A Complete Proof System for Basic Symbolic Heaps with Permissions

S. Demri[1], E. Lozes[2], and D. Lugiez[3]

[1]LSV, CNRS, ENS Paris-Saclay, Université Paris-Saclay, France
[2]I3S, CNRS, Université Côte d'Azur, France
[3]Aix-Marseille Univ, LIF, CNRS, Marseille, France

Abstract. We design a proof system for symbolic heaps with permissions restricted to points-to predicates. The calculus is parameterised by the permission theory and we establish soundness and completeness. A strategy with optimal computational properties is also presented. This is a very preliminary work that is intended to be much further extended.

1 Introduction

Concurrent heap manipulating programs may contain subtle bugs that are hard to detect by testing. Formal proofs of such programs made significant progress in the last ten years, particularly due to the developments of logical frameworks and tools based on concurrent separation logic. One of the strengths of concurrent separation logic is to support reasoning about resources shared among execution units (threads, actors, etc.). In the simplest cases, resources are shared without races, and a mechanism called *permissions* can be used to track which memory regions an execution unit can access at a given time of its execution, and the kind of accesses it can perform: a permission can be thought of as a "quantity of ownership" that gets attached to each cell of the heap. This quantity prescribes whether write accesses are allowed on this cell or not and how such a write access may be restored in the future.

Permissions have been integrated successfully in several tools among which VCC [8], VeriFast [12], Dafny [15], Hip/Sleek [11], Viper [16] or Heap-Hop [17]. Several models of permissions have been considered, among which fractional permissions [5], token-based permissions [4], combinations of the two, and binary tree shares [10]. In particular, a COQ library of certified decision procedures for the theory of binary shares is developed in [13].

In a different line, since the early works on the foundations of Smallfoot [2], several proof systems have been proposed for dialects of separation logic: first for symbolic heaps with lists [1,3], and later on for arbitrary recursive predicates, often based on cyclic proofs [6,7]. Such proof systems were important for the development of separation logic, because they guided proof search heuristics that are easy to understand, they helped integrate separation logic in interactive theorem provers, and in the future, they could be used by automatic tools as proof certificates. Moreover, some of these proof systems proceed by "subtracting"

heap assertions, which convey some heuristics for solving frame inference and biabduction.

In this paper, we look for a proof system in which one may derive valid entailments between symbolic heaps with permissions. For this, we follow closely the methodology we developed in our recent study on the decidability and complexity of satisfiability and entailment for symbolic heaps with permissions and lists [9]. However, instead of providing ad-hoc decision procedures as performed in [9], we design a sequent-style proof system for valid entailments. In this paper, the preliminary results we present concern *basic* symbolic heaps, that is symbolic heaps without lists. Our contributions are the following.

- We introduce a proof system for entailment among symbolic heaps with permissions, for now without lists, that proceeds by "subtracting" heap assertions in the same manner as several other proof systems for permission-free separation logics.
- We establish the soundness and the completeness of this system, up to the price of imposing some canonicity of the heap assertions before applying our subtraction rules.
- We present a proof search strategy that remains complete and optimal from the complexity point of view.

Related work. Bach Le and Hobor recently introduced a proof system for symbolic heaps with permissions for arbitrary recursive predicates [14]. This proof system differs from ours in two ways: first, it is based on a notion of "predicate multiplication" $\pi \cdot P$ between a permission π and an arbitrary formula P that we did not consider in our model. Second, and most importantly for us, it does not address the issue of completeness.

Outline In Section 2, we recall the definition of symbolic heaps and permissions. In Section 3, we define our proof system for logical entailments, together with an auxiliary rewriting system that helps normalise symbolic heaps, as needed by some proof rules. In Section 4, we establish the soundness and the completeness of our proof system.

2 Separation logic with permissions

In this section, we recall the definition of separation logic with permissions.

2.1 Syntax

In order to define the symbolic heaps, that are the formulae of the separation logic, first, we introduce permission formulae. A key feature of the logic we consider remains in its parameterisation by a permission model.

Permission formulae are defined by the grammar below:

$$\begin{array}{lll} p ::= \mathbf{1} \mid \alpha \mid p \oplus p & & \text{(permission term)} \\ L ::= \top \mid \bot \mid p \leq p \mid \neg(p \leq p) & & \text{(permission literal)} \\ A ::= L \mid A \wedge A & & \text{(permission formula)} \end{array}$$

where $\mathtt{PVar} = \{\alpha, \beta, \dots\}$ is a countably infinite set of **permission variables**.

Notation

- We write $\mathsf{defined}(p)$ for $p \leq \mathbf{1}$ and $p = p'$ for $p \leq p' \wedge p' \leq p$.
- We write $p < p'$ for $p \leq p' \wedge \neg(p' \leq p)$.

