A Proof System for Separation Logic with Magic Wand

Wonyeol Lee   Sungwoo Park

Department of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)
Republic of Korea
{leewy,gla}@postech.ac.kr

Abstract

Separation logic is an extension of Hoare logic which is acknowledged as an enabling technology for large-scale program verification. It features two new logical connectives, separating conjunction and separating implication, but most of the applications of separation logic have exploited only separating conjunction without considering separating implication. Nevertheless the power of separating implication has been well recognized and there is a growing interest in its use for program verification. This paper develops a proof system for full separation logic which supports not only separating conjunction but also separating implication. The proof system is developed in the style of sequent calculus and satisfies the admissibility of cut. The key challenge in the development is to devise a set of inference rules for manipulating heap structures that ensure the completeness of the proof system with respect to separation logic. We show that our proof of completeness directly translates to a proof search strategy.

Categories and Subject Descriptors F.4.1 [*Mathematical Logic*]: Mechanical theorem proving, Proof theory

General Terms Verification

Keywords Separation logic; Proof system; Theorem prover

1. Introduction

Separation logic [28] is an extension of Hoare logic designed to simplify reasoning about programs manipulating mutable data structures with potential pointer aliasing. It features two new logical connectives, separating conjunction $\star$ and separating implication $-\!\!\star$, whose semantics directly assumes memory heaps structured as a monoid. Separating conjunction allows us to describe properties of two disjoint heaps with a single logical formula: $A \star B$ means that a given heap can be divided into two disjoint heaps satisfying A and B respectively. Separating implication, commonly known as magic wand, allows us to reason about hypothetical heaps extending a given heap: $A -\!\!\star B$ means that if a given heap is extended with a disjoint heap satisfying A, the resultant heap satisfies B. The use of the two separating connectives naturally leads to local reasoning in program verification in that we only need to reason locally about those heaps directly affected by the program.

So far, most of the applications of separation logic have exploited only separating conjunction. For example, all existing verification tools based on separation logic, such as Smallfoot [3], Space Invader [11], THOR [23], SLAyer [1], HIP [25], jStar [12], Xisa [10], VeriFast [19], Infer [7], and Predator [14], use a decidable fragment by Berdine *et al.* [2] or its extension which provides only separating conjunction. By virtue of the principle of local reasoning, however, these tools are highly successful in their individual verification domains despite not using separating implication at all.

Although separating implication is not discussed as extensively as separating conjunction in the literature, its power in program verification has nevertheless been well recognized. Just around the inception of separation logic, Yang [29] already gives an elegant proof of the correctness of the Schorr-Waite algorithm which relies crucially on the use of separating implication in the main loop invariant. Krishnaswami [20] shows how to reason abstractly about an iterator protocol with separation logic by exploiting separating implication in the specification of iterators. Maeda *et al.* [22] adopt the idea of separating implication in extending an alias type system in order to express tail-recursive operations on recursive data structures. Recently Hobor and Villard [17] give a concise proof of the correctness of Cheney's garbage collector in a proof system based on the ramify rule, a cousin of the frame rule of separation logic, whose premise checks a logical entailment involving separating implication. These promising results arguably suggest that introducing separating implication alone raises the level of technology for program verification as much as separation logic only with separating conjunction improves on Hoare logic.

Despite the potential benefit of separating implication in program verification, however, there is still no practical theorem prover for full separation logic. The state-of-the-art theorem provers for separation logic such as SeLoger [16] and SLP [24] support only separating conjunction, and the labelled tableau calculus by Galmiche and Méry [15] does not directly give rise to a proof search strategy. Because of the unavailability of such a theorem prover, all proofs exploiting separating implication should be manually checked, which can be time-consuming even with the help of lemmas provided by the proof system (as in [17]). Another consequence is that no existing verification tools based on separation logic can fully support backward reasoning by weakest precondition generation, which requires separating implication whenever verifying heap assignments (see Ishtiaq and O'Hearn [18]).

This paper develops a proof system $\mathbf{P_{SL}}$ for full separation logic which supports not only separating conjunction but also separating implication. Its design is based on the principle of proof by contradiction from classical logic, and we develop its inference rules in the style of sequent calculus. $\mathbf{P_{SL}}$ uses a new form of sequent,

POPL '14, January 22–24, 2014, San Diego, CA, USA.
Copyright © 2014 ACM 978-1-4503-2544-8/14/01…$15.00.
http://dx.doi.org/10.1145/2535838.2535871

called *world sequent*, in order to give a complete description of the world of heaps, and its use of world sequents allows us to treat separating implication in the same way that it treats separating conjunction. The key challenge in the development of $\mathbf{P_{SL}}$ is to devise a set of inference rules for manipulating heap structures so as to correctly analyze separating conjunction and separating implication. We show that $\mathbf{P_{SL}}$ satisfies the admissibility of cut and that it is sound and complete with respect to separation logic. The proof of completeness directly translates to a proof search strategy, which is the basis for our prototype implementation of $\mathbf{P_{SL}}$. We show that it is easy to extend $\mathbf{P_{SL}}$ with new logical connectives and predicates, such as an overlapping conjunction $A \uplus B$ by Hobor and Villard [17].

Separating implication has been commonly considered to be much harder to reason about than separating conjunction, as partially evidenced by lack of theorem provers supporting separating implication and abundance of verification systems supporting separating conjunction. Our development of $\mathbf{P_{SL}}$, however, suggests that a proof system designed in a principled way can support both logical connectives in a coherent way without requiring distinct treatments. Our prototype implementation of $\mathbf{P_{SL}}$ also suggests that such a proof system can develop into a practical theorem prover for separation logic. To the best of our knowledge, $\mathbf{P_{SL}}$ is the first proof system for full separation logic that satisfies the admissibility of cut and provides a concrete proof search strategy.

This paper is organized as follows. Section 2 gives preliminaries on separation logic. Section 3 develops our proof system $\mathbf{P_{SL}}$ and Section 4 gives three examples of proving world sequents. Section 6 proves the soundness and completeness of $\mathbf{P_{SL}}$ with respect to separation logic and Section 7 discusses the implementation and extension of $\mathbf{P_{SL}}$. Section 8 discusses related work and Section 9 concludes.

2. Semantics of separation logic

Separation logic extends classical first-order logic with multiplicative formulas from intuitionistic linear logic:

$$\begin{array}{rlll} \text{formula} & A, B, C & ::= & P \mid \bot \mid \neg A \mid A \vee A \mid \\ & & & \mathsf{I} \mid A \star A \mid A \mathrel{-\!\!\star} A \mid \exists a.A \\ \text{primitive formula} & P & ::= & [l \mapsto E] \mid E = E \mid \cdots \\ \text{expression} & E & ::= & x \mid a \mid \mathsf{L} \mid \cdots \\ \text{location expression} & l & ::= & x \mid a \mid \mathsf{L} \\ \text{value} & V & ::= & \mathsf{L} \mid \cdots \\ \text{location} & \mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3, \cdots & & \\ \text{stack variable} & x, y, z & & \\ \text{local variable} & a, b, c & & \end{array}$$

$\bot$, $\neg A$, $A \vee B$, and $\exists a.A$ are from classical first-order logic. I is the multiplicative unit. $A \star B$ is a separating conjunction and $A \mathrel{-\!\!\star} B$ is a separating implication. We define $\top$ as $\neg\bot$, $A \wedge B$ as $\neg(\neg A \vee \neg B)$, and $A \supset B$ as $\neg A \vee B$. We use conventional precedence rules for logical connectives: $\neg > \star > \vee > \mathrel{-\!\!\star} > \exists$. In this work, we do not consider inductively defined predicates.

Primitive formulas include a points-to relation $[l \mapsto E]$ for describing a singleton heap. All other primitive formulas describe relations between expressions; for simplicity, we consider only an equality relation $E = E'$. Expressions denote values which include locations L. Location expressions are a special class of expressions that denote locations. In the present work, we allow only locations as values, but it should be straightforward to introduce additional forms of expressions for new types of values such as booleans and integers. We syntactically distinguish between stack variables which originate from the program being verified (and thus may be called global variables instead) and local variables which are introduced by existential quantifiers (and thus can never appear outside corresponding existential formulas).

We specify the semantics of separation logic with respect to a stack and a heap. A stack S is a finite partial mapping $Var \rightharpoonup Val$ from stack variables to values where Var denotes the set of stack variables and Val denotes the set of values. Given a stack S, we can determine a unique value for every expression E, which we write as $[\![E]\!]_S$. A heap H is a finite partial mapping $Loc \rightharpoonup Val$ from locations to values where Loc denotes the set of locations. We write $H_1 \# H_2$ to mean that heaps H_1 and H_2 are disjoint, *i.e.*, $dom(H_1) \cap dom(H_2) = \varnothing$. We write $H_1 \circ H_2$ for the union of disjoint heaps H_1 and H_2 where $H_1 \# H_2$ is assumed, and ϵ for an empty heap. Heaps form a commutative cancellative monoid with $\circ$ as the associative operation and ϵ as the identity:

$$\begin{array}{rl} \text{(neutrality)} & H \circ \epsilon = H \\ \text{(commutativity)} & H_1 \circ H_2 = H_2 \circ H_1 \\ \text{(associativity)} & H_1 \circ (H_2 \circ H_3) = (H_1 \circ H_2) \circ H_3 \\ \text{(cancellativity)} & H \circ H_1 = H \circ H_2 \text{ implies } H_1 = H_2. \end{array}$$

Given a stack S and a heap H, we obtain the semantics of separation logic from the satisfaction relation $(S, H) \models A$ for formulas defined as follows:

- $(S, H) \models [l \mapsto E]$ iff. $H = \langle [\![l]\!]_S \mapsto [\![E]\!]_S \rangle$, *i.e.*, H is a singleton heap mapping $[\![l]\!]_S$ to $[\![E]\!]_S$.
- $(S, H) \models E = E'$ iff. $[\![E]\!]_S = [\![E']\!]_S$.
- $(S, H) \models \bot$ iff. never.
- $(S, H) \models \neg A$ iff. $(S, H) \not\models A$.
- $(S, H) \models A \vee B$ iff. $(S, H) \models A$ or $(S, H) \models B$.
- $(S, H) \models \mathsf{I}$ iff. $dom(H) = \varnothing$, *i.e.*, $H = \epsilon$.
- $(S, H) \models A \star B$ iff. $H = H_1 \circ H_2$ and $(S, H_1) \models A$ and $(S, H_2) \models B$ for some heaps H_1 and H_2.
- $(S, H) \models A \mathrel{-\!\!\star} B$ iff. $H_2 = H \circ H_1$ implies $(S, H_1) \not\models A$ or $(S, H_2) \models B$ for any heaps H_1 and H_2.
- $(S, H) \models \exists a.A$ iff. $(S, H) \models [V/a]A$ for some value V.

Note that the definition of $(S, H) \models \exists a.A$ directly substitutes value V for local variable a in formula A (in $[V/a]A$) without extending stack S because we syntactically distinguish between stack variables and local variables.

Although the satisfaction relation $(S, H) \models A$ is enough for specifying the semantics of separation logic, we deliberately derive the definition of its negation $(S, H) \not\models A$, which plays an equally important role in the development of our proof system:

- $(S, H) \not\models [l \mapsto E]$ iff. $H \neq \langle [\![l]\!]_S \mapsto [\![E]\!]_S \rangle$, *i.e.*, $dom(H) \neq \{[\![l]\!]_S\}$ or $H([\![l]\!]_S) \neq [\![E]\!]_S$.
- $(S, H) \not\models E = E'$ iff. $[\![E]\!]_S \neq [\![E']\!]_S$.
- $(S, H) \not\models \bot$ iff. always.
- $(S, H) \not\models \neg A$ iff. $(S, H) \models A$.
- $(S, H) \not\models A \vee B$ iff. $(S, H) \not\models A$ and $(S, H) \not\models B$.
- $(S, H) \not\models \mathsf{I}$ iff. $dom(H) \neq \varnothing$, *i.e.*, $H \neq \epsilon$.
- $(S, H) \not\models A \star B$ iff. $H = H_1 \circ H_2$ implies $(S, H_1) \not\models A$ or $(S, H_2) \not\models B$ for any heaps H_1 and H_2.
- $(S, H) \not\models A \mathrel{-\!\!\star} B$ iff. $H_2 = H \circ H_1$ and $(S, H_1) \models A$ and $(S, H_2) \not\models B$ for some heaps H_1 and H_2.
- $(S, H) \not\models \exists a.A$ iff. $(S, H) \not\models [V/a]A$ for any value V.

We observe that the definition for separating implication is symmetric to the definition for separating conjunction:

- $(S, H) \models A \star B$ should find a certain pair of heaps whereas $(S, H) \not\models A \star B$ should analyze an unspecified pair of heaps.
- $(S, H) \models A \mathrel{-\!\!\star} B$ should analyze an unspecified pair of heaps whereas $(S, H) \not\models A \mathrel{-\!\!\star} B$ should find a certain pair of heaps.

This symmetry suggests that we can incorporate separating implication into the proof system in an analogous way to separating conjunction.

A formula A is valid, written $\models A$, if $(S, H) \models A$ holds for every stack S and heap H. Our proof system $\mathbf{P_{SL}}$ can check the validity of every formula in separation logic.

3. Proof system $\mathbf{P_{SL}}$ for separation logic

This section presents the proof system $\mathbf{P_{SL}}$ for separation logic which is developed in the style of sequent calculus. We first explain *world sequents*, the main judgment in $\mathbf{P_{SL}}$, and then present the inference rules.

