$\downarrow$ additional second-order variables, whereas in **FRSL** we only have first-order variables available. In the second problem, the binder breaks the 'local perspective' of separation logic. It may be possible, however, to express the binding operator relative to a sufficiently rich structure that allows encoding heaps as objects in the domain of the underlying structure, but the details remain to be worked out.

Bibliographic note

Remarkably, [35] presents a rather intricate encoding of (dyadic) weak second-order logic into standard separation logic. Apparently this restriction to finite heaps allows to break the local perspective. Our conjecture, however, is that full separation logic is strictly less expressive than (dyadic) second-order logic.