A **symbolic heap** with symbolic permissions is a formula (Π, Σ), where Π is a **pure formula** and Σ a **spatial formula** according to the grammar below:

$$\begin{array}{lll} \Pi ::= \top \mid \bot \mid x = y \mid x \neq y \mid A \mid \Pi \wedge \Pi & & \text{(pure formula)} \\ \Sigma ::= \mathtt{emp} \mid \top \mid x \overset{p}{\mapsto} y \mid \Sigma * \Sigma & & \text{(spatial formula)} \end{array}$$

where $\mathtt{LVAR} = \{x, y, \dots\}$ is a countably infinite set of **location/program variables**. We write Π_{pe} and Π_{pv} to denote respectively the permission part of Π and the part about program variables from Π, so that Π is logically equivalent to $\Pi_{pe} \wedge \Pi_{pv}$. We write $\mathtt{LVAR}(\varphi)$ (resp. $\mathtt{PVar}(\varphi)$) to denote the set of location (resp. permission) variables occurring in the syntactic object φ. Moreover, by slightly abusing the notations, $\neg \bot$ (resp. $\neg\top$, $\neg(x \neq y)$, $\neg(x = y)$, $\neg\neg(p \leq p')$) is understood as $\top$ (resp. $\bot$, $x = y$, $x \neq y$, $p \leq p'$), see the rule (Negation) in Figure 2.

2.2 Semantics

Definition 1. *A* **permission model** *is a tuple* $\mathfrak{P} = (P_{\mathfrak{P}}, \mathbf{1}_{\mathfrak{P}}, \oplus_{\mathfrak{P}})$ *such that*

- $P_{\mathfrak{P}} = \{\pi, \dots\}$ *is a set of* **permissions**,
- $\mathbf{1}_{\mathfrak{P}} \in P_{\mathfrak{P}}$ *is a distinguished permission called the* **write** *permission or the* **total** *permission,*
- $\oplus_{\mathfrak{P}} : P_{\mathfrak{P}} \times P_{\mathfrak{P}} \to P_{\mathfrak{P}}$ *is a partial composition that is cancellative, commutative and associative,*[1]
- *the relation* $<_{\mathfrak{P}} \overset{def}{=} \{(\pi', \pi) \mid \pi = \pi' \oplus_{\mathfrak{P}} \pi''$ *for some* $\pi''\}$ *is irreflexive and transitive, with maximum element* $\mathbf{1}_{\mathfrak{P}}$.

An example of a permission model is Boyland's fractional model $\mathfrak{P}_{\mathsf{Boy}} = ((0, 1], 1, \oplus_{\mathfrak{P}_{\mathsf{Boy}}})$ [5], where $\pi \oplus_{\mathfrak{P}_{\mathsf{Boy}}} \pi' \overset{\text{def}}{=} \pi + \pi'$ is defined when the sum is at most 1.

Given $\mathfrak{P} = (P_{\mathfrak{P}}, \mathbf{1}_{\mathfrak{P}}, \oplus_{\mathfrak{P}})$, a **$\mathfrak{P}$-interpretation** is a map $\iota : \mathtt{PVar} \to P_{\mathfrak{P}}$. The map ι is extended to a partial map from the set of permission terms to $P_{\mathfrak{P}}$ so

[1] In particular, whenever a sum $\pi_1 \oplus_{\mathfrak{P}} \pi_2 \dots \oplus_{\mathfrak{P}} \pi_n$ is defined, $\pi_{i_1} \oplus_{\mathfrak{P}} \dots \oplus_{\mathfrak{P}} \pi_{i_k}$ is defined for each non-empty $\{i_1, \dots, i_k\} \subseteq \{1, \dots, n\}$.

that $\iota(\mathbf{1}) \stackrel{\text{def}}{=} \mathbf{1}_{\mathfrak{P}}$ and $\iota(p \oplus p') \stackrel{\text{def}}{=} \iota(p) \oplus_{\mathfrak{P}} \iota(p')$ if $\iota(p)$, $\iota(p')$ and $\iota(p) \oplus_{\mathfrak{P}} \iota(p')$ are defined. Otherwise $\iota(p \oplus p')$ is undefined. We write $\iota \models A$ to denote that ι satisfies the permission formula A, following the clauses below:

- always $\iota \models \top$; never $\iota \models \bot$;
- $\iota \models p \leq p' \stackrel{\text{def}}{\Leftrightarrow}$ both $\iota(p)$ and $\iota(p')$ are defined and, either $\iota(p) = \iota(p')$ or $\iota(p) <_{\mathfrak{P}} \iota(p')$;
- $\iota \models \neg(p \leq p') \stackrel{\text{def}}{\Leftrightarrow} \iota \not\models p \leq p'$;
- $\iota \models A \wedge A' \stackrel{\text{def}}{\Leftrightarrow} \iota \models A$ and $\iota \models A'$.

Since $<_{\mathfrak{P}}$ is irreflexive and transitive, $\iota(p) = \iota(p')$ iff $\iota \models p \leq p'$ and $\iota \models p' \leq p$. A permission formula A is **$\mathfrak{P}$-satisfiable** if there is a $\mathfrak{P}$-interpretation ι such that $\iota \models A$. The entailment $A \models B$ holds if for all $\mathfrak{P}$-interpretations ι, if $\iota \models A$ then $\iota \models B$. The **satisfiability problem w.r.t. $\mathfrak{P}$**, written SAT($\mathfrak{P}$), takes as input A and asks whether it is satisfiable. The **entailment problem w.r.t. $\mathfrak{P}$**, written ENT($\mathfrak{P}$), takes as input permission formulas A and B and asks whether $A \models B$.

