Completeness for a First-order Abstract Separation Logic

Zhé Hóu and Alwen Tiu

Nanyang Technological University, Singapore
`zhe.hou@ntu.edu.sg, atiu@ntu.edu.sg`

Abstract. Existing work on theorem proving for the assertion language of separation logic (SL) either focuses on abstract semantics which are not readily available in most applications of program verification, or on concrete models for which completeness is not possible. An important element in concrete SL is the points-to predicate which denotes a singleton heap. SL with the points-to predicate has been shown to be non-recursively enumerable. In this paper, we develop a first-order SL, called FOASL, with an abstracted version of the points-to predicate. We prove that FOASL is sound and complete with respect to an abstract semantics, of which the standard SL semantics is an instance. We also show that some reasoning principles involving the points-to predicate can be approximated as FOASL theories, thus allowing our logic to be used for reasoning about concrete program verification problems. We give some example theories that are sound with respect to different variants of separation logics from the literature, including those that are incompatible with Reynolds's semantics. In the experiment we demonstrate our FOASL based theorem prover which is able to handle a large fragment of separation logic with heap semantics as well as non-standard semantics.

1 Introduction

Separation Logic (SL) is widely used in program verification and reasoning about memory models [29, 31]. SL extends the traditional Hoare Logic with logical connectives $*, -\!*$ from the logic of Bunched Implications (BI). These new connectives in BI provide an elegant way to reason about resources locally, enabling analyses of large scale programs. Current work on SL can be divided into two categories: one on the abstract separation logics and the other on the concrete ones. On the abstract side there has been study on BI and its Boolean variant BBI [28, 23]. Closely related are abstract separation logic and its neighbours [9, 22]. Abstract separation logics lack the interpretation of the points-to predicate $\mapsto$, which represents a single memory cell. In this setting, the semantics is just an extension of commutative monoids with certain properties usually called separation theory [13, 8]. On the concrete side there have been developments along several directions, such as proof methods for SL with memory model semantics [15, 17, 25], and symbolic heaps [2, 1, 27, 6]. There have been numerous modifications of SL, e.g., Fictional Separation Logic [19], Rely/Guarantee [35], Concurrent Separation Logic [4].

To support reasoning about Hoare triples, it is essential to have proof methods for the assertion logic. In the reminder of this paper we focus on the assertion logic of separation logic. Although theorem proving for propositional abstract separation logics (PASLs) is undecidable [23, 7], there have been semi-decision procedures for those logics [5, 30, 16, 8, 24]. However, since PASLs do not impose a concrete semantic model, they usually cannot be directly used in program verification. The situation is more intriguing for separation logic with memory model semantics. Calcagno et al. showed that the full logic is not recursively enumerable [10], thus it is not possible to have a sound and complete finite proof system for the full separation logic. Interestingly, their result uses only an extension of first-order logic, without function symbols, but with a two-field points-to predicate, i.e., predicates of the form $[a \mapsto c, d]$, which represents a memory cell with a record of two fields. Recent study also shows that the points-to predicate $\mapsto$ is a source of undecidability. For example, restricting the record field (i.e., right hand side) of $\mapsto$ to one variable and allowing only one quantified variable reduces the satisfiability of SL to PSPACE-complete [12]. The above work indicates that directly handling the points-to predicate in the logic may not be easy. Section 4 of [17] details related issues on various proof systems for separation logic with memory model. Another complicating factor in designing a proof system for separation logic is the "magic wand" $-\!*$ connective. The separation conjunction $*$ can be encoded using $-\!*$, but not the other way around [3, 11]. Consequently, most proof methods for SL with concrete semantics are restricted to fragments of SL, typically omitting $-\!*$. The connective $-\!*$, however, has found many applications, such as tail-recursion [26], iterators [20], "septraction" in rely/guarantee [35], amongst other motivating examples discussed in the introduction of [25].

Since completeness with respect to Reynolds's model is not possible, an interesting question arises as to what properties of points-to ($-\!*$ is not so crucial in terms of the completeness property) one should formalize in the system, and what kind of semantics the resulting proof system captures. There have been at least a couple of attempts at designing a proof system that features both $-\!*$ and points-to; one by Lee and Park [25] and the other by Hou et al. [17]. In [25], Lee and Park claimed incorrectly that their proof system is complete with respect to Reynolds's semantics, though this claim was later retracted.[1] In [17], Hou et al. give a proof system LS_{SL} that is sound with respect to Reynolds's semantics, but no alternative semantics was proposed nor any completeness result stated. It is also not clear in general what proof rules one should include in the proof system such as that in [17]. This has led to the introduction of various ad hoc proof rules in [17], which are often complex and unintuitive, in order to capture specific properties of the underlying concrete separation logic models (e.g., Reynolds's model), resulting in a complex proof system, which is both hard to implement and hard to reason about.

[1] See `http://pl.postech.ac.kr/SL/` for the revised version of their proof system, which is sound but not complete w.r.t. Reynolds's semantics.

In this paper, we revisit our previous work [17] in order to provide an abstract semantics and a sound and complete proof system with respect to that abstract semantics, that are useful for reasoning about the meta-theory of the proof system, and easy to extend to support reasoning about various concrete models of separation logic. Our point of departure is to try to give a minimal proof system and encode as many properties of points-to as possible using *logical theories*, rather than proof rules, and to formalize as inference rules only those properties that cannot be easily encoded as theories. This led us to keep only two proof rules for the points-to predicate from [17] (see the rules $\rightsquigarrow_1$ and $\rightsquigarrow_2$ in Figure 1 in Section 3). Semantically, these two rules are justified by a new semantics of an abstract version of the points-to predicate (notationally represented by $\rightsquigarrow$ here, to distinguish it from the points-to predicate $\mapsto$ in the concrete SL), that is, it is a function that constructs a heap from a tuple of values. In particular, we do not assume that the constructed heap from a tuple is a singleton heap, so a points-to predicate such as $[a \rightsquigarrow b, c]$ in our semantics denotes a heap, but not necessarily a singleton heap mapping a to a two-field record (b, c). Reasoning in the concrete models, such as Reynolds's SL, which may require properties of singleton heaps, can be approximated by adding theories to restrict the interpretation of points-to (see e.g., Section 5.1). Obviously one would not be able to completely restrict the interpretation of $\rightsquigarrow$ to singleton heaps via a finite theory, as one would then run into the same incompleteness problem as shown in [10].

The proof system for the first-order abstract separation logic that we introduce here, called LS_{FOASL}, is based on the proof system LS_{PASL} for propositional abstract separation logic (PASL) [16], which is a labelled sequent calculus. We choose labelled calculi as the framework to formalize our logic since it has been shown to be a good framework for proof search automation for abstract separation logics [17, 24]. Formulas in a labelled sequent in LS_{PASL} are interpreted relative to the interpretation of the labels they are attached to, so a labelled formula such as $h : F$, where h is a label and F is a formula, denotes the truth value of F when evaluated against the heap h. In extending PASL to the first-order case, especially when formulating theories for specific concrete models, it turns out that we need to be able to assert that some formulas hold universally for all heaps, or that they hold in some unspecified heap, both of which are not expressible in PASL. To get around this limitation, we introduce modal operators to allow one to state properties that hold globally in all heaps or in some heap. Section 5 shows some examples of uses of these modal operators in theories approximating concrete models of separation logics.

The semantics of FOASL is formally defined in Section 2. In Section 3 we present the proof system LS_{FOASL}, which is an extension of LS_{PASL} with rules for first-order quantifiers, the points-to predicate and modal operators. In Section 4, we prove soundness and completeness of LS_{FOASL} with respect to the semantics described in Section 2. The completeness proof is done via a counter-model construction similar to that in [21, 16] for the propositional case, but our proof is significantly more complicated as we have to deal with the first-order language and the construction of the model for the points-to predicate, which requires a novel proof technique. We show in Section 5 that all the inference rules for points-to in the labelled sequent calculus LS_{SL} [17] can be derived using theories in our logic, except one rule called HC, which is not used in the current program verification tools. Our theories for points-to cover the widely-used symbolic heap fragment of SL, thus our logic can be used in many existing program verification methods. Furthermore, we can also prove many formulae that are valid in SL but which cannot be proved by most existing tools. An implementation is discussed in Section 6, we show that our prover can reason about the standard heap semantics and the non-standard ones.

2 First-order Abstract Separation Logic

This section introduces First-order Abstract Separation Logic (FOASL). The formulae of FOASL are parameterized by a first order signature $\Sigma = (\mathcal{R}, \mathcal{C})$, consisting of a set of predicate symbols $\mathcal{R}$ and a set of constants $\mathcal{C}$. Constants are ranged by a, b and c, and predicate symbols by p and q (possibly with subscripts). We also assume an infinite set $\mathcal{V}$ of first-order variables, ranged over by x, y, z. A *term* is either a constant or a variable, and is denoted by e, s and t. We assume that $\mathcal{R}$ includes a symbol $=$, denoting the equality predicate, and a finite collection of *abstract points-to* predicates. We shall use the notation $\leadsto^n$ to denote an abstract points-to predicate of arity n. We use an infix notation when writing the abstract points-to predicates. For an abstract points-to predicate of arity k, taking arguments $t_1, \ldots, t_k$, we write it in the form:

$$t_1 \leadsto^k t_2, \ldots, t_k.$$

We shall omit the superscript k when the arity of $\leadsto$ is not important or can be inferred from the context of discussion. Note that the abstract points-to $\leadsto$ is not the points-to predicate in SL, but is a weaker version whose properties will be discussed later. To simplify presentation, we do not consider function symbols in our language, but it is straightforward to add them. In any case, the incompleteness result for concrete SL of [10] holds even in the absence of function symbols, so the omission of function symbols from our logic does not make the completeness proof for our logic conceptually easier.