3.1 World sequents

The design of $\mathbf{P_{SL}}$ is based on the principle of proof by contradiction from classical logic. We describe the state of each heap with a set of true formulas and another set of false formulas. A world sequent in $\mathbf{P_{SL}}$ gives a description of the entire world of heaps, and a derivation of it means that the description is self-contradictory. Hence, in order to check the validity of a formula in separation logic, we use it as a false formula about an arbitrary heap w (about which nothing is known) and attempt to produce a logical contradiction by proving a world sequent consisting solely of heap w. The definition of world sequents and the principle of proof by contradiction are inherited from the nested sequent calculus for Boolean BI by Park *et al.* [26].

Since $\mathbf{P_{SL}}$ is designed to check the validity of a formula, it assumes an arbitrary stack, which implies that every stack variable denotes an arbitrary value. This in turn implies that in a derivation of a world sequent, we may use a fresh stack variable to denote an arbitrary value. We exploit this interpretation of stack variables in an inference rule for first-order formulas.

A world sequent consists of expression relations Θ, heap relations Σ, and heap sequents Π:

$$\begin{array}{rrcl}
\text{expression relation} & \theta & ::= & E = E' \mid E \neq E' \\
\text{expression relations} & \Theta & = & \theta_1, \cdots, \theta_n \\
\text{heap variable} & w, u, v & & \\
\text{heap relation} & \sigma & ::= & w \doteq \epsilon \mid w \not\doteq \epsilon \mid \\
 & & & w \doteq [l \mapsto E] \mid w \not\doteq [l \mapsto E] \mid \\
 & & & w \doteq w_1 \circ w_2 \\
\text{heap relations} & \Sigma & = & \sigma_1, \cdots, \sigma_n \\
\text{truth context} & \Gamma & ::= & \cdot \mid \Gamma, A \\
\text{falsehood context} & \Delta & ::= & \cdot \mid \Delta, A \\
\text{heap sequent} & \pi & ::= & [\Gamma \Longrightarrow \Delta]^w \\
\text{heap sequents} & \Pi & = & \pi_1, \cdots, \pi_n \\
\text{world sequent} & & & \Theta; \Sigma \parallel \Pi
\end{array}$$

- An expression relation θ is an equality or inequality between two expressions. If we introduce new forms of primitive formulas (*e.g.*, $E < E'$), we should introduce corresponding forms of expression relations.
- We assign a heap variable to each heap, and a heap relation σ relates a heap to an empty heap ($w \doteq \epsilon$ and $w \not\doteq \epsilon$), a singleton heap ($w \doteq [l \mapsto E]$ and $w \not\doteq [l \mapsto E]$), or the union of two disjoint heaps ($w \doteq w_1 \circ w_2$). We refer to those heap relation involving an empty heap or a singleton heap as *atomic heap relations*. As heaps form a commutative (cancellative) monoid, we assume commutativity of $\circ$ and use $w_1 \circ w_2$ and $w_2 \circ w_1$ interchangeably.
- A heap sequent $[\Gamma \Longrightarrow \Delta]^w$ describes heap w with truth context Γ and falsehood context Δ which contain true formulas and false formulas, respectively, about heap w.

In this way, a world sequent $\Theta; \Sigma \parallel \Pi$ gives a complete description of the world of heaps. We require that no local variable appear in expression relations and heap relations, and that a world sequent contain a unique heap sequent for each heap variable.

A world sequent represents a graph of heaps induced by heap relations. Given a heap relation $w \doteq w_1 \circ w_2$, we say that parent heap w has two child heaps w_1 and w_2 which are sibling heaps to each other. We can also extend parent-child relations to derive ancestor-descendant relations. If a heap has no pair of child heaps, we call it a terminal heap (where we ignore such a heap relation as $w \doteq w \circ w_\epsilon$ with $w_\epsilon \doteq \epsilon$); otherwise we call it a non-terminal heap. Note that a heap relation $w \doteq \epsilon$ or $w \doteq [l \mapsto E]$ does not immediately mean that w is a terminal heap because we may have another heap relation $w \doteq w_1 \circ w_2$. $\mathbf{P_{SL}}$, however, allows us to normalize all heap relations and turn w into a terminal heap.

$\mathbf{P_{SL}}$ also uses an expression contradiction judgment $\Theta \vdash \bot$ which is an abbreviation of a particular form of a world sequent $\Theta; \cdot \parallel \cdot$ and means that expression relations Θ produce a logical contradiction. Since the definition of expression relations is extensible, we do not give inference rules for the expression contradiction judgment and just assume a decidable system for it. For simplicity, we write $\Theta \vdash E = E'$ for $\Theta, E \neq E' \vdash \bot$, and $\Theta \vdash E \neq E'$ for $\Theta, E = E' \vdash \bot$. We write $\Theta \vdash [l \mapsto E] = [l' \mapsto E']$ for $\Theta \vdash l = l'$ and $\Theta \vdash E = E'$, and $\Theta \vdash [l \mapsto E] \neq [l' \mapsto E']$ for $\Theta \vdash l \neq l'$ or $\Theta \vdash E \neq E'$.

$\mathbf{P_{SL}}$ consists of logical rules in Figure 1, structural rules in Figure 2, and heap contradiction rules in Figure 3. The logical rules deal with formulas in heap sequents Π, the structural rules reorganize graphs of heaps induced by heap relations Σ, and the heap contradiction rules detect logical contradictions in heap relations Σ, or *heap contradictions*. $\mathbf{P_{SL}}$ shares the logical rules (for propositional and multiplicative formulas) with the nested sequent calculus for Boolean BI in [26], but the structural rules and the heap contradiction rules are specific to separation logic. We read every inference rule from the conclusion to the premise, and the derivation of a world sequent always terminates with a proof of a logical contradiction. Hence, in order to show the validity of a formula A, we try to prove a world sequent $\cdot; \cdot \parallel [\cdot \Longrightarrow A]^w$.

3.2 Logical rules of $\mathbf{P_{SL}}$

Figure 1 shows the logical rules of $\mathbf{P_{SL}}$. Except for the rule $\mathsf{ExpCont}$, a logical rule focuses on a principal formula in a heap sequent and either produces a logical contradiction (in the rule $\bot\mathsf{L}$) or rewrites the world sequent of the conclusion according to the semantics of separation logic in Section 2. For each type of formulas, $\mathbf{P_{SL}}$ has both a left rule, which analyzes a true formula about a heap, and a right rule, which analyzes a false formula about a heap, as in a typical sequent calculus. The rules for points-to relations introduce a corresponding heap relation. The rules for propositional and first-order formulas are from first-order classical logic. In the rule $\exists\mathsf{L}$, the fresh stack variable x denotes an arbitrary value. In the rules $\exists\mathsf{L}$ and $\exists\mathsf{R}$, we write $[E/a]A$ for substituting expression E for local variable a in formula A. The rules $=\mathsf{L}$ and $=\mathsf{R}$ are the only rules that add expression relations, and the rule $\mathsf{ExpCont}$ checks if expression relations Θ produce a logical contradiction.

The rules IL and IR use the fact the I is true only at an empty heap. The rules $\star\mathsf{L}$ and $\star\mathsf{R}$ are based on the following interpretation of multiplicative conjunction $\star$ which closely matches the semantics of separation logic in Section 2:

- $A \star B$ is true at heap w iff. $w \doteq w_1 \circ w_2$ and A is true at heap w_1 and B is true at heap w_2 for *some* heaps w_1 and w_2.
- $A \star B$ is false at heap w iff. $w \doteq w_1 \circ w_2$ implies that A is false at heap w_1 or that B is false at heap w_2 for *any* heaps w_1 and w_2.

Hence the rule $\star\mathsf{L}$ creates (*some*) fresh child heaps w_1 and w_2, whereas the rule $\star\mathsf{R}$ chooses (*any*) existing child heaps w_1 and

Rules for points-to relations:

$$\frac{\Theta;\Sigma, w \doteq [l \mapsto E] \parallel \Pi, [\Gamma \Longrightarrow \Delta]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma, [l \mapsto E] \Longrightarrow \Delta]^w}\ \mapsto\mathsf{L} \qquad \frac{\Theta;\Sigma, w \not\doteq [l \mapsto E] \parallel \Pi, [\Gamma \Longrightarrow \Delta]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, [l \mapsto E]]^w}\ \mapsto\mathsf{R}$$

Rules for propositional formulas:

$$\frac{}{\Theta;\Sigma \parallel \Pi, [\Gamma, \bot \Longrightarrow \Delta]^w}\ \bot\mathsf{L} \qquad \frac{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, A]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma, \neg A \Longrightarrow \Delta]^w}\ \neg\mathsf{L} \qquad \frac{\Theta;\Sigma \parallel \Pi, [\Gamma, A \Longrightarrow \Delta]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, \neg A]^w}\ \neg\mathsf{R}$$

$$\frac{\Theta;\Sigma \parallel \Pi, [\Gamma, A \Longrightarrow \Delta]^w \quad \Theta;\Sigma \parallel \Pi, [\Gamma, B \Longrightarrow \Delta]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma, A \vee B \Longrightarrow \Delta]^w}\ \vee\mathsf{L} \qquad \frac{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, A, B]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, A \vee B]^w}\ \vee\mathsf{R}$$

Rules for multiplicative formulas:

$$\frac{\Theta;\Sigma, w \doteq \epsilon \parallel \Pi, [\Gamma \Longrightarrow \Delta]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma, \mathsf{I} \Longrightarrow \Delta]^w}\ \mathsf{IL} \qquad \frac{\Theta;\Sigma, w \not\doteq \epsilon \parallel \Pi, [\Gamma \Longrightarrow \Delta]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, \mathsf{I}]^w}\ \mathsf{IR}$$

$$\frac{\textit{fresh } w_1, w_2 \qquad \Theta;\Sigma, w \doteq w_1 \circ w_2 \parallel \Pi, [\Gamma \Longrightarrow \Delta]^w, [A \Longrightarrow \cdot]^{w_1}, [B \Longrightarrow \cdot]^{w_2}}{\Theta;\Sigma \parallel \Pi, [\Gamma, A \star B \Longrightarrow \Delta]^w}\ \star\mathsf{L}$$

$$\frac{\begin{array}{ll} & \Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, A \star B]^w, [\Gamma_1 \Longrightarrow \Delta_1, A]^{w_1}, [\Gamma_2 \Longrightarrow \Delta_2]^{w_2} \\ w \doteq w_1 \circ w_2 \in \Sigma & \Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, A \star B]^w, [\Gamma_1 \Longrightarrow \Delta_1]^{w_1}, [\Gamma_2 \Longrightarrow \Delta_2, B]^{w_2} \end{array}}{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, A \star B]^w, [\Gamma_1 \Longrightarrow \Delta_1]^{w_1}, [\Gamma_2 \Longrightarrow \Delta_2]^{w_2}}\ \star\mathsf{R}$$

$$\frac{\begin{array}{ll} & \Theta;\Sigma \parallel \Pi, [\Gamma, A \mathrel{-\!\!\star} B \Longrightarrow \Delta]^w, [\Gamma_1 \Longrightarrow \Delta_1, A]^{w_1}, [\Gamma_2 \Longrightarrow \Delta_2]^{w_2} \\ w_2 \doteq w \circ w_1 \in \Sigma & \Theta;\Sigma \parallel \Pi, [\Gamma, A \mathrel{-\!\!\star} B \Longrightarrow \Delta]^w, [\Gamma_1 \Longrightarrow \Delta_1]^{w_1}, [\Gamma_2, B \Longrightarrow \Delta_2]^{w_2} \end{array}}{\Theta;\Sigma \parallel \Pi, [\Gamma, A \mathrel{-\!\!\star} B \Longrightarrow \Delta]^w, [\Gamma_1 \Longrightarrow \Delta_1]^{w_1}, [\Gamma_2 \Longrightarrow \Delta_2]^{w_2}}\ \mathrel{-\!\!\star}\mathsf{L}$$

$$\frac{\textit{fresh } w_1, w_2 \qquad \Theta;\Sigma, w_2 \doteq w \circ w_1 \parallel \Pi, [\Gamma \Longrightarrow \Delta]^w, [A \Longrightarrow \cdot]^{w_1}, [\cdot \Longrightarrow B]^{w_2}}{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, A \mathrel{-\!\!\star} B]^w}\ \mathrel{-\!\!\star}\mathsf{R}$$

Rules for first-order formulas:

$$\frac{\textit{fresh } x \qquad \Theta;\Sigma \parallel \Pi, [\Gamma, [x/a]A \Longrightarrow \Delta]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma, \exists a.A \Longrightarrow \Delta]^w}\ \exists\mathsf{L} \qquad \frac{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, [E/a]A, \exists a.A]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, \exists a.A]^w}\ \exists\mathsf{R}$$

Rules for primitive formulas for expressions:

$$\frac{\Theta, E = E';\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma, E = E' \Longrightarrow \Delta]^w}\ {=}\mathsf{L} \qquad \frac{\Theta, E \neq E';\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta]^w}{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, E = E']^w}\ {=}\mathsf{R} \qquad \frac{\Theta \vdash \bot}{\Theta;\Sigma \parallel \Pi}\ \mathsf{ExpCont}$$

Figure 1. Logical rules in the proof system $\mathbf{P_{SL}}$ for separation logic

w_2. Similarly the rules $\mathrel{-\!\!\star}$L and $\mathrel{-\!\!\star}$R are based on the following interpretation of multiplicative implication $\mathrel{-\!\!\star}$:

- $A \mathrel{-\!\!\star} B$ is true at heap w iff. $w_2 \doteq w \circ w_1$ implies that A is false at heap w_1 or that B is true at heap w_2 for *any* heaps w_1 and w_2.
- $A \mathrel{-\!\!\star} B$ is false at heap w iff. $w_2 \doteq w \circ w_1$ and A is true at heap w_1 and B is false at heap w_2 for *some* heaps w_1 and w_2.