Again, let $\mathfrak{P} = (P_{\mathfrak{P}}, \mathbf{1}_{\mathfrak{P}}, \oplus_{\mathfrak{P}})$ be a fixed permission model and let $\mathsf{Loc} = \{\ell, \dots\}$ be a countably infinite set of locations (by default, $\mathsf{Loc} = \mathbb{N}$). A **$\mathfrak{P}$-memory state** is a triple (s, h, ι) where:

- s is a store, i.e. a function $s : \mathtt{LVAR} \to \mathsf{Loc}$ that assigns to each variable a location,
- h is a **$\mathfrak{P}$-heap**, i.e. a partial function with a finite domain $h : \mathsf{Loc} \rightharpoonup_{\mathsf{fin}} P_{\mathfrak{P}} \times \mathsf{Loc}$,
- ι is a $\mathfrak{P}$-interpretation.

Before defining the semantics of symbolic heaps, we define the composition of $\mathfrak{P}$-heaps. The **composition** $h_1 \bullet h_2$ of two $\mathfrak{P}$-heaps h_1 and h_2 is defined whenever there is no $\ell \in \mathrm{dom}(h_1) \cap \mathrm{dom}(h_2)$ with $h_1(\ell) = (\pi_1, \ell_1)$ and $h_2(\ell) = (\pi_2, \ell_2)$ such that either $\ell_1 \neq \ell_2$ or $\pi_1 \oplus_{\mathfrak{P}} \pi_2$ is undefined. When $h_1 \bullet h_2$ is defined, say equal to the $\mathfrak{P}$-heap h, it takes the unique value satisfying the conditions below:

- if $\ell \notin \mathrm{dom}(h_1) \cup \mathrm{dom}(h_2)$, then $\ell \notin \mathrm{dom}(h)$,
- if $\ell \in \mathrm{dom}(h_i) \setminus \mathrm{dom}(h_j)$, then $\ell \in \mathrm{dom}(h)$ and $h(\ell) = h_i(\ell)$ (for all $i \neq j \in \{1, 2\}$),
- if $\ell \in \mathrm{dom}(h_1) \cap \mathrm{dom}(h_2)$, then $\ell \in \mathrm{dom}(h)$ and $h(\ell) = (\pi_1 \oplus_{\mathfrak{P}} \pi_2, \ell')$ with $h_1(\ell) = (\pi_1, \ell')$, $h_2(\ell) = (\pi_2, \ell')$ and $\pi_1 \oplus_{\mathfrak{P}} \pi_2$ is defined.

We write $h' \sqsubseteq h$ if there is h'' so that $h = h' \bullet h''$ and we also write $h' \sqsubset h$ whenever $h' \sqsubseteq h$ and $h' \neq h$. The satisfaction relations $s, h, \iota \models_{\mathfrak{P}} \Sigma$ and $s, h, \iota \models_{\mathfrak{P}} \Pi$ are defined as follows:

$s,h,\iota \models_{\mathfrak{P}} \top$	always	
$s,h,\iota \models_{\mathfrak{P}} \bot$	never	
$s,h,\iota \models_{\mathfrak{P}} x = y$	iff	$s(x) = s(y)$
$s,h,\iota \models_{\mathfrak{P}} x \neq y$	iff	$s(x) \neq s(y)$
$s,h,\iota \models_{\mathfrak{P}} A$	iff	$\iota \models A$
$s,h,\iota \models_{\mathfrak{P}} \Pi_1 \wedge \Pi_2$	iff	$s,h,\iota \models_{\mathfrak{P}} \Pi_1$ and $s,h,\iota \models_{\mathfrak{P}} \Pi_2$
$s,h,\iota \models_{\mathfrak{P}} \texttt{emp}$	iff	$\mathrm{dom}(h) = \emptyset$
$s,h,\iota \models_{\mathfrak{P}} x \overset{p}{\mapsto} y$	iff	$\mathrm{dom}(h) = \{s(x)\}$, $\iota(p)$ is defined, and $h(s(x)) = (\iota(p), s(y))$
$s,h,\iota \models_{\mathfrak{P}} \Sigma_1 * \Sigma_2$	iff	there are subheaps h_1, h_2 such that $h = h_1 \bullet h_2$, $s,h_1,\iota \models_{\mathfrak{P}} \Sigma_1$, and $s,h_2,\iota \models_{\mathfrak{P}} \Sigma_2$.

A **model** of Π, Σ is an interpretation (s,h,ι) such that $s,h,\iota \models_{\mathfrak{P}} \Pi$ and $(s,h,\iota) \models \Sigma$. Most of the time, we omit the subscript $\mathfrak{P}$ and we simply write $(s,h,\iota) \models \Pi$ and $(s,h,\iota) \models \Sigma$.