The formulae of FOASL are given by the following grammar:

$$\begin{aligned} F ::= {} & \top^* \mid \bot \mid p(t_1, \ldots, t_k) \mid s \leadsto t_1, \ldots, t_l \mid s = t \mid F \to F \mid \\ & F * F \mid F \mathrel{-\!\!*} F \mid \Diamond F \mid \exists x.F \end{aligned}$$

The logical constant $\bot$, the connective $\to$ and the quantifer $\exists$ are the usual logical operators from first-order logic. The operator $\Diamond$ is a modal operator (denoting "possibility"). Classical negation $\neg F$ is defined as $F \to \bot$. Other operators, i.e., $\top$, $\vee$, $\wedge$, $\forall$, and $\Box$, can be defined via classical negation, e.g., $\Box A = \Diamond(A \to \bot) \to \bot$. The connectives $*$ and $\mathrel{-\!\!*}$ correspond to separation conjunction and the "magic wand" from separation logic [31], and $\top^*$ is the multiplicative truth.

The semantics of FOASL is defined based on a *separation algebra*, i.e., a commutative monoid $(H, \circ, \epsilon)$ where H is a non-empty set, $\circ$ is a partial binary

$\mathcal{M}, v, h \Vdash p(t_1, \cdots, t_n)$ iff $(t_1^{\mathcal{M}}, \cdots, t_n^{\mathcal{M}}) \in p^I$ $\quad$ $\mathcal{M}, v, h \Vdash \top^*$ iff $h = \epsilon$
$\mathcal{M}, v, h \Vdash A \to B$ iff $\mathcal{M}, v, h \not\Vdash A$ or $\mathcal{M}, v, h \Vdash B$ $\quad$ $\mathcal{M}, v, h \Vdash \bot$ iff never
$\mathcal{M}, v, h \Vdash A * B$ iff $h_1 \circ h_2 = h$ and $\mathcal{M}, v, h_1 \Vdash A$ and $\mathcal{M}, v, h_2 \Vdash B$ for some h_1, h_2
$\mathcal{M}, v, h \Vdash A -\!\!* B$ iff for all h_1, h_2, if $h \circ h_1 = h_2$ and $\mathcal{M}, v, h_1 \Vdash A$, then $\mathcal{M}, v, h_2 \Vdash B$
$\mathcal{M}, v, h \Vdash \exists x.A(x)$ iff $\exists d \in D.\mathcal{M}, v[d/x], h \Vdash A(x)$
$\mathcal{M}, v, h \Vdash \Diamond A$ iff $\exists h_1 \in H.\mathcal{M}, v, h_1 \Vdash A$
$\mathcal{M}, v, h \Vdash t_1 = t_2$ iff $t_1^{\mathcal{M}}$ and $t_2^{\mathcal{M}}$ are the same element in D.
$\mathcal{M}, v, h \Vdash t_1 \rightsquigarrow t_2, \ldots, t_k$ iff $\mathfrak{f}_k(t_1^{\mathcal{M}}, \ldots, t_k^{\mathcal{M}}) = h$.

Table 1. The semantics of FOASL.

function $H \times H \rightharpoonup H$ written infix, and $\epsilon \in H$ is the unit. This separation algebra satisfies the following conditions, where '=' is interpreted as 'both sides undefined, or both sides defined and equal':

identity: $\forall h \in H.\, h \circ \epsilon = h$.
commutativity: $\forall h_1, h_2 \in H.\, h_1 \circ h_2 = h_2 \circ h_1$.
associativity: $\forall h_1, h_2, h_3 \in H.\, h_1 \circ (h_2 \circ h_3) = (h_1 \circ h_2) \circ h_3$.
cancellativity: $\forall h_1, h_2, h_3, h_4 \in H$. if $h_1 \circ h_2 = h_3$ and $h_1 \circ h_4 = h_3$ then $h_2 = h_4$.
indivisible unit: if $h_1 \circ h_2 = \epsilon$ then $h_1 = \epsilon$.
disjointness: $\forall h_1, h_2 \in H$. if $h_1 \circ h_1 = h_2$ then $h_1 = \epsilon$.
cross-split: if $h_1 \circ h_2 = h_0$ and $h_3 \circ h_4 = h_0$, then $\exists h_{13}, h_{14}, h_{23}, h_{24} \in H$ such that $h_{13} \circ h_{14} = h_1$, $h_{23} \circ h_{24} = h_2$, $h_{13} \circ h_{23} = h_3$, and $h_{14} \circ h_{24} = h_4$.

Note that *partial-determinism* is assumed since $\circ$ is a partial function: for any $h_1, h_2, h_3, h_4 \in H$, if $h_1 \circ h_2 = h_3$ and $h_1 \circ h_2 = h_4$ then $h_3 = h_4$.

A FOASL model is a tuple $\mathcal{M} = (D, I, v, \mathcal{F}, H, \circ, \epsilon)$ where D is a non-empty domain, I is an interpretation function mapping constant symbols to elements of D, and predicate symbols, other than $=$ and $\rightsquigarrow$, to relations. The function v is a valuation function mapping variables to D. We use a set $\mathcal{F}$ of functions to interpret the abstract points-to predicates. To each abstract points-to predicate of arity n, we associate an n-argument total function $\mathfrak{f}_n : D \times \cdots \times D \mapsto H \in \mathcal{F}$. The tuple $(H, \circ, \epsilon)$ is a separation algebra. For a predicate symbol p and a constant symbol c, we write p^I and c^I, respectively, for their interpretations under I. We write $t^{\mathcal{M}}$ for the interpretation of term t in the model $\mathcal{M}$. For a variable x, $x^{\mathcal{M}} = v(x)$. A term t is *closed* if it has no variables, and we write t^I for the interpretation of t since it is independent of the valuation function v.

A separation algebra $(H, \circ, \epsilon)$ can be seen as a Kripke frame, where H is the set of worlds and the (ternary) accessibility relation R is defined as: $R(h_1, h_2, h_3)$ iff $h_1 \circ h_2 = h_3$. Modal operators are thus a natural extension to FOASL.

The semantics of FOASL formulae are defined via Kripke style notations in Table 1, where $\mathcal{M} = (D, I, v, \mathcal{F}, H, \circ, \epsilon)$ is a FOASL model, and $h, h_1, h_2 \in H$. In the table, we write $v[c/x]$ to denote the valuation function that may differ from v only in the mapping of x, i.e., $v[c/x](x) = c$ and $v[c/x](y) = v(y)$ if $y \neq x$. A FOASL formula A is true at h in the model $\mathcal{M} = (D, I, v, \mathcal{F}, H, \circ, \epsilon)$ if $\mathcal{M}, v, h \Vdash A$. It is true in $\mathcal{M}$ if it is true at some h in H. A formula is valid if it is true in all models; a formula is satisfiable if it is true in some model. A formula is called a *sentence* if it has no free variables.

Besides the first-order language, FOASL has two main extensions over PASL: the abstract points-to predicate $\rightsquigarrow$ and the modality $\Diamond$. We explain their intuitions here. In the concrete SL models, the points-to predicate $[a \mapsto b]$ is true only in a singleton heap that maps a to b. In our abstract semantics for $[a \rightsquigarrow b]$, we drop the requirement that the heap must be singleton. Instead, we generalize this by parameterizing the semantics with a function (the function $\mathfrak{f}$ discussed earlier) that associates values to (possibily non-singleton) heaps. The predicate $[a \rightsquigarrow b]$ is true in a world h iff h is the image of $\mathfrak{f}_2(a, b)$. As a consequence of this interpretation of $\rightsquigarrow$, we have the following properties where $\vec{t}$ is a list of fields:

- *(Injectivity)* If $s \rightsquigarrow \vec{t}$ holds in both h_1 and h_2, then $h_1 = h_2$.
- *(Totality)* For any s and $\vec{t}$, there is some h such that $s \rightsquigarrow \vec{t}$ holds in h.

The latter in particular is a consequence of the fact that functions in $\mathcal{F}$ are total functions. We do not impose any other properties on $\rightsquigarrow$. For example, we do not assume an invalid address *nil* such that $nil \mapsto \vec{t}$ must be false. The reason we cannot disprove $nil \rightsquigarrow \vec{t}$ is partly because we do not insist on a fixed interpretation of *nil* in our logic. This *does not* mean that $nil \rightsquigarrow \vec{t}$ is valid in our logic; it is only satisfiable. We can strengthen $\rightsquigarrow$ by adding more theories to it, including a formula to force $nil \rightsquigarrow \vec{t}$ to be unsatisfiable. See Section 5 and 6 for details.

To motivate the need for the modal operators, consider an example to approximate, in our framework, a separation logic where the points-to relation maps an address to a multiset of addresses. In the binary case, one could formalize this as:

$$F = \forall x, y, z.(x \mapsto y, z) \to (x \mapsto z, y)$$

We can encode this property as a rule in a labelled sequent calculus as shown below left. When generalising this to points-to of $(n + 1)$-arities, we will have to consider adding many variants of rules that permute the right hand side of $\mapsto$, which is what we are trying to avoid. Alternatively, we can add the formula F to the antecedent of the sequent, and attach a label l to F, and do a forward-chaining on $l : F$, as shown below right, where Γ, Δ are sets of labelled formulae:

$$\frac{\Gamma; l : (a \mapsto c, b) \vdash \Delta}{\Gamma; l : (a \mapsto b, c) \vdash \Delta} \qquad \frac{\cdots \qquad \Gamma, l : F, l : (a \mapsto c, b) \vdash \Delta}{\Gamma, l : F, l : (a \mapsto b, c) \vdash \Delta}$$

where the $\cdots$ are the instantiation of x, y, z with a, b, c respectively, and the discharge of the assumption $(x \mapsto y, z)$ of F. However, if $(a \mapsto b, c)$ in the conclusion is attached to another label (world) m, we then have to add $m : F$ to the sequent. In effect, we would have to add an infinite set of labelled formulae of the form $k : F$ to the sequent to achieve the same effect of the inference rule. With modalities, we can simply use $l : \Box F$, which would then allow F to be used at any world in the antecedent of the sequent.