Hence the rule $\mathrel{-\!\!\star}$L chooses (*any*) existing sibling heap w_1 and parent heap w_2, whereas as the rule $\mathrel{-\!\!\star}$R creates (*some*) fresh sibling heap w_1 and parent heap w_2. The rules $\star$L and $\mathrel{-\!\!\star}$R are the only logical rules that add parent-child heap relations to extend the graph of heaps, and introduce fresh heap variables w_1 and w_2 that are not found in the world sequent in the conclusion. The rules $\star$R and $\mathrel{-\!\!\star}$L are the only logical rules that replicate the principal formula into world sequents in the premise.

In the rules $\star$R and $\mathrel{-\!\!\star}$L, we allow equalities between heap variables w_1, w_2, and w. Since an equality between these heap variables invalidates the requirement that a world sequent contain a unique heap sequent for each heap variable, we interpret heap sequents for the same heap variable in the rules $\star$R and $\mathrel{-\!\!\star}$L as follows:

- In the conclusion, we implicitly replicate the same heap sequent as necessary.
- In the premise, we combine all changes made to individual heap sequents for the same heap variable to produce a single heap sequent.

For example, an equality $w = w_1$ in the rule $\star$R yields the following special instance:

$$\frac{\begin{array}{ll} & \Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, A \star B, A]^w, [\Gamma_2 \Longrightarrow \Delta_2]^{w_2} \\ w \doteq w \circ w_2 \in \Sigma & \Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, A \star B]^w, [\Gamma_2 \Longrightarrow \Delta_2, B]^{w_2} \end{array}}{\Theta;\Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, A \star B]^w, [\Gamma_2 \Longrightarrow \Delta_2]^{w_2}}$$

The rule $\star$R has two more special instances (corresponding to heap relations $w \doteq w_1 \circ w_1$ and $w \doteq w \circ w$), and similarly the rule $\mathrel{-\!\!\star}$L has a total of three special instances.

Now we can decompose each individual formula by applying its corresponding logical rule, thus accumulating expression relations and heap relations and creating fresh heaps. When expression relations become self-contradictory, we apply the rule ExpCont to complete the proof search. In order to obtain a complete proof search strategy, however, we should also be able to: 1) enumerate all heap relations $w \doteq w_1 \circ w_2$ and $w_2 \doteq w \circ w_1$ for a given heap w for the rules $\star$R and $\mathrel{-\!\!\star}$L; 2) produce heap contradictions, for example, from $w \doteq \epsilon$ and $w \doteq [l \mapsto E]$. (We assume that we can make a correct guess on expression E in the rule $\exists$R.) The remaining challenge is to devise a set of structural rules and another set of heap contradiction rules satisfying these two requirements, which would enable us to enumerate all feasible heap relations from those

Rules for disambiguating heap relations and leaving only disjoint terminal heaps:

$$\frac{\{w \doteq u_1 \circ u_2, w \doteq v_1 \circ v_2\} \subset \Sigma \quad \mathit{fresh}\ w_1, w_2, w_3, w_4 \quad \Theta; \Sigma, \begin{array}{l} u_1 \doteq w_1 \circ w_2, \\ u_2 \doteq w_3 \circ w_4, \\ v_1 \doteq w_1 \circ w_3, \\ v_2 \doteq w_2 \circ w_4 \end{array} \parallel \Pi, \begin{array}{l} [\cdot \Longrightarrow \cdot]^{w_1}, \\ [\cdot \Longrightarrow \cdot]^{w_2}, \\ [\cdot \Longrightarrow \cdot]^{w_3}, \\ [\cdot \Longrightarrow \cdot]^{w_4} \end{array}}{\Theta; \Sigma \parallel \Pi}\ \textsf{Disj}\star$$

$$\frac{\{w_1 \doteq u_1 \circ u_2, w_2 \doteq u_2 \circ u_3\} \subset \Sigma \quad \mathit{fresh}\ w, v_1, v_2, v_3 \quad \Theta; \Sigma, \begin{array}{l} w \doteq w_1 \circ v_3, \\ w \doteq v_1 \circ w_2, \\ u_1 \doteq v_1 \circ v_2, \\ u_3 \doteq v_2 \circ v_3 \end{array} \parallel \Pi, \begin{array}{l} [\cdot \Longrightarrow \cdot]^{w}, \\ [\cdot \Longrightarrow \cdot]^{v_1}, \\ [\cdot \Longrightarrow \cdot]^{v_2}, \\ [\cdot \Longrightarrow \cdot]^{v_3} \end{array}}{\Theta; \Sigma \parallel \Pi}\ \textsf{Disj}{-\!\!\star}$$

Rules for applying associativity of the union of disjoint heaps.

$$\frac{\{w \doteq u \circ v, u \doteq u_1 \circ u_2\} \subset \Sigma \quad \mathit{fresh}\ u' \quad \Theta; \Sigma, u' \doteq u_2 \circ v, w \doteq u_1 \circ u' \parallel \Pi, [\cdot \Longrightarrow \cdot]^{u'}}{\Theta; \Sigma \parallel \Pi}\ \textsf{Assoc}$$

Rules for propagating atomic heap relations:

$$\frac{\{w \doteq \epsilon, w \doteq w_1 \circ w_2\} \subset \Sigma \quad \Theta; \Sigma, w_1 \doteq \epsilon, w_2 \doteq \epsilon \parallel \Pi}{\Theta; \Sigma \parallel \Pi}\ \textsf{Prop}\epsilon$$

$$\frac{\{w \doteq [l \mapsto E], w \doteq w_1 \circ w_2\} \subset \Sigma \quad \Theta; \Sigma, w_1 \doteq [l \mapsto E], w_2 \doteq \epsilon \parallel \Pi \quad \Theta; \Sigma, w_1 \doteq \epsilon, w_2 \doteq [l \mapsto E] \parallel \Pi}{\Theta; \Sigma \parallel \Pi}\ \textsf{Prop}{\mapsto}$$

$$\frac{\{w \not\doteq \epsilon, w \doteq w_1 \circ w_2\} \subset \Sigma \quad \Theta; \Sigma, w_1 \not\doteq \epsilon \parallel \Pi \quad \Theta; \Sigma, w_2 \not\doteq \epsilon \parallel \Pi}{\Theta; \Sigma \parallel \Pi}\ \textsf{Prop}\epsilon\not\doteq$$

$$\frac{\{w \not\doteq [l \mapsto E], w \doteq w_1 \circ w_2\} \subset \Sigma \quad \begin{array}{ll} \Theta; \Sigma, w_1 \not\doteq \epsilon, w_1 \not\doteq [l \mapsto E] \parallel \Pi & \Theta; \Sigma, w_1 \not\doteq [l \mapsto E], w_2 \not\doteq [l \mapsto E] \parallel \Pi \\ \Theta; \Sigma, w_1 \not\doteq \epsilon, w_2 \not\doteq \epsilon \parallel \Pi & \Theta; \Sigma, w_2 \not\doteq \epsilon, w_2 \not\doteq [l \mapsto E] \parallel \Pi \end{array}}{\Theta; \Sigma \parallel \Pi}\ \textsf{Prop}{\mapsto}\not\doteq$$

Rules for normalizing heap relations:

$$\frac{\Theta; [w/w']\Sigma, w \doteq u \circ v \parallel \Pi, [\Gamma, \Gamma' \Longrightarrow \Delta, \Delta']^{w}}{\Theta; \Sigma, w \doteq u \circ v, w' \doteq u \circ v \parallel \Pi, [\Gamma \Longrightarrow \Delta]^{w}, [\Gamma' \Longrightarrow \Delta']^{w'}}\ \textsf{NormEq}$$

$$\frac{\Theta; [w/u]\Sigma, v \doteq \epsilon \parallel \Pi, [\Gamma, \Gamma' \Longrightarrow \Delta, \Delta']^{w}}{\Theta; \Sigma, w \doteq u \circ v, v \doteq \epsilon \parallel \Pi, [\Gamma \Longrightarrow \Delta]^{w}, [\Gamma' \Longrightarrow \Delta']^{u}}\ \textsf{NormPC} \qquad \frac{\Theta; [w/u]\Sigma, w \doteq \epsilon \parallel \Pi, [\Gamma, \Gamma' \Longrightarrow \Delta, \Delta']^{w}}{\Theta; \Sigma, w \doteq \epsilon, u \doteq \epsilon \parallel \Pi, [\Gamma \Longrightarrow \Delta]^{w}, [\Gamma' \Longrightarrow \Delta']^{u}}\ \textsf{NormEmpty}$$

Rules for creating an empty heap and applying the monoid laws for empty heaps:

$$\frac{\mathit{fresh}\ w_\epsilon \quad \Theta; \Sigma, w_\epsilon \doteq \epsilon \parallel \Pi, [\cdot \Longrightarrow \cdot]^{w_\epsilon}}{\Theta; \Sigma \parallel \Pi}\ \textsf{ENew} \qquad \frac{w_\epsilon \doteq \epsilon \in \Sigma \quad \Theta; \Sigma, w \doteq w \circ w_\epsilon \parallel \Pi}{\Theta; \Sigma \parallel \Pi}\ \textsf{EJoin} \qquad \frac{w \doteq w \circ u \in \Sigma \quad \Theta; \Sigma, u \doteq \epsilon \parallel \Pi}{\Theta; \Sigma \parallel \Pi}\ \textsf{ECancel}$$

Figure 2. Structural rules in the proof system $\mathbf{P_{SL}}$ for separation logic

generated by the logical rules and detect all types of heap contradictions.

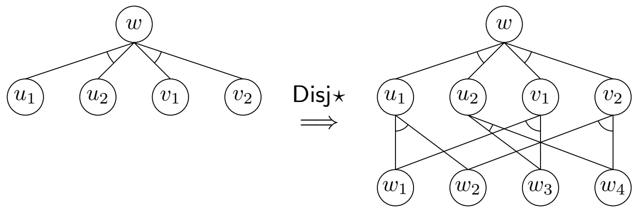
3.3 Structural rules of $\mathbf{P_{SL}}$

The structural rules of $\mathbf{P_{SL}}$ are divided into five groups according to their roles in reorganizing graphs of heaps represented by world sequents. The order of the structural rules in Figure 2 roughly follows their use in the proof of the completeness of $\mathbf{P_{SL}}$ with respect to separation logic (Theorem 6.4). In a certain sense, we design the structural rules so as to obtain a complete proof search strategy for separation logic when the logical rules are already given as in Figure 1. Below we informally discuss the key properties of the structural rules, which we formally present as part of the proof of the completeness of $\mathbf{P_{SL}}$ in Section 6.2.

3.3.1 Rules for disambiguating heap relations

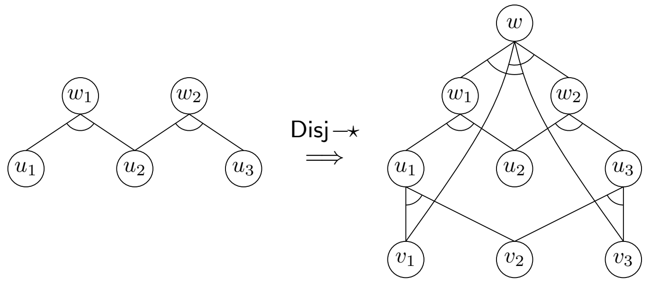
The rules Disj$\star$ and Disj$-\!\!\star$ disambiguate heap relations in order to leave only disjoint terminal heaps. Roughly speaking, two heaps are disjoint if they share a common ancestor which has a heap relation separating them.

In the premise of the rule Disj$\star$, child heaps u_i and v_j $(i, j = 1, 2)$ share a common parent heap w, but their exact relations are unknown. For example, heap u_1 may completely subsume, partially overlap with, or be disjoint from heap v_1. In general, each pair of child heaps u_i and v_j are allowed to share a common child heap, so the rule Disj$\star$ disambiguates their relations by introducing four fresh terminal heaps, w_1 to w_4, which are all disjoint from each other:



Now, for example, we may assume that the intersection of heaps u_1 and v_1 is represented by heap w_1. Note that if heap u_i or v_j is not a terminal heap, the rule Disj$\star$ gives rise to unknown relations between the existing child heaps of u_i or v_j and two of the fresh terminal heaps. The rule Disj$\star$ corresponds to the cross-split axiom for separation algebras [13].

The rule Disj$-\!\star$ disambiguates relations between two sibling heaps u_1 and u_3 of heap u_2 by introducing three fresh terminal heaps, v_1 to v_3, which are all disjoint from each other:

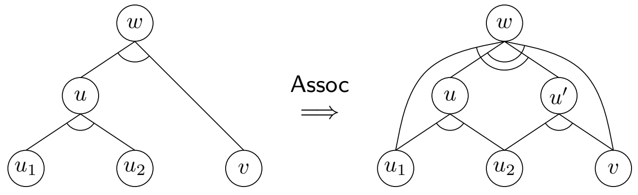


Similarly to the rule Disj$\star$, the rule Disj$-\!\star$ may give rise to new unknown relations involving heap w_1, w_2, u_1, or u_3. Unlike the rule Disj$\star$, however, it also creates a common ancestor w of all heaps. Otherwise the fresh terminal heaps themselves come to have unknown relations, thereby defeating the purpose of applying the rule Disj$-\!\star$.