Our definitions of symbolic heaps and models encompass the standard definitions without permissions: choose $\mathfrak{P}_\mathbf{1} = (\{\mathbf{1}\}, \mathbf{1}, \oplus_{\mathfrak{P}_\mathbf{1}})$ that has only the write permission and the always undefined composition.

2.3 Satisfiability and Entailment Problems

A symbolic heap (Π, Σ) is **$\mathfrak{P}$-satisfiable** if there is a $\mathfrak{P}$-memory state (s,h,ι) such that $s,h,\iota \models_{\mathfrak{P}} \Pi$ and $s,h,\iota \models_{\mathfrak{P}} \Sigma$ and we say that (s,h,ι) is a **$\mathfrak{P}$-model** of (Π, Σ). Two symbolic heaps (Π, Σ) and (Π', Σ') are **equivalent**, written $(\Pi, \Sigma) \equiv_{\mathfrak{P}} (\Pi', \Sigma')$, if they have the same $\mathfrak{P}$-models. The **satisfiability problem w.r.t. $\mathfrak{P}$**, written SATSH($\mathfrak{P}$), takes as input (Π, Σ) and asks whether (Π, Σ) has a $\mathfrak{P}$-model. The **entailment problem w.r.t. $\mathfrak{P}$** written ENTSH($\mathfrak{P}$) takes as input two symbolic heaps (Π, Σ) and (Π', Σ') and asks whether every $\mathfrak{P}$-model of (Π, Σ) is a $\mathfrak{P}$-model of (Π', Σ') (written $(\Pi, \Sigma) \models_{\mathfrak{P}} (\Pi', \Sigma')$). As above, we may omit the subscript $\mathfrak{P}$, when the underlying permission model is clear from the context.

3 A Proof System for Symbolic Heaps with Permissions

In this section, we introduce a proof system for symbolic heaps with permissions. The calculus uses two types of expressions (also called judgments), **entailment judgments** of the form $\Pi, \Sigma \vdash \Pi', \Sigma'$ and **unsatisfiability judgments** of the form $\Pi, \Sigma \vdash \bot$. Below, entailment judgments are also called entailments. We say that $\Pi, \Sigma \vdash \Pi', \Sigma'$ is **true/valid** iff $\Pi, \Sigma \models_{\mathfrak{P}} \Pi', \Sigma'$ (implicitly, we assume a fixed permission model $\mathfrak{P}$). Similarly, we say that $\Pi, \Sigma \vdash \bot$ is **true/valid** iff Π, Σ is unsatisfiable.

We distinguish three types of inference rules:

– rules of the form $\dfrac{}{A}$ with no premisses,

– rules of the form $\dfrac{B}{A}$ with one premiss,

– and rules of the form $\dfrac{B_1 \qquad B_2}{A}$ with two premisses.

The expression A is called the conclusion.

Since we aim at completeness, our proof system will feature some deduction rules that differ from most established proof rules. Therefore, first we discuss what would be these established proof rules and why, aiming at a complete proof system, we need to follow a different approach. In their two complete proof systems for symbolic heaps with lists [1,3], Berdine *et al.* relied on the following "frame rule" (not to be confused with the frame rule for Hoare triples).

$$\frac{\Pi, \Sigma \vdash \Pi', \Sigma'}{\Pi, \Sigma * F \vdash \Pi', \Sigma' * F}$$

This rule allows to "subtract" heap assertions on both sides of an entailment. From the completeness point of view, this rule should be used very carefully, because it is not "invertible" in general: the entailment in the conclusion might be valid even if the premiss is not. Therefore, we prefer to restrict the use of this rule to special cases in which the premiss and the conclusion are equally valid, which will correspond to the rule (Align) in our proof system (side-conditions will apply).

On a different line, all of the reasoning about the $\mapsto$ predicate in the presence of permissions could be summarized in the following equivalence

$$y_1 = y_2, x \overset{p_1 \oplus p_2}{\mapsto} y_1 \equiv_{\mathfrak{P}} \top, x \overset{p_1}{\mapsto} y_1 * x \overset{p_2}{\mapsto} y_2,$$

which can be read as a pair of zero-premiss rules (axioms) to rewrite both the left-hand side and the right-hand side of a given entailment, so as to "merge" all $\mapsto$ predicates with a same left value. This rewriting has an interplay with the equalities in the pure assertions: merging $\mapsto$ predicates in a spatial formula on the left-hand side generates a new equality $y_1 = y_2$, whereas merging $\mapsto$ predicates on the right-hand side of $\vdash$ require to check that this equality is entailed by the left-hand side. We shall proceed differently, and we will only "merge" $\mapsto$ predicates on the left-hand side. On the other hand, we "subtract" every $\mapsto$ predicate found in the right-hand side (see the rule Subtract below). Following this approach, it is a bit simpler to handle a possible occurrence of $\top$ in the right-hand side of $\vdash$.

In the remainder, we first present the normalization rules that are used to merge $\mapsto$ predicates on the left-hand side of a judgment, and later we introduce the proof rules for logical entailments.