Example 1. Consider Reynolds's semantics for separation logic [31], with an abstract points-to predicate of arity two. This can be shown to be an instance of our abstract semantics, where the domain D is the set of integers, H is the set of heaps (i.e., finite partial maps from integers to integers), ϵ denotes the empty heap, and the function $\mathfrak{f}_2$ is defined as $\mathfrak{f}_2(a,b) = [a \mapsto b]$ where $[a \mapsto b]$ is the singleton heap, mapping a to b. The operation $\circ$ on H is defined as heap composition. It can be shown that $(H, \circ, \epsilon)$ forms a separation algebra. Note that if we relax the interpretation of H to allow infinite heaps, $(H, \circ, \epsilon)$ is still a separation algebra, which shows that our semantics may admit non-standard interpretations of separation logic.

3 LS_{FOASL}: A Labelled Calculus for FOASL

Let LVar be an infinite set of *label variables*, the set $\mathcal{L}$ of *labels* is LVar $\cup \{\epsilon\}$, where $\epsilon \notin$ LVar is a label constant. We overload the notation and write h with subscripts as labels. A function $\rho : \mathcal{L} \to H$ from labels to worlds is a *label mapping* iff it satisfies $\rho(\epsilon) = \epsilon$, mapping the label constant ϵ to the identity world of H. A *labelled formula* is a pair consisting of a label and a formula. We write a labelled formula as $h : A$, when h is the label and A is the formula of the labelled formula. A *relational atom* is an expression of the form $(h_1, h_2 \triangleright h_3)$, where h_1, h_2 and h_3 are labels, this corresponds to $h_1 \circ h_2 = h_3$ in the semantics. A relational atom is not a formula; rather it can be thought of as a structural component of a sequent. A *sequent* takes the form $\mathcal{G}; \Gamma \vdash \Delta$ where $\mathcal{G}$ is a set of relational atoms, Γ, Δ are sets of labelled formulae, and ; denotes set union. Thus $\Gamma; h : A$ is the union of Γ and $\{h : A\}$. The left hand side of a sequent is the *antecedent* and the right hand side is the *succedent*.

We call our labelled proof system LS_{FOASL}. The logical rules of LS_{FOASL} are shown in Figure 1, structural rules are in Figure 2. To simplify some rules, we introduce the notation $h_1 \sim h_2$ as an abbreviation of $(\epsilon, h_1 \triangleright h_2)$. We use the notation $[t/x]$ to denote a variable substitution, and similarly $[h'/h]$ for a label substitution, where h is a label variable. The equality rules, for terms ($=_1$ and $=_2$) and labels ($\sim_1$ and $\sim_2$), are the usual equality rules (see e.g., [34]). These rules allow one to replace a term (label) with its equal anywhere in the sequent. Note that in those rules, the replacement of terms (labels) need not be done for all occurrences of equal terms; one can replace just one occurrence or more. For example, below left is a valid instance of $=_2$. This is because both the premise and the conclusion of the rules are instances of the sequent below right:

$$\frac{h : s = t; h_1 : p(t,s) \vdash h_2 : q(s,s)}{h : s = t; h_1 : p(s,s) \vdash h_2 : q(s,s)}\ {=_2} \qquad h : s = t; h_1 : p(x,s) \vdash h_2 : q(s,s)$$

i.e., the premise sequent is obtained from the above sequent with substitution $[t/x]$, and the conclusion sequent with $[s/x]$. A similar remark applies for label replacements in sequents affected via $\sim_2$. The rules $\rightsquigarrow_1$ and $\rightsquigarrow_2$ respectively capture the injectivity and the totality properties of the underlying semantic function interpreting $\rightsquigarrow$.

An *extended model* $(\mathcal{M}, \rho)$ is a FOASL model $\mathcal{M}$ equipped with a label mapping ρ. A sequent $\mathcal{G}; \Gamma \vdash \Delta$ is *falsifiable* in an extended model if: (1) every

$$\frac{}{\mathcal{G};\Gamma;h:A\vdash h:A;\Delta}\;id \qquad \frac{}{\mathcal{G};\Gamma;h:\bot\vdash\Delta}\;\bot L$$

$$\frac{\mathcal{G};h\sim\epsilon;\Gamma\vdash\Delta}{\mathcal{G};\Gamma;h:\top^*\vdash\Delta}\;\top^* L \qquad \frac{}{\mathcal{G};\Gamma\vdash\epsilon:\top^*;\Delta}\;\top^* R$$

$$\frac{\mathcal{G};\Gamma;h:A\vdash h:B;\Delta}{\mathcal{G};\Gamma\vdash h:A\to B;\Delta}\;\to R \qquad \frac{\mathcal{G};\Gamma\vdash h:A;\Delta \qquad \mathcal{G};\Gamma;h:B\vdash\Delta}{\mathcal{G};\Gamma;h:A\to B\vdash\Delta}\;\to L$$

$$\frac{(h_1,h_2\triangleright h_0);\mathcal{G};\Gamma;h_1:A;h_2:B\vdash\Delta}{\mathcal{G};\Gamma;h_0:A*B\vdash\Delta}\;*L \qquad \frac{(h_1,h_0\triangleright h_2);\mathcal{G};\Gamma;h_1:A\vdash h_2:B;\Delta}{\mathcal{G};\Gamma\vdash h_0:A\mathrel{-\!\!*}B;\Delta}\;\mathrel{-\!\!*}R$$

$$\frac{(h_1,h_2\triangleright h_0);\mathcal{G};\Gamma\vdash h_1:A;h_0:A*B;\Delta \qquad (h_1,h_2\triangleright h_0);\mathcal{G};\Gamma\vdash h_2:B;h_0:A*B;\Delta}{(h_1,h_2\triangleright h_0);\mathcal{G};\Gamma\vdash h_0:A*B;\Delta}\;*R$$

$$\frac{(h_1,h_0\triangleright h_2);\mathcal{G};\Gamma;h_0:A\mathrel{-\!\!*}B\vdash h_1:A;\Delta \qquad (h_1,h_0\triangleright h_2);\mathcal{G};\Gamma;h_0:A\mathrel{-\!\!*}B;h_2:B\vdash\Delta}{(h_1,h_0\triangleright h_2);\mathcal{G};\Gamma;h_0:A\mathrel{-\!\!*}B\vdash\Delta}\;\mathrel{-\!\!*}L$$

$$\frac{\mathcal{G};\Gamma;h:A(y)\vdash\Delta}{\mathcal{G};\Gamma;h:\exists x.A(x)\vdash\Delta}\;\exists L \qquad \frac{\mathcal{G};\Gamma\vdash h:A(t);h:\exists x.A(x);\Delta}{\mathcal{G};\Gamma\vdash h:\exists x.A(x);\Delta}\;\exists R \qquad \frac{\mathcal{G};\Gamma;h':A\vdash\Delta}{\mathcal{G};\Gamma;h:\Diamond A\vdash\Delta}\;\Diamond L$$

$$\frac{\mathcal{G};\Gamma;h:t=t\vdash\Delta}{\mathcal{G};\Gamma\vdash\Delta}\;=_1 \qquad \frac{\mathcal{G};h:s=t;\Gamma[t/x]\vdash\Delta[t/x]}{\mathcal{G};h:s=t;\Gamma[s/x]\vdash\Delta[s/x]}\;=_2 \qquad \frac{\mathcal{G};\Gamma\vdash h':A;h:\Diamond A;\Delta}{\mathcal{G};\Gamma\vdash h:\Diamond A;\Delta}\;\Diamond R$$

$$\frac{\mathcal{G};\Gamma;h:s\leadsto\vec{t}\vdash\Delta}{\mathcal{G};\Gamma\vdash\Delta}\;\leadsto_1 \qquad \frac{\mathcal{G};h_1\sim h_2;\Gamma;h_1:s\leadsto\vec{t};h_2:s\leadsto\vec{t}\vdash\Delta}{\mathcal{G};\Gamma;h_1:s\leadsto\vec{t};h_2:s\leadsto\vec{t}\vdash\Delta}\;\leadsto_2$$

Side conditions:
In $*L$ and $\mathrel{-\!\!*}R$, the labels h_1 and h_2 do not occur in the conclusion.
In $\exists L$, y is not free in the conclusion. In $\Diamond L$, h' does not occur in the conclusion.
In $\leadsto_1$, h does not occur in the conclusion.

Fig. 1. Logical rules in LS_{FOASL}.

relational atom $(h_1,h_2\triangleright h_3)\in\mathcal{G}$ is true, i.e., $\rho(h_1)\circ\rho(h_2)=\rho(h_3)$; (2) every labelled formula $h:A\in\Gamma$ is true, i.e., $\mathcal{M},v,\rho(h)\Vdash A$; (3) every labelled formula $h':B\in\Delta$ is false, i.e., $\mathcal{M},v,\rho(h')\not\Vdash B$. A sequent is falsifiable if it is falsifiable in some extended model.

To prove a formula F, we start from the sequent $\vdash h:F$ with an arbitrary label $h\neq\epsilon$, and try to derive a closed derivation by applying inference rules backwards from this sequent. A derivation is closed if every branch can be closed by a rule with no premises. The soundness of LS_{FOASL} can be proved by arguing that each rule preserves falsifiability upwards. The proof is given in [18].