Thus the rule Disj$\star$ eliminates unknown relations between child heaps and the rule Disj$-\!\star$ eliminates unknown relations between sibling heaps of a certain heap, both potentially creating similar unknown relations. By repeatedly applying these rules, we can eventually obtain a graph of heaps such that: *1) there exists a root heap that is an ancestor of all other heaps; 2) all terminal heaps in the graph are disjoint.*

3.3.2 Rule for applying associativity of $\circ$

The rule Assoc creates new heap relations according to associativity of the union of disjoint heaps. Suppose that we have two heap relations $w \doteq u \circ v$ and $u \doteq u_1 \circ u_2$. The rule Assoc introduces a fresh heap u' in order to associate two heaps u_2 and v which are known to be disjoint but do not have a common parent heap yet; it also assigns heap w as the common parent heap of heaps u_1 and u':



Note that unlike the rules Disj$\star$ and Disj$-\!\star$, the rule Assoc creates no fresh terminal heaps.

The rule Assoc is crucial for enumerating all heap relations involving a particular heap. The basic observation is that by repeatedly applying the rule Assoc to a graph of heaps, we can eventually obtain another graph of heaps with the same set of terminal heaps such that *for each combination of terminal heaps, there is at least one heap subsuming exactly these terminal heaps and no others.* By starting with a graph of heaps obtained by repeatedly applying the rules Disj$\star$ and Disj$-\!\star$, then, we can enumerate all feasible heap relations $w \doteq w_1 \circ w_2$ and $w_2 \doteq w \circ w_1$ for a particular heap w where we assume that heaps w_1 and w_2 are in the graph. For the case that w_1 or w_2 is an empty heap, however, we need another set of structural rules for dealing with empty heaps. We should also combine heap sequents for the same heap. (Hence we have not yet accomplished the first requirement for obtaining a complete proof search strategy.)

3.3.3 Rules for propagating atomic heap relations

The rules for propagating atomic heap relations, or propagation rules, are designed to propagate all atomic heap relations ($w \doteq \epsilon$, $w \not\doteq \epsilon$, $w \doteq [l \mapsto E]$, $w \not\doteq [l \mapsto E]$) from non-terminal heaps to terminal heaps. A propagation rule converts an atomic heap relation for a heap w into semantically equivalent heap relations for its child heaps w_1 and w_2 (with $w \doteq w_1 \circ w_2$). It rewrites world sequents according to the following fact on atomic heap relations where we assume $w \doteq w_1 \circ w_2$:

- $w \doteq \epsilon$ iff. $w_1 \doteq \epsilon$ and $w_2 \doteq \epsilon$ (for the rules Propϵ and Prop$\epsilon\not\doteq$).
- $w \doteq [l \mapsto E]$ iff. either $w_1 \doteq [l \mapsto E]$ and $w_2 \doteq \epsilon$, or $w_1 \doteq \epsilon$ and $w_2 \doteq [l \mapsto E]$ (for the rules Prop$\mapsto$ and Prop$\mapsto\not\doteq$).

For example, we negate the second clause to derive the rule Prop$\mapsto\not\doteq$ which has four world sequents in the premise:

- $w \not\doteq [l \mapsto E]$ iff. 1) $w_1 \not\doteq [l \mapsto E]$ and $w_1 \not\doteq \epsilon$; 2) $w_1 \not\doteq [l \mapsto E]$ and $w_2 \not\doteq [l \mapsto E]$; 3) $w_2 \not\doteq \epsilon$ and $w_1 \not\doteq \epsilon$; or 4) $w_2 \not\doteq \epsilon$ and $w_2 \not\doteq [l \mapsto E]$.

Note that although the new heap relations for the child heaps w_1 and w_2 collectively imply the original heap relation σ, we have to preserve σ in every world sequent of the premise because it may still interact with another pair of child heaps w_1' and w_2' (with $w \doteq w_1' \circ w_2'$). After considering all such interactions, however, we may safely discard σ (by the rule Weaken to be introduced in Section 6.2).

The propagation rules are the first step toward a complete procedure for producing heap contradictions (which detect all types of heap contradictions). Suppose that we repeatedly apply the propagation rules until no more new heap relations arise from atomic heap relations. After discarding atomic heap relations for non-terminal heaps, we obtain a set of graphs of heaps (with the same structure as the original graph) in which *atomic heap relations reside only for terminal heaps.* Now, in order to produce heap contradictions from atomic heap relations, we need to inspect only terminal heaps of these graphs, which makes it much easier to develop a complete procedure for producing heap contradictions.

3.3.4 Rules for normalizing heap relations

The rules for normalizing heap relations, or normalization rules, merge two identical heaps and isolate empty heaps while simultaneously shrinking the graph of heaps. In the rule NormEq, heaps w and w' are identical and we merge the two heaps by combining their heap sequents. Here we write $[w/w']\Sigma$ for substituting w for w' in every heap relation in Σ. Note that the rule NormEq implies that $\circ$ is (partial) deterministic. In the rule NormPC, $v \doteq \epsilon$ implies that heaps w and u are identical. Hence we merge the two heaps by combining their heap sequents and isolate the empty heap v from the graph of heaps. Similarly the rule NormEmpty merges two empty heaps w and u by combining their heap sequents. In effect, it allows us to collect all empty heaps, which do not need to be distinguished for the purpose of proof search, into a single empty heap. Note that the rule NormEmpty implies the existence of a single unit of $\circ$. By repeatedly applying the normalization rules to a graph of heaps, we can eventually obtain an equivalent graph which maintains a unique world sequent for each heap and possibly a unique empty heap isolated from the graph.

It is important that the normalization rules shrink the graph of heaps, but preserve all the properties established by the previous structural rules. For example, if the graph satisfies the property that all terminal heaps are disjoint (established by the rules Disj$\star$ and Disj$-\!\star$), it continues to satisfy the same property after an application of any normalization rule. Hence it is safe to aggressively apply the normalization rules after applying the previous structural rules.

$$\dfrac{}{\Theta; \Sigma, w \doteq \epsilon, w \doteq [l \mapsto E] \parallel \Pi}\ \mathsf{Cont}\,\epsilon\mapsto \qquad \dfrac{}{\Theta; \Sigma, w \doteq \epsilon, w \mathrel{\dot{\neq}} \epsilon \parallel \Pi}\ \mathsf{Cont}\,\epsilon\dot{\neq} \qquad \dfrac{\Theta \vdash [l \mapsto E] \neq [l' \mapsto E']}{\Theta; \Sigma, w \doteq [l \mapsto E], w \doteq [l' \mapsto E'] \parallel \Pi}\ \mathsf{Cont}\mapsto\doteq$$

$$\dfrac{\Theta \vdash [l \mapsto E] = [l' \mapsto E']}{\Theta; \Sigma, w \doteq [l \mapsto E], w \mathrel{\dot{\neq}} [l' \mapsto E'] \parallel \Pi}\ \mathsf{Cont}\mapsto\dot{\neq} \qquad \dfrac{\Theta \vdash l_1 = l_2}{\Theta; \Sigma, w \doteq w_1 \circ w_2, w_1 \doteq [l_1 \mapsto E_1], w_2 \doteq [l_2 \mapsto E_2] \parallel \Pi}\ \mathsf{Cont}\circ\mapsto$$

Figure 3. Heap contradiction rules in the proof system $\mathbf{P_{SL}}$ for separation logic

3.3.5 Rules for dealing with empty heaps

The last group of structural rules create an empty heap and apply the monoid laws for empty heaps. We use the rule ENew when no rule can directly produce an empty heap. The rule EJoin, which is based on neutrality of ϵ, is sound because extending a heap with an empty heap makes no change. The rule $\mathsf{ECancel}$ creates an empty heap when a heap is shown to be a child heap of itself. It is based on cancellativity of $\circ$: we can always generate $w \doteq w \circ w_\epsilon$ and $w_\epsilon \doteq \epsilon$ by the rules ENew and EJoin, and $w \doteq w \circ u$ and $w \doteq w \circ w_\epsilon$ imply $u \doteq \epsilon$ by cancellativity of $\circ$. (Similarly the rule NormPC is based on cancellativity of $\circ$: we can always generate $w \doteq w \circ v$ by the rule EJoin, and $w \doteq u \circ v$ and $w \doteq w \circ v$ imply $w = u$.) It turns out that we need the rule $\mathsf{ECancel}$ for the proof of admissibility of cut (Theorem 5.1), but not for the proof of the completeness of $\mathbf{P_{SL}}$ (Theorem 6.4).

Now we can accomplish the first requirement for obtaining a complete proof search strategy. In order to enumerate all heap relations $w \doteq w_1 \circ w_2$ and $w_2 \doteq w \circ w_1$ for a heap w, we first analyze the graph of heaps obtained by repeatedly applying the previous structural rules. This produces all such heap relations that involve only non-empty heaps. Then we apply the rule EJoin as necessary to produce all such heap relations that involve empty heaps.

We may think of the rule EJoin as extending heap relations for heap w with a pair of child heaps w and w_ϵ, or a pair of sibling heap w_ϵ and parent heap w. It is the only rule in $\mathbf{P_{SL}}$ that is capable of creating new heap relations for an arbitrary heap. Thus, whenever an arbitrary heap with no heap relation needs a pair of child heaps or a pair of sibling and parent heaps, we should apply the rule EJoin which inevitably reuses an existing empty heap. For example, we prove the validity of $\top \star \top$ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot; w_\epsilon \doteq \epsilon, w \doteq w \circ w_\epsilon \parallel [\bot \Longrightarrow \top \star \top]^w, [\cdot \Longrightarrow \cdot]^{w_\epsilon}}\ \bot\mathsf{L}}{\cdot; w_\epsilon \doteq \epsilon, w \doteq w \circ w_\epsilon \parallel [\cdot \Longrightarrow \top \star \top, \top]^w, [\cdot \Longrightarrow \cdot]^{w_\epsilon}}\ \neg\mathsf{R} \qquad \vdots}{\cdot; w_\epsilon \doteq \epsilon, w \doteq w \circ w_\epsilon \parallel [\cdot \Longrightarrow \top \star \top]^w, [\cdot \Longrightarrow \cdot]^{w_\epsilon}}\ \star\mathsf{R}}{\cdot; w_\epsilon \doteq \epsilon \parallel [\cdot \Longrightarrow \top \star \top]^w, [\cdot \Longrightarrow \cdot]^{w_\epsilon}}\ \mathsf{EJoin}}{\cdot; \cdot \parallel [\cdot \Longrightarrow \top \star \top]^w}\ \mathsf{ENew}$$

Note that there is no need to create fresh child heaps w_1 and w_2 with $w \doteq w_1 \circ w_2$: if we can prove the world sequent using fresh child heaps about which nothing is known, we should be able to prove it equally by reusing an existing empty heap. Similarly we prove the validity of $\neg(\top \mathrel{-\!\!\star} \bot)$ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots \qquad \dfrac{}{\cdot; w_\epsilon \doteq \epsilon, w \doteq w \circ w_\epsilon \parallel [\top \mathrel{-\!\!\star} \bot, \bot \Longrightarrow \cdot]^w, [\cdot \Longrightarrow \cdot]^{w_\epsilon}}\ \bot\mathsf{L}}{\cdot; w_\epsilon \doteq \epsilon, w \doteq w \circ w_\epsilon \parallel [\top \mathrel{-\!\!\star} \bot \Longrightarrow \cdot]^w, [\cdot \Longrightarrow \cdot]^{w_\epsilon}}\ \mathrel{-\!\!\star}\mathsf{L}}{\cdot; w_\epsilon \doteq \epsilon \parallel [\top \mathrel{-\!\!\star} \bot \Longrightarrow \cdot]^w, [\cdot \Longrightarrow \cdot]^{w_\epsilon}}\ \mathsf{EJoin}}{\cdot; \cdot \parallel [\top \mathrel{-\!\!\star} \bot \Longrightarrow \cdot]^w}\ \mathsf{ENew}}{\cdot; \cdot \parallel [\cdot \Longrightarrow \neg(\top \mathrel{-\!\!\star} \bot)]^w}\ \neg\mathsf{R}$$

Again we do not create fresh sibling and parent heaps and instead reuse an existing empty heap.

3.4 Heap contradiction rules of $\mathbf{P_{SL}}$

The proof system $\mathbf{P_{SL}}$ has five rules, $\mathsf{Cont}\,\epsilon\dot{\neq}$ to $\mathsf{Cont}\circ\mapsto$, for producing heap contradictions. In conjunction with the structural rules, these rules enable us to detect all types of heap contradictions, thereby accomplishing the second requirement for obtaining a complete proof strategy.

To see why, assume a world sequent $\Theta; \Sigma \parallel \Pi$. By repeatedly applying the structural rules in the same order as presented above, we can obtain a semantically equivalent set of world sequents $\Theta; \Sigma_i \parallel \Pi_i$ $(i = 1, \cdots, n)$ such that: 1) Σ_i induces a graph of heaps in which all terminal heaps are disjoint; 2) atomic heap relations reside only for terminal heaps and we need to consider only terminal heaps to detect heap contradictions. For an empty heap, we use the rules $\mathsf{Cont}\,\epsilon\dot{\neq}$ and $\mathsf{Cont}\,\epsilon\mapsto$ which express the only ways to produce heap contradictions from an empty heap with $w \doteq \epsilon$. Note that $w \doteq \epsilon$ and $w \mathrel{\dot{\neq}} [l \mapsto E]$ do not produce a heap contradiction because the former implies the latter. For a terminal singleton heap, we use the rules $\mathsf{Cont}\mapsto\doteq$ and $\mathsf{Cont}\mapsto\dot{\neq}$ which express the only ways to produce heap contradictions from a terminal singleton heap with $w \doteq [l \mapsto E]$. Note that $w \doteq [l \mapsto E]$ implies $w \mathrel{\dot{\neq}} \epsilon$ always and $w \mathrel{\dot{\neq}} [l' \mapsto E']$ if $\Theta \vdash [l \mapsto E] \neq [l' \mapsto E']$ holds. We do not need to consider other forms of terminal heaps, for example, those with no atomic heap relations. Finally the rule $\mathsf{Cont}\circ\mapsto$ expresses the only way to produce heap contradictions from two disjoint terminal singleton heaps with $w_1 \doteq [l_1 \mapsto E_1]$ and $w_2 \doteq [l_2 \mapsto E_2]$. This is because if $\Theta \vdash l_1 \neq l_2$ holds, the two heap relations are consistent with $w \doteq w_1 \circ w_2$ and we cannot produce a heap contradiction. In this way, we can detect all types of heap contradictions in heap relations.