3.1 Normalization Rules

In Figure 1, we present a set R of rewrite rules that are used to normalise formulae. The reduction $\Longrightarrow$ is the rewrite relation associated to R and $\Longrightarrow^*$ is

its reflexive and transitive closure. Moreover, $\Pi[y/x], \Sigma[y/x]$ denotes the formula obtained from Π, Σ by replacing each occurrence of x by y.

$$
\begin{array}{lll}
\text{(R-Subst)} & (\Pi, \Sigma) & \Longrightarrow (\Pi, \Sigma[y/x]) \quad \text{if } \Pi \models x = y, \{x, y\} \subseteq \mathtt{LVAR}(\Sigma) \\
\text{(R-Merge)} & (\Pi, \Sigma * x \overset{p}{\mapsto} y * x \overset{p'}{\mapsto} z) & \Longrightarrow (\Pi \wedge y = z, \Sigma * x \overset{p \oplus p'}{\mapsto} y) \\
\text{(R-Empty)} & (\Pi, \Sigma * \mathtt{emp}) & \Longrightarrow (\Pi, \Sigma) \quad \text{if non-empty } \Sigma \\
\text{(R-True)} & (\Pi, \Sigma * \top * \top) & \Longrightarrow (\Pi, \Sigma * \top)
\end{array}
$$

Fig. 1: Rewriting system R

We write $|\Pi, \Sigma|$ to denote the size of the symbolic heap Π, Σ for some reasonably succinct encoding (for instance the number of symbols occurring in Π, Σ).

Lemma 1 ([9]). *The rewrite relation $\Longrightarrow$ has the following properties.*

- *If $(\Pi, \Sigma) \Longrightarrow (\Pi', \Sigma')$ then $(\Pi, \Sigma) \equiv (\Pi', \Sigma')$.*
- *Any rewrite sequence starting from (Π, Σ) terminates in time $\mathcal{O}(|(\Pi, \Sigma)|)$.*

We say that a symbolic heap Π, Σ is **in normal form** iff it is in normal form w.r.t. the $\Longrightarrow$ relation.

Lemma 2 ([9]). *Given (Π, Σ) in normal form, Π, Σ is satisfiable iff $\Pi_{pe} \wedge \mathsf{defined}(\Sigma)$ is satisfiable.*

Therefore each symbolic heap is equivalent to a symbolic heap in normal form and this normal form can be computed in polynomial time.

3.2 Inference Rules

A variable x occurring in $\Pi, \Sigma \vdash \Pi', \Sigma'$ is **solved** iff Π contains an equation $x = y$ and x does not occur anywhere else. In Figure 2 below, we present the inference rules of the proof system.

Remark 1. If the application of a rule results in an empty spatial or pure formula, we write $\mathtt{emp}$ or $\top$ instead, like in:

$$
\frac{x \neq y, \mathtt{emp} \vdash \top, \mathtt{emp}}{x \neq y, x \overset{p}{\mapsto} y \vdash \top, x \overset{p}{\mapsto} y} \textsc{ Align}
$$

Remark 2. To apply one of the rules (Align) or (Subtract), one must check $\Pi_{pe} \wedge \mathsf{defined}(\Sigma) \models p = p'$ and $\Pi_{pe} \wedge \mathsf{defined}(\Sigma) \models p > p'$, respectively. Both conditions can be understood as instances of the entailment problem ENT($\mathfrak{P}$).

(Emp1) $$\frac{}{\Pi, \texttt{emp} \vdash \top, \texttt{emp}}$$

(Emp2) $$\frac{\Pi, \Sigma \vdash \Pi', \Sigma'}{\Pi, \Sigma \vdash \Pi', \Sigma' * \texttt{emp}}$$

(True1) $$\frac{}{\Pi, \Sigma \vdash \top, \top}$$

(True2) $$\frac{\Pi, \Sigma \vdash \Pi', \Sigma' * \top}{\Pi, \Sigma \vdash \Pi', \Sigma' * \top * \top}$$

(Unsat_{pe}) $$\frac{}{\Pi, \Sigma \vdash \Pi', \Sigma'}$$ if $\Pi_{pe} \wedge \mathsf{defined}(\Sigma)$ is $\mathfrak{P}$-unsatisfiable

(Unsat_{pv}) $$\frac{}{\Pi, \Sigma \vdash \Pi', \Sigma'}$$ if Π_{pv} is unsatisfiable (checkable in PTime)

(Simp) $$\frac{\Pi, \Sigma \vdash \Pi', \Sigma'}{x = x \wedge \Pi, \Sigma \vdash \Pi', \Sigma'}$$

(Normalize) $$\frac{\Pi_{NF}, \Sigma_{NF} \vdash \Pi', \Sigma'}{\Pi, \Sigma \vdash \Pi', \Sigma'} \ (\dagger)$$

($\dagger$) if Π, Σ is not in normal form and $\Pi_{NF} \wedge \Sigma_{NF}$ is a normal form of Π, Σ