Theorem 1 (Soundness). *For every FOASL formula F, if $\vdash h:F$ is derivable in LS_{FOASL} for any label h, then F is a valid FOASL formula.*

4 Counter-model Construction

We now give a counter-model construction for LS_{FOASL} to show that LS_{FOASL} is complete w.r.t. FOASL. The proof here is motivated by the completeness proof of the labelled sequent calculus and labelled tableaux for PASL [16, 21], but this

$$\frac{h \sim h; \mathcal{G}; \Gamma \vdash \Delta}{\mathcal{G}; \Gamma \vdash \Delta} \sim_1 \qquad \frac{h_1 \sim h_2; \mathcal{G}[h_2/h]; \Gamma[h_2/h] \vdash \Delta[h_2/h]}{h_1 \sim h_2; \mathcal{G}[h_1/h]; \Gamma[h_1/h] \vdash \Delta[h_1/h]} \sim_2$$

$$\frac{(h_2, h_1 \triangleright h_0); (h_1, h_2 \triangleright h_0); \mathcal{G}; \Gamma \vdash \Delta}{(h_1, h_2 \triangleright h_0); \mathcal{G}; \Gamma \vdash \Delta} E \qquad \frac{(h_1, h_1 \triangleright h_2); h_1 \sim \epsilon; \mathcal{G}; \Gamma \vdash \Delta}{(h_1, h_1 \triangleright h_2); \mathcal{G}; \Gamma \vdash \Delta} D$$

$$\frac{(h_3, h_5 \triangleright h_0); (h_2, h_4 \triangleright h_5); (h_1, h_2 \triangleright h_0); (h_3, h_4 \triangleright h_1); \mathcal{G}; \Gamma \vdash \Delta}{(h_1, h_2 \triangleright h_0); (h_3, h_4 \triangleright h_1); \mathcal{G}; \Gamma \vdash \Delta} A$$

$$\frac{(h_1, h_2 \triangleright h_0); h_0 \sim h_3; \mathcal{G}; \Gamma \vdash \Delta}{(h_1, h_2 \triangleright h_0); (h_1, h_2 \triangleright h_3); \mathcal{G}; \Gamma \vdash \Delta} P \qquad \frac{(h_1, h_2 \triangleright h_0); h_2 \sim h_3; \mathcal{G}; \Gamma \vdash \Delta}{(h_1, h_2 \triangleright h_0); (h_1, h_3 \triangleright h_0); \mathcal{G}; \Gamma \vdash \Delta} C$$

$$\frac{(h_5, h_6 \triangleright h_1); (h_7, h_8 \triangleright h_2); (h_5, h_7 \triangleright h_3); (h_6, h_8 \triangleright h_4); (h_1, h_2 \triangleright h_0); (h_3, h_4 \triangleright h_0); \mathcal{G}; \Gamma \vdash \Delta}{(h_1, h_2 \triangleright h_0); (h_3, h_4 \triangleright h_0); \mathcal{G}; \Gamma \vdash \Delta} CS$$

Side conditions:
In A, the label h_5 does not occur in the conclusion.
In CS, the labels h_5, h_6, h_7, h_8 do not occur in the conclusion.

Fig. 2. Structural rules in LS_{FOASL}.

proof is significantly more complex, as can be seen in the definition of Hintikka sequent below, which has almost twice as many cases as the previous work. The constructed model extends the non-classical logic model in the previous work with a Herbrand model as in first-order logic. For space reasons we only set up the stage here and give the full proofs in [18].

We define a notion of *saturated sequent*, i.e., *Hintikka sequent*, on which all possible rule instances in LS_{FOASL} have been applied. In the following, we denote with R a relational atom or a labelled formula.

Definition 1 (Hintikka sequent). *Let L be a FOASL language and let T be the set of closed terms in L. A labelled sequent $\mathcal{G}; \Gamma \vdash \Delta$, where Γ, Δ are sets of labelled sentences, is a* Hintikka sequent *w.r.t. L if it satisfies the following conditions for any sentences A, B, any terms t, t', and any labels $h, h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7$:*

1. *If $h_1 : A \in \Gamma$ and $h_2 : A \in \Delta$ then $h_1 \sim h_2 \notin \mathcal{G}$.*
2. *$h : \bot \notin \Gamma$.*
3. *If $h : \top^* \in \Gamma$ then $h \sim \epsilon \in \mathcal{G}$.*
4. *If $h : \top^* \in \Delta$ then $h \sim \epsilon \notin \mathcal{G}$.*
5. *If $h : A \to B \in \Gamma$ then $h : A \in \Delta$ or $h : B \in \Gamma$.*
6. *If $h : A \to B \in \Delta$ then $h : A \in \Gamma$ and $h : B \in \Delta$.*
7. *If $h_0 : A * B \in \Gamma$ then $\exists h_1, h_2 \in \mathcal{L}$ s.t. $(h_1, h_2 \triangleright h_0) \in \mathcal{G}$, $h_1 : A \in \Gamma$ and $h_2 : B \in \Gamma$.*
8. *If $h_3 : A * B \in \Delta$ then $\forall h_0, h_1, h_2 \in \mathcal{L}$ if $(h_1, h_2 \triangleright h_0) \in \mathcal{G}$ and $h_0 \sim h_3 \in \mathcal{G}$ then $h_1 : A \in \Delta$ or $h_2 : B \in \Delta$.*
9. *If $h_3 : A \mathrel{-\!\!*} B \in \Gamma$ then $\forall h_0, h_1, h_2 \in \mathcal{L}$ if $(h_1, h_2 \triangleright h_0) \in \mathcal{G}$ and $h_2 \sim h_3 \in \mathcal{G}$, then $h_1 : A \in \Delta$ or $h_0 : B \in \Gamma$.*
10. *If $h_2 : A \mathrel{-\!\!*} B \in \Delta$ then $\exists h_0, h_1 \in \mathcal{L}$ s.t. $(h_1, h_2 \triangleright h_0) \in \mathcal{G}$, $h_1 : A \in \Gamma$ and $h_0 : B \in \Delta$.*
11. *If $h : \exists x.A(x) \in \Gamma$ then $h : A(t) \in \Gamma$ for some $t \in T$.*
12. *If $h : \exists x.A(x) \in \Delta$ then $h : A(t) \in \Delta$ for every $t \in T$.*
13. *If $h : \Diamond A \in \Gamma$ then $\exists h_1 \in \mathcal{L}$ s.t. $h_1 : A \in \Gamma$.*

14. *If $h : \Diamond A \in \Delta$ then $\forall h_1 \in \mathcal{L}$, $h_1 : A \in \Delta$.*
15. *For any $t \in T$, $\exists h \in \mathcal{L}$ s.t. $h : t = t \in \Gamma$.*
16. *If $h_1 : t = t' \in \Gamma$ and $h_2 : A[t/x] \in \Gamma$ ($h_2 : A[t/x] \in \Delta$) then $h_2 : A[t'/x] \in \Gamma$ (resp. $h_2 : A[t'/x] \in \Delta$).*
17. *For any label $h \in \mathcal{L}$, $h \sim h \in \mathcal{G}$.*
18. *If $h_1 \sim h_2 \in \mathcal{G}$ and a relational atom or a labelled formula $R[h_1/h] \in \mathcal{G} \cup \Gamma$ (resp. $R[h_1/h] \in \Delta$), then $R[h_2/h] \in \mathcal{G} \cup \Gamma$ (resp. $R[h_2/h] \in \Delta$).*
19. *If $(h_1, h_2 \triangleright h_0) \in \mathcal{G}$ then $(h_2, h_1 \triangleright h_0) \in \mathcal{G}$.*
20. *If $\{(h_1, h_2 \triangleright h_0); (h_3, h_4 \triangleright h_6); h_1 \sim h_6\} \subseteq \mathcal{G}$ then $\exists h_5 \in \mathcal{L}$. $\{(h_3, h_5 \triangleright h_0), (h_2, h_4 \triangleright h_5)\} \subseteq \mathcal{G}$.*
21. *If $\{(h_1, h_2 \triangleright h_0); (h_3, h_4 \triangleright h_9); h_0 \sim h_9\} \subseteq \mathcal{G}$ then $\exists h_5, h_6, h_7, h_8 \in \mathcal{L}$ s.t. $\{(h_5, h_6 \triangleright h_1), (h_7, h_8 \triangleright h_2), (h_5, h_7 \triangleright h_3), (h_6, h_8 \triangleright h_4)\} \subseteq \mathcal{G}$.*
22. *For every abstract points-to predicate $\rightsquigarrow^k$ in the language and for any $t_1, \dots, t'_k \in T$, $\exists h \in \mathcal{L}$ s.t. $h : t_1 \rightsquigarrow^k t_2, \dots, t_k \in \Gamma$.*
23. *If $\{h_1 : s \rightsquigarrow \vec{t}, h_2 : s \rightsquigarrow \vec{t}\} \subseteq \Gamma$ then $h_1 \sim h_2 \in \mathcal{G}$.*
24. *If $\{(h_1, h_3 \triangleright h_2), h_1 \sim h_3\} \subseteq \mathcal{G}$ then $h_1 \sim \epsilon \in \mathcal{G}$.*
25. *If $\{(h_1, h_2 \triangleright h_0), (h_4, h_5 \triangleright h_3), h_1 \sim h_4, h_2 \sim h_5\} \subseteq \mathcal{G}$ then $h_0 \sim h_3 \in \mathcal{G}$.*
26. *If $\{(h_1, h_2 \triangleright h_0), (h_4, h_5 \triangleright h_3), h_1 \sim h_4, h_0 \sim h_3\} \subseteq \mathcal{G}$ then $h_2 \sim h_5 \in \mathcal{G}$.*

The next lemma shows that we can build an extended FOASL model $(\mathcal{M}, \rho)$ where $\mathcal{M} = (D, I, v, \mathcal{F}, H, \circ, \epsilon)$ that falsifies the Hintikka sequent $\mathcal{G}; \Gamma \vdash \Delta$. The D, I part is a Herbrand model as in first-order logic. The construction of the monoid $(H, \circ, \epsilon)$ is similar to the one for PASL [16], where H is the set of equivalent classes of labels in the sequent. The interpretation of the predicate $\rightsquigarrow$ is defined based the set of functions $\mathcal{F}$. For each n-ary predicate $\rightsquigarrow^n$, there is a function $\mathfrak{f}_n \in \mathcal{F}$ defined as below:

$$\mathfrak{f}_n(t_1, \cdots, t_n) = [h]_{\mathcal{G}} \text{ iff } h' : t_1 \rightsquigarrow^n t_2, \cdots, t_n \in \Gamma \text{ and } h \sim h' \in \mathcal{G}.$$

where $[h]_{\mathcal{G}}$ is the class of labels equivalent to h in $\mathcal{G}$. $\mathcal{F}$ is the set of all such functions. By Condition 22 and 23 of the Hintikka sequent, each function in $\mathcal{F}$ must be a total function. The full proof is in [18].