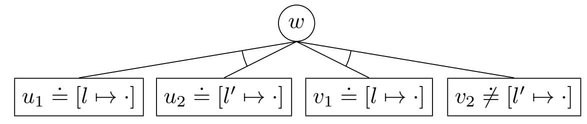
4. Examples of proving world sequents

This section presents three examples of proving world sequents in $\mathbf{P_{SL}}$. We write $[l \mapsto \cdot]$ to denote $[l \mapsto E]$ for some expression E and assume two distinct location expressions l and l' $(l \neq l')$.

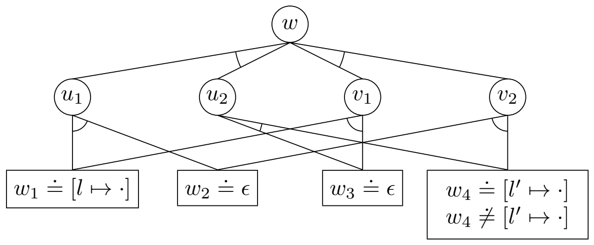
4.1 $\neg(([l \mapsto \cdot] \star [l' \mapsto \cdot]) \wedge ([l \mapsto \cdot] \star \neg[l' \mapsto \cdot]))$

The goal formula implies that given a fragment of a heap, we can uniquely determine the remaining fragment. Its proof illustrates that the rule $\mathsf{Disj}\star$ indirectly applies cancellativity of $\circ$ to two pairs of child heaps.

We begin with a world sequent $\cdot; \cdot \parallel [\cdot \Longrightarrow C]^w$ where C is the goal formula. After applying the logical rules, we obtain the following graph of heaps where heap relations are displayed for child heaps:



Then we apply the rule $\mathsf{Disj}\star$ and the propagation rules $\mathsf{Prop}\mapsto$ and $\mathsf{Prop}\mapsto\dot{\neq}$ to generate $2 \times 2 \times 2 \times 4 = 32$ different world sequents as new goals. All these new goals are immediately provable by the rules $\mathsf{Cont}\,\epsilon\mapsto$, $\mathsf{Cont}\,\epsilon\dot{\neq}$, and $\mathsf{Cont}\mapsto\dot{\neq}$. An example of such a world sequent has heap relations $w_4 \doteq [l' \mapsto \cdot]$, originating from heap u_2 by the rule $\mathsf{Prop}{\mapsto}$, and $w_4 \not\doteq [l' \mapsto \cdot]$, originating from heap v_2 by the rule $\mathsf{Prop}{\mapsto}{\not\doteq}$:



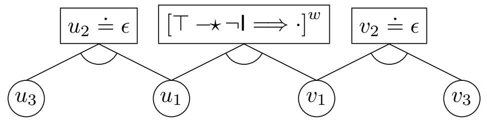
w
u_1
u_2
v_1
v_2
$w_1 \doteq [l \mapsto \cdot]$
$w_2 \doteq \epsilon$
$w_3 \doteq \epsilon$
$w_4 \doteq [l' \mapsto \cdot]$
$w_4 \not\doteq [l' \mapsto \cdot]$

By applying the rule $\mathsf{Cont}{\mapsto}{\not\doteq}$ to heap w_4, we complete the proof.

4.2 $A \star A \supset A$ where $A = \neg(\top \mathrel{-\!\!\star} \neg\mathsf{I})$

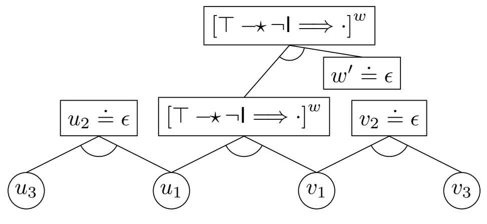
The goal formula is valid in separation logic because heaps form a partial deterministic monoid: $H_1 \circ H_2$ may be undefined (when $H_1 \# H_2$ does not hold), but if it is defined, the result is unique. In contrast, the same formula is not valid in Boolean BI, the underlying theory of separation logic, which assumes a non-deterministic monoid [21].

The proof illustrates the use of the rule EJoin in proving a non-trivial formula. After applying the logical rules to a world sequent $\cdot; \cdot \parallel [\cdot \Longrightarrow A \star A \supset A]^w$, we obtain the following graph of heaps:



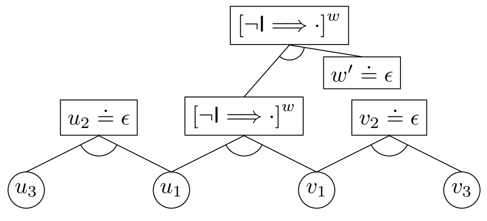
$u_2 \doteq \epsilon$
$[\top \mathrel{-\!\!\star} \neg\mathsf{I} \Longrightarrow \cdot]^w$
$v_2 \doteq \epsilon$
u_3
u_1
v_1
v_3

Since heap w has no sibling and parent heaps, we cannot apply the rule $\mathrel{-\!\!\star}\mathsf{L}$ to $\top \mathrel{-\!\!\star} \neg\mathsf{I}$ at this point. To make further progress, we apply either the rule $\mathsf{Disj}{\mathrel{-\!\!\star}}$ or the rule EJoin after creating an empty heap. If we apply the rule EJoin, we obtain the following graph:



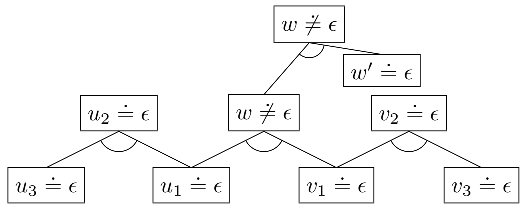
$[\top \mathrel{-\!\!\star} \neg\mathsf{I} \Longrightarrow \cdot]^w$
$w' \doteq \epsilon$
$u_2 \doteq \epsilon$
$[\top \mathrel{-\!\!\star} \neg\mathsf{I} \Longrightarrow \cdot]^w$
$v_2 \doteq \epsilon$
u_3
u_1
v_1
v_3

An application of the rule $\mathrel{-\!\!\star}\mathsf{L}$ to $\top \mathrel{-\!\!\star} \neg\mathsf{I}$ at heap w generates two new goals, and the interesting case produces $\neg\mathsf{I}$ as a true formula at the same heap (where we omit $\top \mathrel{-\!\!\star} \neg\mathsf{I}$):



$[\neg\mathsf{I} \Longrightarrow \cdot]^w$
$w' \doteq \epsilon$
$u_2 \doteq \epsilon$
$[\neg\mathsf{I} \Longrightarrow \cdot]^w$
$v_2 \doteq \epsilon$
u_3
u_1
v_1
v_3

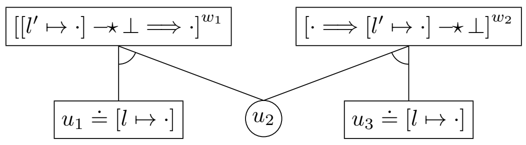
By applying the logical rules to heap w and the propagation rule $\mathsf{Prop}\,\epsilon$, we obtain the following graph:



Now we can either apply the propagation rule $\mathsf{Prop}\,\epsilon{\not\doteq}$ to heap w or use the rule NormPC to complete the proof.

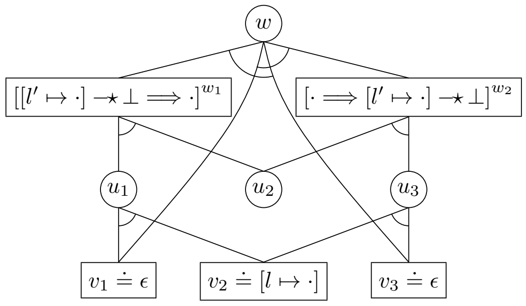
4.3 $\neg(([l' \mapsto \cdot] \mathrel{-\!\!\star} \bot) \wedge ([l \mapsto \cdot] \star \neg([l \mapsto \cdot] \mathrel{-\!\!\star} ([l' \mapsto \cdot] \mathrel{-\!\!\star} \bot))))$

The goal formula is an example of a formula that cannot be proven without an application of the rule $\mathsf{Disj}{\mathrel{-\!\!\star}}$. After applying the logical rules to a world sequent $\cdot; \cdot \parallel [\cdot \Longrightarrow C]^{w_1}$ where C is the goal formula, we obtain the following graph:



$[[l' \mapsto \cdot] \mathrel{-\!\!\star} \bot \Longrightarrow \cdot]^{w_1}$
$[\cdot \Longrightarrow [l' \mapsto \cdot] \mathrel{-\!\!\star} \bot]^{w_2}$
$u_1 \doteq [l \mapsto \cdot]$
u_2
$u_3 \doteq [l \mapsto \cdot]$

We wish to identify heaps w_1 and w_2 (because heaps u_1 and u_3 are identical), but there is no way to make further progress toward identifying w_1 and w_2 without applying the rule $\mathsf{Disj}{\mathrel{-\!\!\star}}$. Hence we apply the rule $\mathsf{Disj}{\mathrel{-\!\!\star}}$ and the propagation rule $\mathsf{Prop}{\mapsto}$ to heaps u_1 and u_3. After eliminating three new goals that are immediately provable, we obtain the following graph:



w
$[[l' \mapsto \cdot] \mathrel{-\!\!\star} \bot \Longrightarrow \cdot]^{w_1}$
$[\cdot \Longrightarrow [l' \mapsto \cdot] \mathrel{-\!\!\star} \bot]^{w_2}$
u_1
u_2
u_3
$v_1 \doteq \epsilon$
$v_2 \doteq [l \mapsto \cdot]$
$v_3 \doteq \epsilon$

By applying the rule NormPC to merge heaps w_1 and w_2, we obtain a heap sequent $[[l' \mapsto \cdot] \mathrel{-\!\!\star} \bot \Longrightarrow [l' \mapsto \cdot] \mathrel{-\!\!\star} \bot]^w$, which is easily provable.

5. Admissibility of cut

We state the admissibility of cut in $\mathbf{P_{SL}}$ as follows:

Theorem 5.1 (Admissibility of cut)**.** *If* $\Theta; \Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta, C]^w$ *and* $\Theta; \Sigma \parallel \Pi, [\Gamma, C \Longrightarrow \Delta]^w$*, then* $\Theta; \Sigma \parallel \Pi, [\Gamma \Longrightarrow \Delta]^w$.

Theorem 5.1 assumes a few properties, such as weakening and contraction, of the expression contradiction judgment $\Theta \vdash \bot$ (for which we do not provide inference rules). In particular, we assume its own admissibility of cut (where $\neg\theta$ denotes the negation of θ): $\Theta_1, \theta \vdash \bot$ and $\Theta_2, \neg\theta \vdash \bot$ imply $\Theta_1, \Theta_2 \vdash \bot$.

The first step in the proof of Theorem 5.1 is to show that it is safe to merge two arbitrary heap sequents:

Lemma 5.2. *If* $\Theta; \Sigma \parallel \Pi, [\Gamma_1 \Longrightarrow \Delta_1]^u, [\Gamma_2 \Longrightarrow \Delta_2]^v$*, then* $\Theta; [u/v]\Sigma \parallel \Pi, [\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2]^u$.

Intuitively the second world sequent inherits every heap relation from the first world sequent, so we should be able to prove the second by the same sequence of rules in the proof of the first or its subsequence.

Next we prove the contraction property for heap relations:

Proposition 5.3. *If* $\Theta; \Sigma, \sigma, \sigma \parallel \Pi$, *then* $\Theta; \Sigma, \sigma \parallel \Pi$.

The statement in Proposition 5.3 implicitly assumes that we may apply the rules Disj$\star$, Disj$-\!\!\star$, and Assoc to the same heap relation. For the rules Disj$\star$ and Disj$-\!\!\star$, the proof for such an application, which essentially has no effect, requires the rules ENew and EJoin, which are necessary for the completeness of $\mathbf{P_{SL}}$ anyway. For the rule Assoc, however, the proof for such an application requires the rule ECancel, which is unnecessary for the completeness of $\mathbf{P_{SL}}$. Hence, if we never apply the rule Assoc to the same heap relation, we may discard the rule ECancel.

To prove Theorem 5.1, we generalize its statement as follows:

Lemma 5.4. *If* $\Theta_1; \Sigma_1 \parallel \Pi_1, [\Gamma_1 \Longrightarrow \Delta_1, C]^w$ *and* $\Theta_2; \Sigma_2 \parallel \Pi_2, [\Gamma_2, C \Longrightarrow \Delta_2]^w$, *then* $\Theta_1, \Theta_2; \Sigma_1, \Sigma_2 \parallel \Pi_1 \uplus \Pi_2, [\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2]^w$.

Here $\Pi_1 \uplus \Pi_2$ denotes the result of combining heap sequents for the same heap variable. In conjunction with the contraction property for formulas, Lemma 5.4 implies Theorem 5.1.