(Subst) $$\frac{x = y \wedge \Pi[y/x], \Sigma[y/x] \vdash \Pi'[y/x], \Sigma'[y/x]}{\Pi, \Sigma \vdash \Pi', \Sigma'} \ (\ddagger)$$

($\ddagger$) if x is not solved and $\Pi_{pv} \models x = y$

(Substract) $$\frac{p = p' \oplus \alpha \wedge \Pi, \Sigma * x \overset{\alpha}{\mapsto} y \vdash \Pi', \Sigma'}{\Pi, \Sigma * x \overset{p}{\mapsto} y \vdash \Pi', \Sigma' * x' \overset{p'}{\mapsto} y'} \ (\dagger\dagger)$$

($\dagger\dagger$) if $\Pi_{pv} \models x = x' \wedge y = y'$, $\Pi_{pe} \wedge \mathsf{defined}(\Sigma) \models p > p'$, and $\alpha \notin \texttt{PVar}(\Pi, \Sigma, \Pi', \Sigma')$

(Align) $$\frac{\Pi, \Sigma \vdash \Pi', \Sigma'}{\Pi, \Sigma * x \overset{p}{\mapsto} y \vdash \Pi', \Sigma' * x' \overset{p'}{\mapsto} y'} \ (\ddagger\ddagger)$$

($\ddagger\ddagger$) if $\Pi \wedge \mathsf{defined}(\Sigma) \models p = p' \wedge x = x' \wedge y = y'$

(Negation) $$\frac{\neg A \wedge \Pi, \Sigma \vdash \bot \qquad \Pi, \Sigma \vdash \Pi', \Sigma'}{\Pi, \Sigma \vdash A \wedge \Pi', \Sigma'} \ (\dagger\dagger\dagger)$$

($\dagger\dagger\dagger$) if A is a permission literal or an atomic formula

Fig. 2: Inference rules

3.3 Proof Trees

As usual, the construction of a proof tree starts from the given entailment judgment and proceeds upwards until each leaf is an axiom. The construction fails when it ends up with a formula which is not empty but for which no rule applies.

Example 1. Below, we present a proof of the entailment judgment $y = z \wedge \mathbf{1} = \alpha \oplus \alpha, x \overset{1}{\mapsto} y \vdash y = z, x \overset{\alpha}{\mapsto} y * x \overset{\alpha}{\mapsto} y$ for the permission model $((0,1], 1, \oplus_{\mathfrak{P}_{\mathsf{Boy}}})$, in which we use the abbreviation $A' \overset{\text{def}}{=} \mathbf{1} = \alpha \oplus \alpha \wedge \mathbf{1} = \alpha \oplus \alpha'$. Note that the equality $\mathbf{1} = \alpha \oplus \alpha$ enforces that α can only be interpreted by $1/2$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y \neq z \wedge y = z \wedge A', \mathtt{emp} \vdash \bot}\,\textsc{Unsat}_{pv} \quad \dfrac{}{y = z \wedge A', \mathtt{emp} \vdash \top, \mathtt{emp}}\,\textsc{Emp1}}{y = z \wedge A', \mathtt{emp} \vdash y = z, \mathtt{emp}}\,\textsc{Negation}}{y = z \wedge \mathbf{1} = \alpha \oplus \alpha \wedge \mathbf{1} = \alpha \oplus \alpha', x \overset{\alpha'}{\mapsto} y \vdash y = z, x \overset{\alpha}{\mapsto} y}\,\textsc{Align}}{y = z \wedge \mathbf{1} = \alpha \oplus \alpha, x \overset{1}{\mapsto} y \vdash y = z, x \overset{\alpha}{\mapsto} y * x \overset{\alpha}{\mapsto} y}\,\textsc{Subtract}$$

It is worth noting that we do not require that the right-hand side is normalised. Moreover, for the application of the rule (Subtract), we take advantage of $\mathbf{1} = \alpha \oplus \alpha \models_{\mathfrak{P}_{\mathsf{Boy}}} \mathbf{1} > \alpha$ and $\mathbf{1} = \alpha \oplus \alpha \wedge \mathbf{1} = \alpha \oplus \alpha' \models_{\mathfrak{P}_{\mathsf{Boy}}} \alpha = \alpha'$.

In contrast, the formula $x \neq y \wedge \mathbf{1} = \alpha \oplus \alpha, x \overset{\mathbf{1}}{\mapsto} y \vdash \top, x \overset{\alpha}{\mapsto} y$ has no proof tree since the construction fails after one application of the rule (Subtract). Actually, in any model of the left-hand side formula, a write permission is given to x when the right-hand-side only gives a read permission.

$$\dfrac{\dfrac{\dfrac{?}{}\,\textsc{No Rule Applies}}{x \neq y \wedge \mathbf{1} = \alpha \oplus \alpha \wedge \mathbf{1} = \alpha \oplus \alpha', x \overset{\alpha'}{\mapsto} y \vdash \top, \mathtt{emp}}\,\textsc{Subtract}}{x \neq y \wedge \mathbf{1} = \alpha \oplus \alpha, x \overset{\mathbf{1}}{\mapsto} y \vdash \top, x \overset{\alpha}{\mapsto} y}$$

4 Correctness and Completeness of the Proof System

In this section, we establish the correctness and the completeness of our proof system. We proceed first by proving that all axioms are sound (Lemma 3), then we establish that all deduction rules are not only sound, but actually "reversible" (Lemmas 4 and 5). Then, we show that there is no infinite sequence of applications of deduction rules (Lemma 6), and finally that if no rule applies to a given entailment judgment, then this judgment is not valid (Lemma 7).