Lemma 1 (Hintikka's Lemma). *Suppose L is a FOASL language with a non-empty set of closed terms. Every Hintikka sequent w.r.t. L is falsifiable.*

Then we show how to construct a Hintikka sequent for an unprovable formula using the proof system LS_{FOASL}. Unlike the usual procedure, we have to consider the rules with no (or more than one) principal formulae. To this end, we define a notion of *extended formulae* as in the previous work [16]:

$$ExF ::= F \mid \equiv_1 \mid \equiv_2 \mid \mapsto_1 \mid \mapsto_2 \mid \mathbb{E} \mid \mathbb{A} \mid \mathbb{CS} \mid \approx_1 \mid \approx_2 \mid \mathbb{P} \mid \mathbb{C} \mid \mathbb{D}$$

Here, F is a FOASL formula, the other symbols correspond to the special rules in LS_{FOASL}. For example, $\equiv_1$ and $\equiv_2$ correspond to rules $=_1$ and $=_2$; $\mapsto_1$ and $\mapsto_2$ correspond to $\rightsquigarrow_1$ and $\rightsquigarrow_2$; $\approx_1$ and $\approx_2$ correspond to $\sim_1$ and $\sim_2$. The saturation procedure is performed according to a schedule, which is defined below.

Definition 2 (Schedule). *A* rule instance *is a tuple (O, h, ExF, R, S, n), where O is either 0 (left) or 1 (right), h is a label, ExF is an extended formula, R is a set of relational atoms such that $|R| \leq 2$, S is a set of labelled formulae with $|S| \leq 2$, and n is a natural number. Let $\mathcal{I}$ denote the set of all rule instances. A* schedule *is a function from natural numbers $\mathbb{N}$ to $\mathcal{I}$. A schedule ϕ is* fair *if for every rule instance I, the set $\{i \mid \phi(i) = I\}$ is infinite.*

It is easy to verify that a fair schedule must exist. This is proved by checking that $\mathcal{I}$ is a countable set [21], which follows from the fact that $\mathcal{I}$ is a finite product of countable sets. We fix a fair schedule ϕ for the following proofs. We assume the set $\mathcal{L}$ of labels is totally ordered and can be enumerated as $h_0, h_1, h_2, \cdots$, where $h_0 = \epsilon$. Similarly, we assume an infinite set of closed terms which can be enumerated as $t_0, t_1, t_2, \cdots$, all of which are disjoint from the terms in F. Suppose F is an unprovable formula, we start from the sequent $\vdash h_1 : F$ and construct an underivable sequent as below.

Definition 3. *Let F be a formula which is not provable in LS_{FOASL}. We assume that every variable in F is bounded, otherwise we can rewrite F so that unbounded variables are universally quantified. We construct a series of finite sequents $\langle \mathcal{G}_i; \Gamma_i \vdash \Delta_i \rangle_{i \in \mathcal{N}}$ from F where $\mathcal{G}_1 = \Gamma_1 = \emptyset$ and $\Delta_1 = a_1 : F$. Suppose $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$ has been defined, we define $\mathcal{G}_{i+1}; \Gamma_{i+1} \vdash \Delta_{i+1}$ in the sequel. Suppose $\phi(i) = (O_i, h_i, ExF_i, R_i, S_i, n_i)$. When we use n_i to select a term (resp. label) in a formula (resp. relational atom), we assume the terms (resp. labels) are ordered from left to right. If n_i is greater than the number of terms in the formula (labels in the relational atom), then no effect is taken. We only show a few cases here, and display this rather involved construction in the Appendix of [18].*

- *If $O_i = 0$, ExF_i is a FOASL formula $C_i = F_1 * F_2$ and $h_i : C_i \in \Gamma_i$, then $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(h_{4i}, h_{4i+1} \triangleright h_i)\}$, $\Gamma_{i+1} = \Gamma_i \cup \{h_{4i} : F_1, h_{4i+1} : F_2\}$, $\Delta_{i+1} = \Delta_i$.*
- *If ExF_i is $\equiv_1$ and $S_i = \{h_i : t_n = t_n\}$, where $n \leq i + 1$, then $\mathcal{G}_{i+1} = \mathcal{G}_i$, $\Gamma_{i+1} = \Gamma_i \cup \{h_i : t_n = t_n\}$, and $\Delta_{i+1} = \Delta_i$.*
- *If ExF_i is $\equiv_2$ and $S_i = \{h : t = t', h' : A[t/x]\} \subseteq \Gamma_i$, where x is the n_ith term in A, then $\mathcal{G}_{i+1} = \mathcal{G}_i$, $\Gamma_{i+1} = \Gamma_i \cup \{h' : A[t'/x]\}$, and $\Delta_{i+1} = \Delta_i$.*
- *If ExF_i is $\equiv_2$ and $S_i = \{h : t = t', h' : A[t/x]\}$ where $h : t = t' \in \Gamma_i$, $h' : A[t/x] \in \Delta_i$, and x is the n_ith term in A. Then $\mathcal{G}_{i+1} = \mathcal{G}_i$, $\Gamma_{i+1} = \Gamma_i$, and $\Delta_{i+1} = \Delta_i \cup \{h' : A[t'/x]\}$.*

The first rule shows how to use the $*L$ rule and how to deal with fresh variables. The indexing of labels guarantees that the choice of h_{4i}, h_{4i+1}, h_{4i+2}, h_{4i+3} are always fresh for the sequent $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$. Similarly, the term t_{i+1} does not occur in the sequent $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$. The second rule generates an identity equality relation for the term t_n. The last two rules find a formula $h' : A$ in the antecedent and succedent respectively, and replace t with t' in A. The construction in Definition 3 non-trivially extends a similar construction of Hintikka CSS due to Larchey-Wendling [21] and a similar one in [16].

We also borrow the notions of consistency and finite-consistency from Larchey-Wendling's work [21]. We say $\mathcal{G}'; \Gamma' \vdash \Delta' \subseteq \mathcal{G}; \Gamma \vdash \Delta$ iff $\mathcal{G}' \subseteq \mathcal{G}$, $\Gamma' \subseteq \Gamma$

and $\Delta' \subseteq \Delta$. A sequent $\mathcal{G}; \Gamma \vdash \Delta$ is *finite* if $\mathcal{G}, \Gamma, \Delta$ are finite sets. Define $\mathcal{G}'; \Gamma' \vdash \Delta' \subseteq_f \mathcal{G}; \Gamma \vdash \Delta$ iff $\mathcal{G}'; \Gamma' \vdash \Delta' \subseteq \mathcal{G}; \Gamma \vdash \Delta$ and $\mathcal{G}'; \Gamma' \vdash \Delta'$ is finite. If $\mathcal{G}; \Gamma \vdash \Delta$ is a finite sequent, it is *consistent* iff it does not have a derivation in LS_{FOASL}. A (possibly infinite) sequent $\mathcal{G}; \Gamma \vdash \Delta$ is *finitely-consistent* iff every $\mathcal{G}'; \Gamma' \vdash \Delta' \subseteq_f \mathcal{G}; \Gamma \vdash \Delta$ is consistent.

We write $\mathcal{L}_i$ for the set of labels occurring in the sequent $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$, and write D_i for the set of terms which are disjoint from those in F in that sequent. Thus $\mathcal{L}_1 = \{a_1\}$ and $D_1 = \emptyset$. The following lemma states some properties of the construction of the sequents $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$.

Lemma 2. *For any $i \in \mathcal{N}$, the following properties hold:*

1. *$\mathcal{G}_i; \Gamma_i \vdash \Delta_i$ has no derivation*
2. *$\mathcal{L}_i \subseteq \{a_0, a_1, \cdots, a_{4i-1}\}$*
3. *$D_i \subseteq \{t_0, t_1, \cdots, t_i\}$*
4. *$\mathcal{G}_i; \Gamma_i \vdash \Delta_i \subseteq_f \mathcal{G}_{i+1}; \Gamma_{i+1} \vdash \Delta_{i+1}$*

Given the construction of the series of sequents in Definition 3, we define a notion of a *limit sequent* as the union of every sequent in the series.

Definition 4 (Limit sequent). *Let F be a formula unprovable in LS_{FOASL}. The* limit sequent *for F is the sequent $\mathcal{G}^\omega; \Gamma^\omega \vdash \Delta^\omega$ where $\mathcal{G}^\omega = \bigcup_{i \in \mathcal{N}} \mathcal{G}_i$ and $\Gamma^\omega = \bigcup_{i \in \mathcal{N}} \Gamma_i$ and $\Delta^\omega = \bigcup_{i \in \mathcal{N}} \Delta_i$ and where $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$ is as defined in Def.3.*

The last step is to show that the limit sequent is a Hintikka sequent, which gives rise to a counter-model of the formula that cannot be proved.

Lemma 3. *If F is a formula unprovable in LS_{FOASL}, then the limit sequent for F is a Hintikka sequent.*

Now we can finish the completeness theorem: whenever a FOASL formula has no derivation in LS_{FOASL}, there is an infinite counter-model. The theorem states the contraposition.

Theorem 2 (Completeness). *If F is valid in FOASL, then F is provable in LS_{FOASL}.*

5 Theories for $\mapsto$ in Separation Logics

Our predicate $\rightsquigarrow$ admits more interpretations than the standard $\mapsto$ predicate in SL heap model semantics. We can, however, approximate the behaviors of $\mapsto$ by formulating additional properties of $\mapsto$ as logical theories. We show next some of the theories for $\mapsto$ arising in various SL semantics.