6. Soundness and completeness of $\mathbf{P_{SL}}$

This section proves the soundness and completeness of the proof system $\mathbf{P_{SL}}$ with respect to separation logic. The proof of soundness is straightforward, whereas the proof of completeness uses several subtle properties of graphs of heaps represented by world sequents. In this section, metavariable W denotes world sequents and heap variables directly refer to heaps. For simplicity, we do not consider first-order formulas.

6.1 Soundness

The soundness property states that a derivation of a world sequent means that its semantic interpretation is self-contradictory. As a special case, we obtain Theorem 6.1:

Theorem 6.1 (Soundness). *If* $\cdot; \cdot \parallel [\cdot \Longrightarrow A]^w$, *then* $\models A$.

The key step in the proof of soundness is to show that in any inference rule of $\mathbf{P_{SL}}$, the world sequent in the conclusion is either self-contradictory in itself or semantically implies the disjunction of all world sequents in the premise. Given a stack S, let us write $[\![W]\!]_S$ for the interpretation of world sequent W according to the semantics of separation logic (which is formally defined below). We wish to prove that a derivation of W implies $\neg[\![W]\!]_S$, *i.e.*, $[\![W]\!]_S$ is self-contradictory, for any stack S. Suppose that the last inference rule in the derivation of W is not an axiom and has world sequents $W_1, \cdots, W_n$ in its premise ($n \geq 1$). By induction hypothesis, we have $\neg[\![W_1]\!]_S, \cdots, \neg[\![W_n]\!]_S$, or equivalently, $\bigwedge_{i=1,\cdots,n} \neg[\![W_i]\!]_S$. Then, by proving that $[\![W]\!]_S$ implies $\bigvee_{i=1,\cdots,n} [\![W_i]\!]_S$, we prove that $\bigwedge_{i=1,\cdots,n} \neg[\![W_i]\!]_S$ implies $\neg[\![W]\!]_S$. Now $\neg[\![W]\!]_S$ immediately follows.

Formally we define $[\![W]\!]_S$ using three auxiliary semantic functions $[\![\theta]\!]_S$, $[\![\sigma]\!]_S$, and $[\![\pi]\!]_S$, all of which follow our intuition on world sequents given in Section 3.1:

$$\begin{array}{rcl}
[\![E = E']\!]_S & = & [\![E]\!]_S = [\![E']\!]_S \\
[\![E \neq E']\!]_S & = & [\![E]\!]_S \neq [\![E']\!]_S \\
[\![w \doteq \epsilon]\!]_S & = & w = \epsilon \\
[\![w \not\doteq \epsilon]\!]_S & = & w \neq \epsilon \\
[\![w \doteq [l \mapsto E]]\!]_S & = & w = \langle [\![l]\!]_S \mapsto [\![E]\!]_S \rangle \\
[\![w \not\doteq [l \mapsto E]]\!]_S & = & w \neq \langle [\![l]\!]_S \mapsto [\![E]\!]_S \rangle \\
[\![w \doteq w_1 \circ w_2]\!]_S & = & w = w_1 \circ w_2 \\
[\![[\Gamma \Longrightarrow \Delta]^w]\!]_S & = & \bigwedge_{A \in \Gamma} (S, w) \models A \wedge \bigwedge_{B \in \Delta} (S, w) \not\models B \\
[\![\Theta; \Sigma \parallel \Pi]\!]_S & = & \bigwedge_{\theta \in \Theta} [\![\theta]\!]_S \wedge \bigwedge_{\sigma \in \Sigma} [\![\sigma]\!]_S \wedge \bigwedge_{\pi \in \Pi} [\![\pi]\!]_S
\end{array}$$

Now we prove the key step in the proof of soundness:

Lemma 6.2. *For every inference rule with the conclusion W and the premise consisting of $W_1, \cdots, W_n$, it holds that $[\![W]\!]_S$ implies $\bigvee_{i=1,\cdots,n} [\![W_i]\!]_S$ for any stack S. If $n = 0$, we have $\neg[\![W]\!]_S$.*

As a corollary, we prove that a derivation of a world sequent means that its semantic interpretation is self-contradictory.

Corollary 6.3. *If there is a derivation of a world sequent W in $\mathbf{P_{SL}}$, then $\neg[\![W]\!]_S$ holds for any stack S. For the rule* ExpCont, *we assume that $\Theta \vdash \bot$ implies $\neg[\![\Theta \vdash \bot]\!]_S$.*

Then a derivation of $\cdot; \cdot \parallel [\cdot \Longrightarrow A]^w$ implies $(S, w) \models A$:

$$\neg[\![\cdot; \cdot \parallel [\cdot \Longrightarrow A]^w]\!]_S \;=\; \neg(S, w) \not\models A \;=\; (S, w) \models A$$

Since w denotes an arbitrary heap, we have $\models A$ and Theorem 6.1 follows.

6.2 Completeness

The completeness property states that a valid formula in separation logic has a proof of its negation in $\mathbf{P_{SL}}$:

Theorem 6.4 (Completeness). *If* $\models A$, *then* $\cdot; \cdot \parallel [\cdot \Longrightarrow A]^w$.

For the proof of completeness, we weaken the rules $\star$R and $-\!\!\star$L by discarding their principal formula in the premise. The original rules are invertible and useless applications with wrong heap relations are safe, but only because the principal formula survives in the premise and we can always try different heap relations without having to backtrack. This is, however, inadequate for the proof of completeness, which must show how to actually find a right heap relation. Hence we weaken the rules $\star$R and $-\!\!\star$L to directly reflect the semantics of $\star$ and $-\!\!\star$, but also present a scheme for computing the complete set of heap relations for a given heap. We also introduce an explicit weakening rule for eliminating an atomic heap relation (which is admissible) as a new structural rule:

$$\frac{\sigma \text{ is an atomic heap relation} \quad \Theta; \Sigma \parallel \Pi}{\Theta; \Sigma, \sigma \parallel \Pi}\ \mathsf{Weaken}$$

We use the rule Weaken to eliminate all atomic heap relations at non-terminal heaps when the propagation rules can produce no more new heap relations. As explained in Section 5, we do not use the rule ECancel.

Our proof of Theorem 6.4 uses three new concepts: *canonical world sequents*, *disjunctive derivation states*, and *conjunctive proof goals*.

A canonical world sequent Z is a special form of a world sequent such that if $\neg[\![Z]\!]_S$ holds for any stack S, we can construct its derivation using only the rules $\bot$L, ExpCont, and the heap contradiction rules. (In Proposition 6.12, we introduce a class of world sequents that are shown to be canonical.)

A disjunctive derivation state Ψ for a world sequent W is a set of world sequents that constitute all the leaves in a partial derivation of W. That is, a disjunctive derivation state $\Psi = \{W_1, \cdots, W_n\}$ for a world sequent W means that there is a partial derivation of the following form:

$$\frac{\begin{array}{c} W_1 \quad \cdots \quad W_n \\ \ddots \vdots \therefore \end{array}}{W}$$

We use a reduction judgment $\Psi \overset{R}{\mapsto} \Psi'$ to mean that such a partial derivation expands to another partial derivation with disjunctive derivation state Ψ' by an application of the logical or structural rule R to some world sequent W_i ($1 \leq i \leq n$). That is, we have $\Psi' = \Psi - \{W_i\} \cup \{W_i^1, \cdots, W_i^m\}$ with:

$$\begin{array}{c} W_1 \quad \dfrac{W_i^1 \quad \cdots \quad W_i^m}{\cdots \quad W_i \quad \cdots}R \quad W_n \\ \ddots \vdots \iddots \\ \hline W \end{array}$$

We write $\Psi \mapsto^* \Psi'$ for the reflexive and transitive closure of $\mapsto$. For a stack S, we define the interpretation $[\![\Psi]\!]_S = \bigvee_{W_i \in \Psi} [\![W_i]\!]_S$.

A conjunctive proof goal Ω is a set of disjunctive derivation states for a common world sequent. Given a logical or structural rule R, we use a reduction judgment $\Omega \overset{R}{\leadsto} \Omega'$ to mean that we can generate Ω' by applying the rule R to some disjunctive derivation state Ψ in Ω. That is, we have $\Omega' = \Omega - \{\Psi\} \cup \{\Psi'_1, \cdots, \Psi'_n\}$ and $\Psi \overset{R}{\mapsto} \Psi'_i$ for $i = 1, \cdots, n$. If R is the rule $\star$R or $-\!\!\star$L, we have $n \geq 1$ and produce each Ψ'_i by focusing on the same formula in the same heap sequent in the same world sequent in Ψ. For all the other rules, we have $n = 1$ and replace Ψ by Ψ'_1. We write $\Omega \leadsto^* \Omega'$ for the reflexive and transitive closure of $\leadsto$. For a stack S, we define the interpretation $[\![\Omega]\!]_S = \bigwedge_{\Psi_i \in \Omega} [\![\Psi_i]\!]_S$.

In order to prove Theorem 6.4, assume a goal formula A such that $\models A$. We aim to build a sequence of conjunctive proof goals $\Omega^1, \cdots, \Omega^N$ such that:

- $\Omega^1 = \{\{W\}\}$ where $W = \cdot; \cdot \parallel [\cdot \Longrightarrow A]^w$.
- $\Omega^i \overset{R_i}{\leadsto} \Omega^{i+1}$ for $i = 1, \cdots, N-1$ where R_i is a logical or structural rule.
- $[\![\Omega^{i+1}]\!]_S$ implies $[\![\Omega^i]\!]_S$ for $i = 1, \cdots, N-1$ and any stack S.
- Ω^N contains only canonical world sequents.

With such a sequence of conjunctive proof goals, we can build a derivation of W as follows:

- Since we have $\models A$, we have
$$(S, w) \models A = \neg(S, w) \not\models A = \neg[\![W]\!]_S = \neg[\![\Omega^1]\!]_S$$
for any stack S.
- Since $[\![\Omega^N]\!]_S$ implies $[\![\Omega^1]\!]_S$, we have $\neg[\![\Omega^N]\!]_S$.
- Since every disjunctive derivation state in Ω^N contains only canonical world sequents, we may write $\Omega^N = \bigcup_j \Psi_j$ and $\Psi_j = \bigcup_k Z_k^j$.
- Since we have $\neg[\![\Omega^N]\!]_S = \neg \bigwedge_j [\![\Psi_j]\!]_S = \bigvee_j \neg[\![\Psi_j]\!]_S$, there exists a disjunctive derivation state Ψ_j such that $\neg[\![\Psi_j]\!]_S$ holds.
- Since we have $\neg[\![\Psi_j]\!]_S = \neg \bigvee_k [\![Z_k^j]\!]_S = \bigwedge_k \neg[\![Z_k^j]\!]_S$, we have $\neg[\![Z_k^j]\!]_S$ for each k.
- Since $\Omega^1 \leadsto^* \Omega^N$, we have $\{W\} \mapsto^* \Psi_j$.
- Since we have $\neg[\![Z_k^j]\!]_S$ for each k, there is a derivation of Z_k^j for each k by the definition of canonical world sequents.
- By combining $\{W\} \mapsto^* \Psi_j$ and the derivation of Z_k^j for each k, we obtain a derivation of W.

Below we explain how to build such a sequence of conjunctive proof goals.

6.2.1 Completeness of the invertible rules

Suppose that a world sequent in Ω^i contains a formula to which we can apply a rule R other than $\bot$L, $\star$R, and $-\!\!\star$L. By applying the rule R, we obtain $\Omega^i \overset{R}{\leadsto} \Omega^{i+1}$. By Corollary 6.6, $[\![\Omega^{i+1}]\!]_S$ implies $[\![\Omega^i]\!]_S$ for any stack S. In this way, we can eliminate all such formulas without losing completeness.

Proposition 6.5. *Except for the rules $\bot$L, ExpCont, $\star$R, $-\!\!\star$L, and Weaken, every logical or structural rule with the conclusion W and the premise consisting of $W_1, \cdots, W_n$ is invertible in that $\bigvee_{i=1,\cdots,n} [\![W_i]\!]_S$ implies $[\![W]\!]_S$ for any stack S.*

Corollary 6.6 (Completeness of the invertible rules)**.** *For such an invertible rule R and any stack S,*

if $\{W\} \overset{R}{\mapsto} \Psi$, then $[\![\Psi]\!]_S$ implies $[\![\{W\}]\!]_S$.

6.2.2 Completeness of the rules $\star$R, $-\!\!\star$L, and Weaken

We now show how to recover the completeness of the rules $\star$R and $-\!\!\star$L. We introduce several notations in order to concisely describe properties of graphs of heaps:

- $w \nearrow u$ means that there is a sequence of zero or more child-parent relations from heap w to heap u in the graph: $w = w_0$, $w_1 \doteq w_0 \circ w'_0, \cdots, w_n \doteq w_{n-1} \circ w'_{n-1}$, and $w_n = u$ for $n \geq 0$. Hence, if $w \neq u$, heap w is a descendant of heap u, or equivalently, heap u is an ancestor of heap w. Note that we allow $w \nearrow w$.
- $w\!\uparrow$ means that w is a terminal heap, *i.e.*, there is no heap relation $w \doteq u \circ v$.
- $T(w)$ denotes the set of terminal descendants of heap w, *i.e.*, $T(w) = \{v \mid v\!\uparrow \text{ and } v \nearrow w\}$.