4.1 Correctness

Lemma 3. *Let $\Pi, \Sigma \vdash \Pi', \Sigma'$ be an entailment judgment such that one rule among (Emp1), (True1), ($\textsc{Unsat}_{pe}$), ($\textsc{Unsat}_{pv}$) applies. Then, the entailment $\Pi, \Sigma \vdash \Pi', \Sigma'$ is valid.*

Proof. (idea) Each model of Π, Σ is a model of Π', Σ' when (Emp1) or (True1) applies, and there is no such model for (Unsat_{pe}), (Unsat_{pv}) otherwise.

Lemma 4. *Let $\Pi, \Sigma \vdash \Pi', \Sigma'$ be an entailment such that one rule among (Emp2), (True2), (Simp), (Normalize), (Subst), (Substract), (Align) applies (backward). Let $\Pi_1, \Sigma_1 \vdash \Pi'_1, \Sigma'_1$ be the premiss of the application. Then the entailment $\Pi, \Sigma \vdash \Pi', \Sigma'$ is valid iff the entailment $\Pi_1, \Sigma_1 \vdash \Pi'_1, \Sigma'_1$ is valid.*

Proof. We prove the property for each application of an inference rule of the form $\dfrac{B}{A}$ (with a unique premiss).

- The cases with (Emp2), (True2) and (Simp) are immediate, as it amounts to eliminate a tautology. For the inference rule (Normalize), the property holds by Lemma 1 and for the inference rule (Subst), a simple reasoning on equality suffices.
- Let us consider the case with the inference rule (Align). Let $\Pi, \Sigma * x \overset{p}{\mapsto} y \vdash \Pi', \Sigma' * x' \overset{p'}{\mapsto} y'$ be the conclusion of the application of (Align).
  - Assume that $\Pi, \Sigma * x \overset{p}{\mapsto} y \vdash \Pi', \Sigma' * x' \overset{p'}{\mapsto} y'$ is true. We shall prove that $\Pi, \Sigma \vdash \Pi', \Sigma'$ is true too when the rule is applicable.
    Let (s, h, ι) be a model for Π, Σ. Consequently, $(s, h, \iota) \models \Pi \wedge \mathsf{defined}(\Sigma)$ by definition of $\models$. By the side-condition of the inference rule (Align), this entails that $(s, h, \iota) \models p = p' \wedge x = x' \wedge y = y'$. Let us introduce the model $(s, h \bullet h', \iota)$ where $\mathrm{dom}(h') = \{s(x)\}$ and $h'(s(x)) = (\iota(s(x)), s(y))$. One can check that $(s, h \bullet h', \iota)$ is a model of $\Pi, \Sigma * x \overset{p}{\mapsto} y$. By assumption $\Pi, \Sigma * x \overset{p}{\mapsto} y \vdash \Pi', \Sigma' * x' \overset{p'}{\mapsto} y'$ is true and therefore $(s, h \bullet h', \iota) \models \Pi', \Sigma' * x' \overset{p'}{\mapsto} y'$ as $s(x) = s(x')$, $s(y) = s(y')$, $\iota(p) = \iota(p')$. So, one can conclude that $(s, h, \iota) \models \Pi', \Sigma'$.
  - Conversely, assume that $\Pi, \Sigma \vdash \Pi', \Sigma'$ is true. For any interpretation (s, h, ι) which is a model of Π, Σ -hence a model of Π', Σ'- , we get a model $(s, h \bullet h', \iota)$ of $\Pi, \Sigma * x \overset{p}{\mapsto} y$ which is also a model of $\Pi', \Sigma' * x \overset{p'}{\mapsto} y'$ since we must have $\iota(p) = \iota(p')$, $s(x) = s(x')$ and $s(y) = s(y')$ as (Align) is applicable.
- Let us consider the case with the inference rule (Substract). Let $\Pi, \Sigma * x \overset{p}{\mapsto} y \models \Pi', \Sigma' * x' \overset{p'}{\mapsto} y'$ be the conclusion of the application of (Substract) and $p = p' \oplus \alpha \wedge \Pi, \Sigma * x \overset{\alpha}{\mapsto} y \vdash \Pi', \Sigma'$ be its premiss. Since the rule is applicable, we assume that $\begin{cases} \Pi_{pv} \models x = x' \wedge y = y' \\ \Pi_{pe} \wedge \mathsf{defined}(\Sigma) \models p > p' \end{cases}$
  - We prove that if $\Pi, \Sigma * x \overset{p}{\mapsto} y \vdash \Pi', \Sigma' * x' \overset{p'}{\mapsto} y'$ is true, then $p = p' \oplus \alpha \wedge \Pi, \Sigma * x \overset{\alpha}{\mapsto} y \vdash \Pi', \Sigma'$ is true.
    Assume that (s, h, ι) is a model of $p = p' \oplus \alpha \wedge \Pi, \Sigma * x \overset{\alpha}{\mapsto} y$.