5.1 Reynolds's semantics

The $\mapsto$ predicate in Reynolds's semantics can be formalized as follows, where the store s is a total function from variables to values, and the heap h is a finite partial function from addresses to values:

$$\frac{}{\mathcal{G};\Gamma;\epsilon : e_1 \mapsto e_2 \vdash \Delta} \mapsto L_1$$

$$\frac{\begin{array}{c}(\epsilon, h_0 \triangleright h_0);\mathcal{G}[\epsilon/h_1,h_0/h_2];\Gamma[\epsilon/h_1,h_0/h_2];h_0 : e_1 \mapsto e_2 \vdash \Delta[\epsilon/h_1,h_0/h_2]\\(h_0,\epsilon \triangleright h_0);\mathcal{G}[\epsilon/h_2,h_0/h_1];\Gamma[\epsilon/h_2,h_0/h_1];h_0 : e_1 \mapsto e_2 \vdash \Delta[\epsilon/h_2,h_0/h_1]\end{array}}{(h_1,h_2 \triangleright h_0);\mathcal{G};\Gamma;h_0 : e_1 \mapsto e_2 \vdash \Delta} \mapsto L_2$$

$$\frac{}{(h_1,h_2 \triangleright h_0);\mathcal{G};\Gamma;h_1 : e \mapsto e_1;h_2 : e \mapsto e_2 \vdash \Delta} \mapsto L_3 \qquad \frac{\mathcal{G};\Gamma\theta;h : e_1\theta \mapsto e_2\theta \vdash \Delta\theta}{\mathcal{G};\Gamma;h : e_1 \mapsto e_2;h : e_3 \mapsto e_4 \vdash \Delta} \mapsto L_4$$

$$\frac{\mathcal{G}[h_1/h_2];\Gamma[h_1/h_2];h_1 : e_1 \mapsto e_2 \vdash \Delta[h_1/h_2]}{\mathcal{G};\Gamma;h_1 : e_1 \mapsto e_2;h_2 : e_1 \mapsto e_2 \vdash \Delta} \mapsto L_5 \qquad \frac{(h_1,h_0 \triangleright h_2);\mathcal{G};\Gamma;h_1 : e_1 \mapsto e_2 \vdash \Delta}{\mathcal{G};\Gamma \vdash \Delta} HE$$

Side conditions:
Each label being substituted cannot be ϵ.
In HE, h_0 occurs in conclusion, h_1, h_2, e_1 are fresh.
In $\mapsto L_4$, $\theta = mgu(\{(e_1,e_3),(e_2,e_4)\})$.

Fig. 3. Points-to rules in LS_{SL}.

$$s, h \Vdash x \mapsto y \text{ iff } dom(h) = \{s(x)\} \text{ and } h(s(x)) = s(y).$$

Here we tackle the problem indirectly from the abstract separation logic angle. We give the following theories to approximate the semantics of $\mapsto$ in SL:

1. $\Box\forall e_1, e_2.(e_1 \mapsto e_2)\land\top^* \to \bot$
2. $\Box\forall e_1, e_2.(e_1 \mapsto e_2) \to \neg(\neg\top^* * \neg\top^*)$
3. $\Box\forall e_1, e_2, e_3, e_4.(e_1 \mapsto e_2) * (e_3 \mapsto e_4) \to \neg(e_1 = e_3)$
4. $\Box\forall e_1, e_2, e_3, e_4.(e_1 \mapsto e_2) \land (e_3 \mapsto e_4) \to (e_1 = e_3) \land (e_2 = e_4)$
5. $\Box\exists e_1\forall e_2.\neg((e_1 \mapsto e_2) \mathrel{-\!\!*} \bot)$
6. $\Box\forall e_1, e_2.(e_1 \mapsto e_2) \to (e_1 \rightsquigarrow e_2)$

Note that the opposite direction $(e_1 \rightsquigarrow e_2) \to (e_1 \mapsto e_2)$ does not necessarily hold because $\rightsquigarrow$ is weaker than $\mapsto$. The above theories intend to capture the inference rules for $\mapsto$ in LS_{SL} [17], the captured rules are given in Figure 3. The first five formulae simulate the rules $\mapsto L_1$, $\mapsto L_2$, $\mapsto L_3$, $\mapsto L_4$, and HE respectively. The rule $\mapsto L_5$ can be derived by $\rightsquigarrow_2$ and Formula 6.

Lemma 4. *The inference rules in Figure 3 are admissible in LS_{FOASL} when Formula 1 $\sim$ 6 are assumed true.*

The validity of Formula 1 to 6 w.r.t. Reynolds's SL model is easy to check, the rationale is similar to the soundness of corresponding rules in LS_{SL} [17]. Therefore Reynolds's SL is an instance of our logic.

Lemma 5. *Formula 1 $\sim$ 6 are valid in Reynolds's SL semantics.*

The rules in Figure 3 cover most of the rules for $\mapsto$ in LS_{SL} [17], but we have not found a way to handle the following rule (with two premises):

$$\frac{\begin{array}{c}(h_1,h_2 \triangleright h_0);\mathcal{G};\Gamma \vdash \Delta\\(h_3,h_4 \triangleright h_1);(h_5,h_6 \triangleright h_2);\mathcal{G};\Gamma;h_3 : e_1 \mapsto e_2;h_5 : e_1 \mapsto e_3 \vdash \Delta\end{array}}{\mathcal{G};\Gamma \vdash \Delta} HC$$

The rule HC effectively picks two arbitrary heaps h_1 and h_2, and does a case split of whether they can be combined or not. This rule seems to require more expressiveness than our logic. However, the above formulae cover most of properties about $\mapsto$ that existing tools for SL can handle, including the treatments in [17] and those for symbolic heaps [2].

5.2 Vafeiadis and Parkinson's SL

Vafeiadis and Parkinson's SL [35] is almost the same as Reynolds's definition, but they only consider values as addresses. This is a common setting in many applications, such as [14]. In this setting, the following formula is valid: $\top^* \rightarrow \neg((e_1 \mapsto e_2) -\!\!* \neg(e_1 \mapsto e_2))$. This formula, however, is invalid in Reynolds's SL. Obviously Formula 1 to 6 are valid in Vafeiadis and Parkinson's SL, thus their logic is also an instance of our abstract logic. To cater for the special feature, we propose a formula for "total addressability":

7. $\forall e_1, e_2. \Diamond(e_1 \mapsto e_2)$

This formula ensures that there must exist a heap $(e_1 \mapsto e_2)$ no matter what values e_1, e_2 have. This is sound because in Vafeiadis and Parkinson's SL, e_1 must denote a valid address, thus h with $dom(h) = \{s(e_1)\}$ and $h(s(e_1)) = s(e_2)$, where s is the store, must be a legitimate function, which by definition is a heap.

5.3 Lee et al.'s SL

Lee et al.'s proof system for SL corresponds to a non-standard semantics (although they used Reynolds's semantics in their paper) [25]. While there is not a reference of a formal definition of their non-standard semantics, their inference rule $-\!\!*\ Disj$ suggests that they forbid "incompatible heaps". For example, if there exists a heap $e_1 \mapsto e_2$, then there shall not exist another heap $(e_1 \mapsto e_3)$, where $e_2 \neq e_3$. Their $-\!\!*\ Disj$ rule can help derive the following formula, which is invalid in Reynolds's SL:

$$(((e_1 \mapsto e_2) * \top) -\!\!* \bot) \vee (((e_1 \mapsto e_3) * \top) -\!\!* \neg((e_1 \mapsto e_2) -\!\!* \bot)) \vee (e_2 = e_3)$$

If we assume that the above non-standard semantics conform with Reynolds's SL in other aspects (as validated by Formula 1 to 6), then it can be seen as a special instance of our abstract logic. The compatibility property can then be formulated as follows:

8. $\forall e_1, e_2. \Diamond(e_1 \mapsto e_2) \rightarrow \neg(\exists e_3. \neg(e_2 = e_3) \wedge \Diamond(e_1 \mapsto e_3))$

With Formula 8 we can prove the invalid formula above.

5.4 Thakur et al.'s SL

There are SL variants that forbid heaps with cyclic lists, for example, the one defined by Thakur et al. [33]. Consequently, the following two formulae are unsatisfiable in their SL:

$$e_1 \mapsto e_1 \qquad\qquad e_1 \mapsto e_2 * e_2 \mapsto e_1$$

To formulate this property, we first define a notion of a path:

$$\forall e_1, e_2.\Box(path(e_1, e_2) \equiv e_1 \mapsto e_2 \vee (\exists e_3.(e_1 \mapsto e_3) * path(e_3, e_2)))$$

where $\equiv$ denotes logical equivalence (bi-implication). Now the property of "acyclism" can be formulated as

9. $\forall e_1, e_2.\Box(path(e_1, e_2) \rightarrow e_1 \neq e_2)$

which renders cyclic paths unsatisfiable in our logic, too. Note that since our proof system does not support inductive definitions, we cannot force the interpretation of *path* to be the least fixed point of its definition. We leave the incorporation of inductive definitions to future work.

6 Implementation and Experiment

Our theorem prover for FOASL extends our previous prover for Reynolds's SL [17] with the ability to handle (non-inductive) predicates and modalities. To speed up proof search, instead of implementing $=_2$ and $\sim_2$, we use the following rules:

$$\frac{\mathcal{G}; \Gamma[s/t] \vdash \Delta[s/t]}{\mathcal{G}; h : s = t; \Gamma \vdash \Delta} =_2' \qquad \frac{\mathcal{G}\theta; \Gamma\theta \vdash \Delta\theta}{h_1 \sim h_2; \mathcal{G}; \Gamma \vdash \Delta} \sim_2'$$

where $\theta = [h_1/h_2]$ if $h_2 \neq \epsilon$ and $\theta = [h_2/h_1]$ otherwise.

These two rules can be shown to be interchangeable with $=_2$ and $\sim_2$. One direction, i.e., showing that $=_2'$ and $\sim_2'$ can be derived in FOASL, is straightforward. The other direction requires some further justification. Let LS'_{FOASL} be LS_{FOASL} with $=_2$ and $\sim_2$ replaced by $=_2'$ and $\sim_2'$ respectively, we then need to show that $=_2$ and $\sim_2$ are admissible in LS'_{FOASL}. To prove this, we follow a similar proof for free-equality rules for first-order terms by Schroeder-Heister [32]. The key part in that proof is in showing that provability is closed under substitutions. In our setting, we need to show that LS'_{FOASL} is closed under both term substitutions and label substitutions, which are stated below.