We assume that heap relations in every world sequent induce not only a graph of heaps but also a special empty heap w_ϵ with heap relation $w_\epsilon \doteq \epsilon$ that is separate from the graph. This assumption is safe because we can always generate such a special empty heap with the rule ENew if there is none, and combine multiple empty heaps with the rule NormEmpty if there are many. We classify world sequents according to the property of graphs of heaps induced by their heap relations (without considering its special empty heap w_ϵ) as follows:

1. Elementary: if $w \doteq w_1 \circ w_2$, then $T(w_1) \cap T(w_2) = \varnothing$.
2. Rooted: elementary and there is a root heap w such that $v \nearrow w$ for every heap v.
3. Consistent: rooted and if $w \doteq u_1 \circ u_2$ and $w \doteq v_1 \circ v_2$, then $T(u_1) \cup T(u_2) = T(v_1) \cup T(v_2)$.
4. Full: consistent and for any non-empty set S of terminal heaps, there exists at least one heap w with $T(w) = S$.
5. $\star$-ready for heap w: full and for any pair of non-empty sets S_1 and S_2 of terminal heaps such that $S_1 \cap S_2 = \varnothing$ and $S_1 \cup S_2 = T(w)$, there exist heaps w_1 and w_2 such that $w \doteq w_1 \circ w_2$ with $T(w_1) = S_1$ and $T(w_2) = S_2$.
6. $-\!\!\star$-ready for heap w: full and for any pair of non-empty sets S_1 and S_2 of terminal heaps such that $T(w) \cap S_1 = \varnothing$ and $T(w) \cup S_1 = S_2$, there exist heaps w_1 and w_2 such that $w_2 \doteq w \circ w_1$ with $T(w_1) = S_1$ and $T(w_2) = S_2$.
7. Saturated: full and applications of the propagation rules produce no more new heap relations.
8. Sanitized: full and non-terminal heaps have no atomic heap relations.
9. Normalized: sanitized with no empty heaps and for any non-empty set S of terminal heaps, there exists a unique heap w with $T(w) = S$.
10. Expanded: obtained by applying only the logical rules except $\star$R and $-\!\!\star$L to some consistent world sequent.

Suppose that a world sequent in Ω^i contains a false formula $A \star B$ or a true formula $A -\!\!\star B$ about heap w. By converting it into normalized world sequents that are also $\star$-ready or $-\!\!\star$-ready for heap w according to Corollary 6.10, we obtain $\Omega^i \leadsto^* \Omega^{i+1}$ such that $[\![\Omega^{i+1}]\!]_S$ implies $[\![\Omega^i]\!]_S$ for any stack S. By Proposition 6.11, we can apply the rule $\star$R or $-\!\!\star$L to obtain Ω^{i+2} with $\Omega^{i+1} \overset{\star\mathsf{R}}{\leadsto} \Omega^{i+2}$ or $\Omega^{i+1} \overset{-\!\star\mathsf{L}}{\leadsto} \Omega^{i+2}$ such that $[\![\Omega^{i+2}]\!]_S$ implies $[\![\Omega^{i+1}]\!]_S$ for any stack S. In this way, we can eliminate every false formula $A \star B$ or true formula $A \mathrel{-\!\!\star} B$ without losing completeness.

Lemma 6.7. *For a world sequent W of a particular kind, there exists a world sequent W' of another kind such that:*
1) $\{W\} \mapsto^* \{W'\}$ by applying only the structural rules;
2) $[\![\{W'\}]\!]_S$ implies $[\![\{W\}]\!]_S$ for any stack S,
where one of the following holds:
1. *W is expanded and W' is rooted;*
2. *W is rooted (generated in step 1) and W' is consistent;*
3. *W is consistent and W' is full;*
4. *W is full and W' is $\star$-ready for a given heap w;*

4'. *W is full and W' is $\mathrel{-\!\!\star}$-ready for a given heap w;*

8. *W is sanitized and $\star$-ready ($\mathrel{-\!\!\star}$-ready) for a given heap w, and W' is normalized and $\star$-ready ($\mathrel{-\!\!\star}$-ready) for heap w.*

Lemma 6.8. *For a world sequent W of a particular kind, there exists a disjunctive derivation state Ψ such that:*
1) $\{W\} \mapsto^* \Psi$ by applying only the propagation rules;
2) $[\![\Psi]\!]_S$ implies $[\![\{W\}]\!]_S$ for any stack S,
where one of the following holds:

5. *W is $\star$-ready for a given heap w, and every world sequent in Ψ is saturated as well as $\star$-ready for heap w.*
6. *W is $\mathrel{-\!\!\star}$-ready for a given heap w, and every world sequent in Ψ is saturated as well as $\mathrel{-\!\!\star}$-ready for heap w.*

Lemma 6.9 (Completeness of the rule Weaken). *For any saturated world sequent W, there exists a sanitized world sequent W' such that:*
1) $\{W\} \mapsto^* \{W'\}$ by applying only the rule Weaken*;*
2) $[\![\{W'\}]\!]_S$ implies $[\![\{W\}]\!]_S$ for any stack S.

Corollary 6.10. *For any expanded world sequent W and heap w, there exists a disjunctive derivation state Ψ such that:*
1) $\{W\} \mapsto^* \Psi$ by applying only the structural rules;
2) Ψ contains only normalized world sequents that are also $\star$-ready or $\mathrel{-\!\!\star}$-ready for heap w;
3) $[\![\Psi]\!]_S$ implies $[\![\{W\}]\!]_S$ for any stack S.

Proposition 6.11 (Completeness of the rules $\star$R and $\mathrel{-\!\!\star}$L).
Consider a normalized world sequent W that is also $\star$-ready for heap w. Suppose that we obtain $\{W\} \overset{\star\mathsf{R}}{\mapsto} \Psi_i$ by applying the rule $\star$R to a false formula $A \star B$ about heap w for each heap relation $w \doteq w_i \circ w'_i$ $(i = 1, \cdots, n)$, and $\{W\} \overset{\star\mathsf{R}}{\mapsto} \Psi_\epsilon$ by applying the rule $\star$R to the same formula for another heap relation $w \doteq w \circ w_\epsilon$. Then a conjunctive proof goal $\Omega = \{\Psi_1, \cdots, \Psi_n, \Psi_\epsilon\}$ satisfies:
1) $\{\{W\}\} \overset{\star\mathsf{R}}{\leadsto} \Omega$;
2) $[\![\Omega]\!]_S$ implies $[\![\{\{W\}\}]\!]_S$ for any stack S;
3) No world sequent in Ω contains the false formula $A \star B$ about heap w;
4) Every world sequent in Ω is still normalized and thus consistent.
Similarly for the rule $\mathrel{-\!\!\star}$L where we use each heap relation $w_i \doteq w \circ w'_i$.

6.2.3 Completeness of the heap contradiction rules

Suppose that after eliminating all formulas other than $\bot$, we obtain a conjunctive proof goal Ω^i which contains only expanded world sequents. By Corollary 6.10, we can obtain $\Omega^i \leadsto^* \Omega^{i+1}$ such that every world sequent in Ω^{i+1} is normalized and $[\![\Omega^{i+1}]\!]_S$ implies $[\![\Omega^i]\!]_S$ for any stack S. Proposition 6.12 shows that Ω^{i+1} contains only canonical world sequents.

Proposition 6.12 (Completeness of the heap contradiction rules).
A normalized world sequent W with no formulas other than $\bot$ is canonical. That is, if $\neg[\![W]\!]_S$ holds for any stack S, we can construct its derivation using only the rules $\bot$L, ExpCont*, and the heap contradiction rules. For the rule* ExpCont*, we assume that $\neg[\![\Theta \vdash \bot]\!]_S$ implies $\Theta \vdash \bot$.*

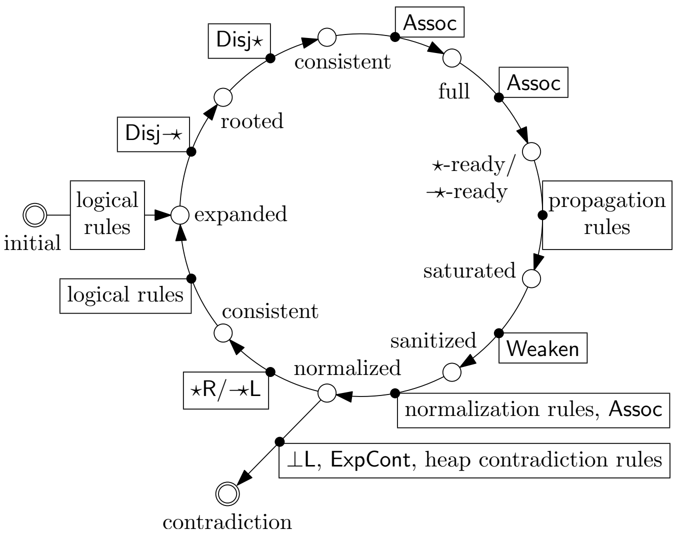


Figure 4. Proof search strategy based on the proof of Theorem 6.4

Thus we conclude the proof of Theorem 6.4.

6.3 Proof search strategy

Figure 4 shows the proof search strategy based on the proof of Theorem 6.4. Given a world sequent of the form $\cdot;\cdot \parallel [\cdot \Longrightarrow A]^w$, we repeatedly apply the logical rules other than the rules $\star$R and $\mathrel{-\!\!\star}$L to obtain expanded world sequents according to Corollary 6.6 (from initial to expanded). Then we apply a series of structural rules to obtain normalized world sequents according to Corollary 6.10 (from expanded to normalized). If no formulas other than $\bot$ remain, we attempt to generate a logical contradiction according to Proposition 6.12 (from normalized to contradiction). Otherwise we apply the rule $\star$R or $\mathrel{-\!\!\star}$L to obtain consistent world sequents according to Proposition 6.11 (from normalized to consistent). By repeatedly applying the logical rules other than the rules $\star$R and $\mathrel{-\!\!\star}$L again, we obtain another set of expanded world sequents according to Corollary 6.6 (from consistent to expanded). In this way, we can eventually decompose all formulas other than $\bot$ and complete the proof search.

As it directly translates to a proof search strategy, the proof of Theorem 6.4 agrees with the result that propositional separation logic is decidable [9]. Since first-order separation logic is undecidable [5], the proof system $\mathbf{P_{SL}}$ becomes undecidable in the presence of the rule $\exists$R. We also remark that in the presence of propositional variables, even propositional separation logic is undecidable [6].

6.4 Validity of formulas in $\mathbf{P_{SL}}$

As established by Theorems 6.1 and 6.4, the notion of validity in $\mathbf{P_{SL}}$ means that a formula is true (or an assumption of its falsehood leads to a logical contradiction) at an arbitrary heap about which nothing is known. An important implication of assuming such an arbitrary heap is that we in effect assume *an arbitrary world of heaps*, since even the relationship with an external heap can be thought of as a property of the arbitrary heap, which is assumed to be unknown. In particular, we may not assume the existence of a singleton heap with location L even if the goal formula contains a points-to relation $[\mathsf{L} \mapsto E]$.

As an example, consider a formula

$$A \;=\; \mathsf{I} \supset \neg([\mathsf{L} \mapsto E] -\!\!\star \neg[\mathsf{L} \mapsto E])$$

which states that an empty heap combined with a heap satisfying $[\mathsf{L} \mapsto E]$ cannot be a heap where $[\mathsf{L} \mapsto E]$ is false. According to the semantics of separation logic in Section 2, this formula is valid *but only provided that a singleton heap satisfying* $[\mathsf{L} \mapsto E]$ *is known to exist.* The existence of such a singleton heap, however, is an assumption that can be justified only by inspecting the formula at the meta-logical level. Since $\mathbf{P_{SL}}$ assumes an arbitrary world of heaps without knowing the existence of such a singleton heap, it fails to prove the above formula.

If we are to show the validity of a formula by exploiting prior knowledge on the world of heaps, we should prove an extended formula that incorporates its meta-logical property as well. For example, in order to show the validity of the above formula A when the existence of a singleton heap satisfying $[\mathsf{L} \mapsto E]$ is already known, we should instead prove an extended formula

$$[\mathsf{L} \mapsto E] \star \top \supset \neg(\neg A \star \top)$$

which states that if the world of heaps contains a heap satisfying $[\mathsf{L} \mapsto E]$, there cannot exist a heap where A is false, *i.e.*, A is true at any heap. An attempt to prove the extended formula produces a world sequent

$$\begin{array}{lll} & & [\cdot \Longrightarrow \cdot]^{w}, \\ & & [[\mathsf{L} \mapsto E] \Longrightarrow \cdot]^{u_1}, \\ & & [\top \Longrightarrow \cdot]^{u_2}, \\ & w \doteq u_1 \circ u_2, & [\cdot \Longrightarrow A]^{v_1}, \\ \cdot; & w \doteq v_1 \circ v_2 \quad \| & [\top \Longrightarrow \cdot]^{v_2} \end{array}$$

which essentially expresses that formula A is false at an arbitrary heap when there exists a heap satisfying $[\mathsf{L} \mapsto E]$. A proof of this world sequent complies with the validity of formula A in separation logic.

7. Discussion

Our prototype implementation of $\mathbf{P_{SL}}$ (without first-order formulas) is based on the proof of Theorem 6.4, but with a few changes. In particular, it internally uses a different type of normalized world sequents which maintain a unique heap corresponding to each nonempty set of terminal heaps, but permit unknown relations between heaps. The decision is based on the observation that it is the rules $\mathsf{Disj}\star$ and $\mathsf{Disj}-\!\!\star$ (for eliminating unknown relations between heaps) that contributes the most to the complexity of graphs of heaps. Thus it selectively applies the rules $\mathsf{Disj}\star$ and $\mathsf{Disj}-\!\!\star$ only when it cannot complete the proof search otherwise.