Lemma 6. *If $\mathcal{G}; \Gamma \vdash \Delta$ is derivable in LS'_{FOASL}, then so is $\mathcal{G}; \Gamma[s/t] \vdash \Delta[s/t]$ for any terms s and t.*

Lemma 7. *If $\mathcal{G}; \Gamma \vdash \Delta$ is derivable in LS'_{FOASL}, then so is $\mathcal{G}[h_1/h_2]; \Gamma[h_1/h_2] \vdash \Delta[h_1/h_2]$ for any label h_1 and label variable h_2.*

	Formula	Time
1	$((\top \mathrel{-\!\!*} (((k \mapsto c, d) \mathrel{-\!\!*} (l \mapsto a, b)) \to (l \mapsto a, b))) \to (l \mapsto a, b))$	< 0.001s
2	$((\exists x_2.((\exists x_1.((x_2 \mapsto x_1, b) \to \bot)) \to \bot)) \to (\exists x_3.(x_3 \mapsto a, b)))$	< 0.001s
3	$(((\top^* \to \bot) \to \bot) \to ((\exists x_1.((x_1 \mapsto a, b) * \top)) \to \bot))$	< 0.001s
4	$((\exists x_3\ x_2\ x_1.(((x_3 \mapsto a, x_2) * (x_1 \mapsto c, d)) \land x_2 = x_1)) \to$ $(\exists x_5\ x_4.((x_4 \mapsto c, d) * (x_5 \mapsto a, x_4))))$	< 0.001s
5	$((((e_1 \mapsto e_2) * \top) \land (((e_3 \mapsto e_4) * \top) \land$ $(((e_5 \mapsto e_6) * \top) \land (\neg(e_1 = e_3) \land (\neg(e_1 = e_5) \land \neg(e_3 = e_5)))))) \to$ $(((e_1 \mapsto e_2) * ((e_3 \mapsto e_4) * (e_5 \mapsto e_6))) * \top))$	0.9s
6	$((((e_1 \mapsto e_2) * \neg((e_3 \mapsto e_4) * \top)) \land ((e_3 \mapsto e_4) * \top)) \to e_1 = e_3)$	< 0.001s
7	$\neg((\neg\top^* * \neg\top^*) \mathrel{-\!\!*} \bot)$	0.0015s
8	$((\neg(((l_1 \mapsto p) * (l_2 \mapsto q)) \mathrel{-\!\!*} (\neg(l_3 \mapsto r)))) \to$ $(\neg((l_1 \mapsto p) \mathrel{-\!\!*} (\neg(\neg((l_2 \mapsto q) \mathrel{-\!\!*} (\neg(l_3 \mapsto r))))))))$	< 0.001s
9	$((\neg((l_1 \mapsto p) \mathrel{-\!\!*} (\neg(\neg((l_2 \mapsto q) \mathrel{-\!\!*} (\neg(l3 \mapsto r))))))) \to$ $(\neg(((l_1 \mapsto p) * (l_2 \mapsto q)) \mathrel{-\!\!*} (\neg(l_3 \mapsto r)))))$	< 0.001s
10	$((\neg((lx \mapsto ly) \mathrel{-\!\!*} (\neg((l1 \mapsto p) * (l2 \mapsto q))))) \to$ $(\neg((\neg((\neg((lx \mapsto ly) \mathrel{-\!\!*} (\neg(l1 \mapsto p)))) * ((l2 \mapsto q) \land$ $(\neg(\exists x_1.((lx \mapsto x_1) * \top)))))) \land (\neg((\neg((lx \mapsto ly) \mathrel{-\!\!*}$ $(\neg(l2 \mapsto q)))) * ((l1 \mapsto p) \land (\neg(\exists x_2.((lx \mapsto x_2) * \top)))))))))$	< 0.001s
11	$((\forall x_2\ x_1.\Diamond(x_2 \mapsto x_1)) \to (\top^* \to \neg((e_1 \mapsto e_2) \mathrel{-\!\!*} \neg(e_1 \mapsto e_2))))$	< 0.001s
12	$((\forall x_3\ x_2.(\Diamond(x_3 \mapsto x_2) \to \neg(\exists x_1.(\neg(x2 = x1) \land \Diamond(x_3 \mapsto x_1))))) \to$ $((((e_1 \mapsto e_2) * \top) \mathrel{-\!\!*} \bot) \lor ((((e_1 \mapsto e_3) * \top) \mathrel{-\!\!*} \neg((e_1 \mapsto e_2) \mathrel{-\!\!*} \bot))$ $\lor e_2 = e_3)))$	0.0025s

Table 2. Experiment on selected formulae.

Note that by restricting h_2 to a label variable, we forbid ϵ to be substituted in the above lemma. These two lemmas require induction on the height of derivations, and routine checks confirm that they both hold. Then it is a corollary that $=_2$ and $\sim_2$ are admissible in LS'_{FOASL}.

Since the heap model is widely used, our prover also includes useful rules to reason in the heap model, such as the derived rules in Figure 3. But we currently have not included the HC rule in our proof search procedure. Since many applications of SL involve reasoning about invalid addresses, such as *nil*, we also add a theory to capture a simple aspect of the invalid address *nil*:

10. $\Box\forall\vec{e}.(nil \mapsto \vec{e}) \to \bot$

Since the current prover is an extension of our previous prover, it builds in the inference rules for linked lists and binary trees for reasoning about the *symbolic heap* fragment of SL. It is also capable of proving theorems used in verification of a tail-recursive append function [26], as shown in [17]. However, we do not exploit these aspects here.

We illustrate a list of formulae provable by our prover in Table 2. Formulae 1 to 4 are examples drawn from Galmiche and Méry's work on resource graph tableaux for SL [15]. Formula 5 is a property about overlaid data structures: if the current heap contains $(e_1 \mapsto e_2)$ and $(e_3 \mapsto e_4)$ and $(e_5 \mapsto e_6)$, and they are pairwise distinct, then the current heap contains the combination of the three heaps. Formula 6 says that if the current heap can be split into two parts, one is $(e_1 \mapsto e_2)$ and the other part does not contain $(e_3 \mapsto e_4)$, and the current heap contains $(e_3 \mapsto e_4)$, then we deduce that $(e_3 \mapsto e_4)$ and $(e_1 \mapsto e_2)$ must be the same heap, therefore $e_1 = e_3$. Formula 7 says that any heap can be combined with a composite heap. We give a derivation of formula 7 in Appendix A. Formulae 8 to 10 are properties of "septraction" in SL with Rely-Guarantee [35]. Finally, formulae 11 and 12 show that our prover can easily support reasoning about Vafeiadis and Parkinson's SL (cf. Section 5.2) and Lee et al.'s SL (cf. Section 5.3) by simply adding the corresponding theories as assumptions. This is a great advantage over our previous work where new rules have to be implemented to extend the ability of the prover. To our knowledge most existing provers for SL cannot prove the formulae in Table 2. Examples of larger formulae used in program verification can be found in the experiment of our previous prover [17], upon which this prover is built.

7 Conclusion

This paper presents a first-order abstract separation logic with modalities. This logic is rich enough to express formulae in real-world applications such as program verification. We give a sound and complete labelled sequent calculus for this logic. The completeness of the finite calculus implies that our logic is recursively enumerable. To deal with $\mapsto$, we give a set of formulae to approximate the semantics of memory model. Of course, we cannot fully simulate $\mapsto$, but we can handle most properties about $\mapsto$ compared with existing tools for SL. Moreover, we can prove numerous formulae that many existing tools for SL cannot handle. The techniques discussed in this paper are demonstrated in a rather flexible theorem prover which supports automated reasoning in different SL variants without any change to the implementation. With this foundation, one can simply add formulae as "assumption", and prove theorems that cannot be proved in the base logic.

Acknowledgments

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2014NCR-NCR001-30) and administered by the National Cybersecurity R&D Directorate.

References

1. Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO*, pages 115–137. Springer, 2005.
2. Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Symbolic execution with separation logic. In *APLAS*, volume 3780, pages 52–68, 2005.

3. Rémi Brochenin, Stphane Demri, and Etienne Lozes. On the almighty wand. *Inform. and Comput.*, 211:106–137, 2012.
4. Stephen Brookes. A semantics for concurrent separation logic. *Theor. Comput. Sci.*, 375(1-3):227–270, April 2007.
5. James Brotherston. A unified display proof theory for bunched logic. *ENTCS*, 265:197–211, September 2010.
6. James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. Automated cyclic entailment proofs in separation logic. In *CADE*, pages 131–146, 2011.
7. James Brotherston and Max Kanovich. Undecidability of propositional separation logic and its neighbours. *Journal of ACM*, 2014.
8. James Brotherston and Jules Villard. Parametric completeness for separation theories. In *POPL*, pages 453–464. ACM, 2014.
9. Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. Local action and abstract separation logic. In *LICS*, pages 366–378. IEEE, 2007.
10. Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *FSTTCS*, volume 2245 of *LNCS*, pages 108–119, 2001.
11. Stéphane Demri and Morgan Deters. Expressive completeness of separation logic with two variables and no separating conjunction. In *CSL/LICS*, 2014.
12. Stéphane Demri, Didier Galmiche, Dominique Larchey-Wendling, and Daniel Mèry. Separation logic with one quantified variable. In *CSR*. LNCS 8476, 2014.
13. Robert Dockins, Aquinas Hobor, and Andrew W. Appel. A fresh look at separation algebras and share accounting. In *APLAS*, volume 5904, pages 161–177, 2009.
14. Dider Galmiche, Daniel Méry, and David Pym. The semantics of BI and resource tableaux. *MSCS*, 15(6):1033–1088, December 2005.
15. Didier Galmiche and Daniel Méry. Tableaux and resource graphs for separation logic. *J. Logic Comput.*, 20(1):189–231, 2010.
16. Zhé Hóu, Ranald Clouston, Rajeev Goré, and Alwen Tiu. Proof search for propositional abstract separation logics via labelled sequents. In *POPL*, 2014.
17. Zhé Hóu, Rajeev Goré, and Alwen Tiu. Automated theorem proving for assertions in separation logic with all connectives. In *CADE*, 2015.
18. Zhe Hóu and Alwen Tiu. Completeness for a first-order abstract separation logic. *arXiv:1608.06729 [cs.LO]*, 2016.
19. Jonas B. Jensen and Lars Birkedal. Fictional separation logic. In *ESOP*, volume 7211 of *LNCS*, pages 377–396, 2012.
20. Neelakantan R. Krishnaswami. Reasoning about iterators with separation logic. In *SAVCBS*, pages 83–86. ACM, 2006.
21. Dominique Larchey-Wendling. The formal strong completeness of partial monoidal Boolean BI. *JLC*, 2014.
22. Dominique Larchey-Wendling and Didier Galmiche. Exploring the relation between intuitionistic BI and Boolean BI: An unexpected embedding. *MSCS*, 19(3):435–500, 2009.
23. Dominique Larchey-Wendling and Didier Galmiche. The undecidability of Boolean BI through phase semantics. *LICS*, 0:140–149, 2010.
24. Dominique Larchey-Wendling and Didier Galmiche. Looking at separation algebras with Boolean BI-eyes. *TCS*, 2014.
25. Wonyeol Lee and Sungwoo Park. A proof system for separation logic with magic wand. In *POPL*, pages 477–490. ACM, 2014.
26. Toshiyuki Maeda, Haruki Sato, and Akinori Yonezawa. Extended alias type system using separating implication. In *TLDI*, pages 29–42. ACM, 2011.

27. Juan Antonio Navarro Pérez and Andrey Rybalchenko. Separation logic + superposition calculus = heap theorem prover. In *PLDI*. ACM, 2011.
28. Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *BSL*, 5(2):215–244, 1999.
29. Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *CSL*, pages 1–19, 2001.
30. Jonghyun Park, Jeongbong Seo, and Sungwoo Park. A theorem prover for Boolean BI. In *POPL*, POPL '13, pages 219–232, New York, NY, USA, 2013.
31. John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE, 2002.
32. Peter Schroeder-Heister. *Definitional reflection and the completion*, pages 333–347. Springer, 1994.
33. Aditya Thakur, Jason Breck, and Thomas Reps. Satisfiability modulo abstraction for separation logic with linked lists. SPIN 2014, pages 58–67, 2014.
34. Anne S. Troelstra and Helmut Schwichtenberg. *Basic Proof Theory.* 1996.
35. Viktor Vafeiadis and Matthew Parkinson. A marriage of rely/guarantee and separation logic. In *CONCUR*, volume 4703 of *LNCS*, 2007.

A An example derivation

We sometimes write $r \times n$ when it is obvious that the rule r is applied n times. We omit some formulae to save space. The derivation is given in the next page. The sub-derivation Π_1 is similar to Π_2.

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{$\scriptstyle \bot L$}
\UnaryInfC{$\cdots; h_4 : \bot \vdash \cdots$}
\AxiomC{$\mathit{\Pi}_1$}
\AxiomC{}
\RightLabel{$\scriptstyle \top^* R$}
\UnaryInfC{$\cdots; \epsilon : e_3 \mapsto e_4 \vdash \epsilon : \top^* \cdots$}
\AxiomC{}
\RightLabel{$\scriptstyle id$}
\UnaryInfC{$\cdots; \epsilon : e_3 \mapsto e_4 \vdash \epsilon : e_3 \mapsto e_4; \cdots$}
\RightLabel{$\scriptstyle \wedge R$}
\BinaryInfC{$\cdots; \epsilon : e_3 \mapsto e_4 \vdash \epsilon : (e_3 \mapsto e_4) \wedge \top^*; \cdots$}
\AxiomC{}
\RightLabel{$\scriptstyle \bot L$}
\UnaryInfC{$\cdots; \epsilon : \bot; \epsilon : e_3 \mapsto e_4 \vdash \cdots$}
\RightLabel{$\scriptstyle \to L$}
\BinaryInfC{$\cdots; \epsilon : ((e_3 \mapsto e_4) \wedge \top^*) \to \bot; \epsilon : e_3 \mapsto e_4 \vdash \cdots$}
\RightLabel{$\scriptstyle \forall L \times 2$}
\UnaryInfC{$\cdots; \epsilon : \forall e_1, e_2.((e_1 \mapsto e_2) \wedge \top^*) \to \bot; \epsilon : e_3 \mapsto e_4 \vdash \cdots$}
\RightLabel{$\scriptstyle \Box L \text{ on Formula 1}$}
\UnaryInfC{$\cdots; \epsilon : e_3 \mapsto e_4 \vdash \cdots$}
\RightLabel{$\scriptstyle Eq_1$}
\UnaryInfC{$(\epsilon, h_3 \rhd \epsilon); \cdots; h_3 : e_3 \mapsto e_4 \vdash \cdots$}
\RightLabel{$\scriptstyle \top^* L$}
\UnaryInfC{$\cdots; h_3 : e_3 \mapsto e_4; h_3 : \top^* \vdash \cdots$}
\RightLabel{$\scriptstyle \neg R$}
\UnaryInfC{$\cdots; h_3 : e_3 \mapsto e_4 \vdash h_3 : \neg\top^*; \cdots$}
\RightLabel{$\scriptstyle * R$}
\BinaryInfC{$(h_3, h_1 \rhd h_5); \cdots; h_1 : e_1 \mapsto e_2; h_3 : e_3 \mapsto e_4 \vdash h_5 : ((\neg\top^*) * (\neg\top^*)); \cdots$}
\RightLabel{$\scriptstyle -\!\!* L$}
\BinaryInfC{$(h_5, h_0 \rhd h_4); (h_3, h_1 \rhd h_5); \cdots; h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot; h_1 : (e_1 \mapsto e_2); h_3 : (e_3 \mapsto e_4) \vdash \cdots$}
\RightLabel{$\scriptstyle E$}
\UnaryInfC{$(h_0, h_5 \rhd h_4); (h_3, h_1 \rhd h_5); \cdots; h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot; h_1 : (e_1 \mapsto e_2); h_3 : (e_3 \mapsto e_4) \vdash \cdots$}
\RightLabel{$\scriptstyle A$}
\UnaryInfC{$(h_0, h_1 \rhd h_2); (h_2, h_3 \rhd h_4); \cdots; h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot; h_1 : (e_1 \mapsto e_2); h_3 : (e_3 \mapsto e_4) \vdash \cdots$}
\RightLabel{$\scriptstyle E \times 2$}
\UnaryInfC{$(h_1, h_0 \rhd h_2); (h_3, h_2 \rhd h_4); \cdots; h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot; h_1 : (e_1 \mapsto e_2); h_3 : (e_3 \mapsto e_4) \vdash h_4 : \bot; h_2 : \bot$}
\RightLabel{$\scriptstyle -\!\!* \text{ R}$}
\UnaryInfC{$(h_1, h_0 \rhd h_2); \cdots; h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot; h_1 : (e_1 \mapsto e_2) \vdash h_2 : (e_3 \mapsto e_4) -\!\!* \bot; h_2 : \bot$}
\RightLabel{$\scriptstyle \neg L$}
\UnaryInfC{$(h_1, h_0 \rhd h_2); \cdots; h_2 : \neg((e_3 \mapsto e_4) -\!\!* \bot); h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot; h_1 : (e_1 \mapsto e_2) \vdash h_2 : \bot$}
\RightLabel{$\scriptstyle \forall L$}
\UnaryInfC{$(h_1, h_0 \rhd h_2); \cdots; h_2 : \forall e_2.\neg((e_3 \mapsto e_2) -\!\!* \bot); h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot; h_1 : (e_1 \mapsto e_2) \vdash h_2 : \bot$}
\RightLabel{$\scriptstyle \exists L$}
\UnaryInfC{$(h_1, h_0 \rhd h_2); \cdots; h_2 : \exists e_1 \forall e_2.\neg((e_1 \mapsto e_2) -\!\!* \bot); h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot; h_1 : (e_1 \mapsto e_2) \vdash h_2 : \bot$}
\RightLabel{$\scriptstyle \Box L \text{ on Formula 5}$}
\UnaryInfC{$(h_1, h_0 \rhd h_2); \cdots; h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot; h_1 : (e_1 \mapsto e_2) \vdash h_2 : \bot$}
\RightLabel{$\scriptstyle -\!\!* R$}
\UnaryInfC{$\cdots; h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot \vdash h_0 : (e_1 \mapsto e_2) -\!\!* \bot$}
\RightLabel{$\scriptstyle \neg L$}
\UnaryInfC{$\cdots; h_0 : \neg((e_1 \mapsto e_2) -\!\!* \bot); h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot \vdash$}
\RightLabel{$\scriptstyle \forall L$}
\UnaryInfC{$\cdots; h_0 : \forall e_2.\neg((e_1 \mapsto e_2) -\!\!* \bot); h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot \vdash$}
\RightLabel{$\scriptstyle \exists L$}
\UnaryInfC{$\cdots; h_0 : \exists e_1 \forall e_2.\neg((e_1 \mapsto e_2) -\!\!* \bot); h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot \vdash$}
\RightLabel{$\scriptstyle \Box L \text{ on Formula 5}$}
\UnaryInfC{$; h_0 : \mathcal{A}; h_0 : ((\neg\top^*) * (\neg\top^*)) -\!\!* \bot \vdash$}
\RightLabel{$\scriptstyle \neg R$}
\UnaryInfC{$; h_0 : \mathcal{A} \vdash h_0 : \neg(((\neg\top^*) * (\neg\top^*)) -\!\!* \bot)$}
\RightLabel{$\scriptstyle \to R$}
\UnaryInfC{$; \vdash h_0 : \mathcal{A} \to \neg(((\neg\top^*) * (\neg\top^*)) -\!\!* \bot)$}
\end{prooftree}
$$