Our experience with the prototype implementation of $\mathbf{P_{SL}}$ shows that it allows us to incorporate new logical connectives and predicates in a principled way without having to introduce additional structural rules. As an example, consider an overlapping conjunction $A \uplus B$ by Hobor and Villard [17] which can be defined in the framework of $\mathbf{P_{SL}}$ as follows:

- $A \uplus B$ is true at heap w iff. $w \doteq w_1 \circ v_2$, $w \doteq v_1 \circ w_2$, $w_1 \doteq v_1 \circ u$, $w_2 \doteq u \circ v_2$, and A is true at heap w_1 and B is true at heap w_2 for *some* heaps w_1, w_2, v_1, v_2, and u.
- $A \uplus B$ is false at heap w iff. $w \doteq w_1 \circ v_2$, $w \doteq v_1 \circ w_2$, $w_1 \doteq v_1 \circ u$, and $w_2 \doteq u \circ v_2$ implies that A is false at heap w_1 or that B is false at heap w_2 for *any* heaps w_1, w_2, v_1, v_2, and u.

We directly translate this definition into two inference rules for $\uplus$:

$$\dfrac{\begin{array}{l} \textit{fresh}\ w_1, w_2, v_1, v_2, u \\ \begin{array}{lll} & & [A \Longrightarrow \cdot]^{w_1}, \\ & w \doteq w_1 \circ v_2, & [B \Longrightarrow \cdot]^{w_2}, \\ & w \doteq v_1 \circ w_2, & [\cdot \Longrightarrow \cdot]^{v_1}, \\ & w_1 \doteq v_1 \circ u, & [\cdot \Longrightarrow \cdot]^{v_2}, \\ \Theta; \Sigma, & w_2 \doteq u \circ v_2 \quad \| \ \Pi, [\Gamma \Longrightarrow \Delta]^{w}, & [\cdot \Longrightarrow \cdot]^{u} \end{array} \end{array}}{\Theta; \Sigma \ \| \ \Pi, [\Gamma, A \uplus B \Longrightarrow \Delta]^{w}}\ \uplus\mathsf{L}$$

$$\dfrac{\begin{array}{ll} & \begin{array}{ll} & [\Gamma \Longrightarrow \Delta, A \uplus B]^{w}, \\ \Theta; \Sigma \ \| \ \Pi, & [\Gamma_1 \Longrightarrow \Delta_1, A]^{w_1}, \\ & [\Gamma_2 \Longrightarrow \Delta_2]^{w_2} \end{array} \\ w \doteq w_1 \circ v_2, & \\ w \doteq v_1 \circ w_2, & \\ w_1 \doteq v_1 \circ u, & \begin{array}{ll} & [\Gamma \Longrightarrow \Delta, A \uplus B]^{w}, \\ \Theta; \Sigma \ \| \ \Pi, & [\Gamma_1 \Longrightarrow \Delta_1]^{w_1}, \\ & [\Gamma_2 \Longrightarrow \Delta_2, B]^{w_2} \end{array} \\ \{\, w_2 \doteq u \circ v_2 \ \} \subset \Sigma & \end{array}}{\begin{array}{ll} & [\Gamma \Longrightarrow \Delta, A \uplus B]^{w}, \\ \Theta; \Sigma \ \| \ \Pi, & [\Gamma_1 \Longrightarrow \Delta_1]^{w_1}, \\ & [\Gamma_2 \Longrightarrow \Delta_2]^{w_2} \end{array}}\ \uplus\mathsf{R}$$

Note that we obtain the rules $\uplus\mathsf{L}$ and $\uplus\mathsf{R}$ exactly in the same way that we derive the rules $\star\mathsf{L}$ and $\star\mathsf{R}$ from the interpretation of multiplicative conjunction $\star$. The only difference is that we create five fresh heaps in the rule $\uplus\mathsf{L}$ and try to detect a subgraph consisting of six existing heaps in the rule $\uplus\mathsf{R}$. Equally important is that we need no additional structural or heap contradiction rules because overlapping conjunction does not require new forms of heap relations. Thus, in principle, it is relatively easy to incorporate overlapping conjunction into our prototype implementation of $\mathbf{P_{SL}}$. Overall we may think of $\mathbf{P_{SL}}$ as a highly extensible proof system for separation logic.

8. Related work

8.1 Automated verification tools based on separation logic

Separation logic has been the basis for a number of automated verification tools targeting programs using mutable data structures. The first such tool is Smallfoot by Berdine *et al.* [3] which aims to test the feasibility of automated verification using separation logic. To achieve full automation, it permits no pointer arithmetic and verifies only shape properties of linked lists and trees. Space Invader by Distefano *et al.* [11] permits pointer arithmetic by integrating the abstract interpretation method into the symbolic execution method in [4]. THOR by Magill *et al.* [23] is an extension of Space Invader which is capable of tracking the length of linked lists. SLAyer by Berdine *et al.* [1] is another extension of Space Invader which uses higher-order predicates to express common properties of nodes in linked lists. The use of higher-order predicates enables SLAyer to verify shape properties of composite linked lists such as linked lists of circular linked lists.

There are also several tools supporting arbitrary data structures. HIP by Nguyen and Chin [25] allows users to specify invariants on arbitrary data structures in terms of inductive predicates. Since checking these invariants usually relies on basic properties of inductive predicates that are easy to prove but difficult to discover automatically, HIP requires users to explicitly state such properties in the form of lemmas, which are automatically proven and then applied as necessary. Similarly to HIP, VeriFast by Jacobs *et al.* [19] relies on user-supplied inductive predicates and lemmas. Unlike HIP, however, VeriFast requires users to provide proofs of these lemmas and specify when to apply them. jStar by Distefano and Parkinson [12] is an extension of Space Invader which exploits user-supplied abstraction rules in order to support arbitrary data structures. Its distinguishing feature is the ability to infer loop invariants automatically. Xisa by Chang and Rival [10] takes a different approach by indirectly specifying invariants on data structures with validation code. Xisa analyzes validation code to extract inductive predicates for describing invariants as well as lemmas for describing their basic properties. Since validation code can be written in common programming languages, users of Xisa do not need the expertise to specify invariants of interest in terms of inductive predicates.

All these tools use as their logical foundation not full separation logic but only its decidable fragment by Berdine *et al.* [2], which does not include separating implication $-\!\!\star$. As shown by Ishtiaq and O'Hearn [18], lack of separating implication implies no support for backward reasoning by weakest precondition generation for those programs performing heap assignments or allocation. As a result, these tools allow only forward reasoning based on symbolic execution as in [4] and do not demonstrate the full potential of separation logic in program verification.

8.2 Proof search in full separation logic

Despite the practical importance of separating implication, proof search in full separation logic has not drawn much attention from researchers. Calcagno *et al.* [8] present a translation from propositional separation logic to first-order logic (with only propositional connectives and no multiplicative connectives) for which a decision procedure already exists. The labelled tableau calculus for separation logic by Galmiche and Méry [15] supports both separating conjunction and separating implication. Similarly to our proof system $\mathbf{P_{SL}}$, their calculus combines both syntactic (tableau) and semantic (labelled) formulations and uses labels to directly refer to heaps. Although it is shown to be sound and complete, their calculus does not give rise to a proof search strategy. Specifically, in order to check that all branches in a tableau are logically or structurally inconsistent, we need two semantic functions, a measure and an interpretation, for each branch. Their calculus, however, does not explain how to construct such semantic functions for each branch and it is not clear how to extract a concrete proof search strategy.

The closest proof system to ours is the nested sequent calculus $\mathbf{S_{BBI}}$ for Boolean BI by Park *et al.* [26], which inspired the overall design of $\mathbf{P_{SL}}$. Similarly to world sequents in $\mathbf{P_{SL}}$, sequents in $\mathbf{S_{BBI}}$ use a truth context consisting of true formulas and a falsehood context consisting of false formulas, and both systems are based on the principle of proof by contradiction. Because of the similarity in syntactic formulations, their approach to dealing with separating conjunction and separating implication in $\mathbf{S_{BBI}}$ equally applies to our setting for $\mathbf{P_{SL}}$, which is not surprising considering that separation logic is just an instance of Boolean BI with additional restrictions on the semantic structure. The structural rules of $\mathbf{P_{SL}}$, however, are specific to separation logic and are designed independently of $\mathbf{S_{BBI}}$. Since $\mathbf{S_{BBI}}$ allows propositional variables, we may use its theorem prover as a supplementary system for our implementation of $\mathbf{P_{SL}}$.

For theorem provers based on the decidable fragment of separation logic by Berdine *et al.* [2] (without separating implication), see, for example, SeLoger [16] and SLP [24]. For an isomorphism between (intuitionistic) separation logic and implicit dynamic frames, see [27].

9. Conclusion

We have presented a proof system $\mathbf{P_{SL}}$ for full separation logic with separating implication. Considering the potential benefit of separating implication, we envision that program verification systems in the future will provide separating implication and support backward reasoning by weakest precondition generation for their scalability in program verification. We also envision that proof assistants can interface with theorem provers for separation logic and provide a powerful automation tactic for dealing with logical connectives from separation logic. When extended with inductively defined predicates, $\mathbf{P_{SL}}$ may serve as a practical foundation for such systems.

Acknowledgments

We are grateful to the anonymous reviewers for their helpful comments and Zhé Hóu for a comment on the completeness of $\mathbf{P_{SL}}$ that led to the discussion in Section 6.4. This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Science, ICT & Future Planning (MSIP)/National Research Foundation of Korea (NRF) (Grant NRF-2008-0062609) and Mid-career Researcher Program through NRF funded by the MEST (2010-0022061).

References

[1] Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano, Peter W. O'Hearn, Thomas Wies, and Hongseok Yang. Shape analysis for composite data structures. In *Proc. CAV*, pages 178–192, 2007.

[2] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A decidable fragment of separation logic. In *Proc. FSTTCS*, pages 97–109, 2004.

[3] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *Proc. FMCO*, pages 115–137, 2005.

[4] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Symbolic execution with separation logic. In *Proc. APLAS*, pages 52–68, 2005.

[5] Rémi Brochenin, Stéphane Demri, and Etienne Lozes. On the almighty wand. *Information and Computation*, 211:106–137, 2012.

[6] James Brotherston and Max Kanovich. Undecidability of propositional separation logic and its neighbours. In *Proc. LICS*, pages 130–139, 2010.

[7] Cristiano Calcagno and Dino Distefano. Infer: an automatic program verifier for memory safety of C programs. In *Proceedings of the Third international conference on NASA Formal methods*, pages 459–465, 2011.

[8] Cristiano Calcagno, Philippa Gardner, and Matthew Hague. From separation logic to first-order logic. In *Proc. FOSSACS*, pages 395–409, 2005.

[9] Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 108–119, 2001.

[10] Bor-Yuh Evan Chang and Xavier Rival. Relational inductive shape analysis. In *Proc. POPL*, pages 247–260, 2008.

[11] Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. A local shape analysis based on separation logic. In *Proc. TACAS*, pages 287–302, 2006.

[12] Dino Distefano and Matthew J. Parkinson. jStar: towards practical verification for Java. In *Proc. OOPSLA*, pages 213–226, 2008.

[13] Robert Dockins, Aquinas Hobor, and Andrew W. Appel. A fresh look at separation algebras and share accounting. In *Proc. APLAS*, pages 161–177, 2009.

[14] Kamil Dudka, Petr Müller, Petr Peringer, and Tomáš Vojnar. Predator: a tool for verification of low-level list manipulation. In *Proc. TACAS*, pages 627–629, 2013.

[15] Didier Galmiche and Daniel Méry. Tableaux and resource graphs for separation logic. *Journal of Logic and Computation*, 20:189–231, 2010.

[16] Christoph Haase, Samin Ishtiaq, Joël Ouaknine, and Matthew J. Parkinson. SeLoger: A tool for graph-based reasoning in separation logic. In *Proc. CAV*, pages 790–795, 2013.

[17] Aquinas Hobor and Jules Villard. The ramifications of sharing in data structures. In *Proc. POPL*, pages 523–536, 2013.

[18] Samin S. Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. POPL*, pages 14–26, 2001.

[19] Bart Jacobs, Jan Smans, and Frank Piessens. VeriFast: Imperative programs as proofs. In *Proc. VSTTE*, pages 59–68, 2010.

[20] Neelakantan R. Krishnaswami. Reasoning about iterators with separation logic. In *Proc. SAVCBS*, pages 83–86, 2006.

[21] Dominique Larchey-Wendling and Didier Galmiche. The undecidability of boolean BI through phase semantics. In *Proc. LICS*, pages 140–149, 2010.

[22] Toshiyuki Maeda, Haruki Sato, and Akinori Yonezawa. Extended alias type system using separating implication. In *Proc. TLDI*, pages 29–42, 2011.

[23] Stephen Magill, Josh Berdine, Edmund M. Clarke, and Byron Cook. Arithmetic strengthening for shape analysis. In *Proc. SAS*, pages 419–436, 2007.

[24] Juan Antonio Navarro Pérez and Andrey Rybalchenko. Separation logic + superposition calculus = heap theorem prover. In *Proc. PLDI*, pages 556–566, 2011.

[25] Huu Hai Nguyen and Wei-Ngan Chin. Enhancing program verification with lemmas. In *Proc. CAV*, pages 355–369, 2008.

[26] Jonghyun Park, Jeongbong Seo, and Sungwoo Park. A theorem prover for Boolean BI. In *Proc. POPL*, pages 219–232, 2013.

[27] Matthew J. Parkinson and Alexander J. Summers. The relationship between separation logic and implicit dynamic frames. In *Proc. ESOP*, pages 439–458, 2011.

[28] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. LICS*, pages 55–74, 2002.

[29] Hongseok Yang. An example of local reasoning in BI pointer logic: the Schorr-Waite graph marking algorithm. In *Proceedings of the 1st Workshop on Semantics, Program Analysis, and Computing Environments for Memory Management*, pages 41–68, 2001.