    From this model we obtain a model (s, h', ι) of $x \overset{p'}{\mapsto} y$ where $h'(s(x)) = (\iota(p'), s(y))$ yielding a model $(s, h \bullet h', \iota)$ of $\Pi, \Sigma * x \overset{\alpha}{\mapsto} y * x \overset{p}{\mapsto} y$. It is a

model of $\Pi, \Sigma * x \overset{p}{\mapsto} y$ hence of $\Pi', \Sigma' * x' \overset{p'}{\mapsto} y'$ by hypothesis. Therefore (s, h, ι) is also a model of Π', Σ'.

- We prove that if $p = p' \oplus \alpha \wedge \Pi, \Sigma * x \overset{\alpha}{\mapsto} y \vdash \Pi', \Sigma'$ is true, then $\Pi, \Sigma * x \overset{p}{\mapsto} y \vdash \Pi', \Sigma' * x' \overset{p'}{\mapsto} y'$ is true.
  Assume that (s, h, ι) is a model of $\Pi, \Sigma * x \overset{p}{\mapsto} y$.
  Since $\Pi_{pv} \models x = x' \wedge y = y'$, we have $s(x) = s(x')$ and $s(y) = s(y')$. We have $(s, h, \iota) \models \Pi \wedge \mathsf{defined}(\Sigma)$ by definition of $\models$. By satisfaction of the side-condition, we conclude that $\iota(p) > \iota(p')$, i.e., $\iota(p) = \iota(p') \oplus \pi$ for some permission π. We can write $h = h_1 \bullet h_2$ where $\mathrm{dom}(h_1) = \mathrm{dom}(h)$, $h_1(s(x)) = (\pi, s(y))$, $h_1(s(z)) = h_1(s(z))$ if $z \neq x$, and $\mathrm{dom}(h_2) = \{s(x)\}$, $h(s(x)) = (\iota(p'), s(y))$. We extend ι to ι' by setting $\iota(\alpha) = \pi$ yielding a model (s, h_1, ι') of $p = p' \oplus \alpha \wedge \Pi, \Sigma * x \overset{\alpha}{\mapsto} y$ which is also a model of Π', Σ' by hypothesis. This proves that $(s, h_1 \bullet h_2, \iota) = (s, h, \iota)$ is a model of $\Pi', \Sigma' * x' \overset{p'}{\mapsto} y'$.

Lemma 5. *Let $\Pi, \Sigma \vdash \Pi', \Sigma'$ be a formula such that (*NEGATION*) applies and let $\Pi_1, \Sigma_1 \vdash \Pi'_1, \Sigma'_1$ and $\Pi_2, \Sigma_2 \vdash \Pi'_2, \Sigma'_2$ be the premisses of the rule. Then the entailment $\Pi, \Sigma \vdash \Pi', \Sigma'$ is true if and only if the entailments $\Pi_1, \Sigma_1 \vdash \Pi'_1, \Sigma'_1$ and $\Pi_2, \Sigma_2 \vdash \Pi'_2, \Sigma'_2$ are true.*

Proof. We prove each implication.

- We prove that if A is a literal and $\Pi, \Sigma \vdash A \wedge \Pi', \Sigma'$ is true then $\neg A \wedge \Pi, \Sigma \models \bot$ and $\Pi, \Sigma \vdash \Pi', \Sigma'$ are true.
  Assume that (s, h, ι) is a model of Π, Σ. Since the entailment is true, it is a model of $A \wedge \Pi', \Sigma'$, i.e. a model of A and a model of Π', Σ'. This proves that $\Pi \wedge \neg A, \Sigma$ is unsatisfiable and that $\Pi, \Sigma \vdash \Pi', \Sigma'$ is true.
- We prove that if A is a literal, $\Pi \wedge \neg A, \Sigma \models \bot$ and $\Pi, \Sigma \vdash \Pi', \Sigma'$ is true then the entailment $\Pi, \Sigma \vdash A \wedge \Pi', \Sigma'$ is true.
  Assume that (s, h, ι) is a model of Π, Σ. Since $\Pi, \Sigma \vdash \Pi', \Sigma'$, it is a model of Π', Σ' and since $\Pi \wedge \neg A, \Sigma \models \bot$, it is a model of Π. Therefore it is a model of $A \wedge \Pi', \Sigma'$.

4.2 Termination

To show the termination of the application of rules, we define the following complexity measure for an entailment $\Pi, \Sigma \vdash \Pi', \Sigma'$. To each entailment judgment we associate the tuple $(rightlit, nbunsolvars, nbpointsto, nbsym, nbvarleft)$ where

- $rightlit$: the number of permission literals or atomic formula of Π',
- $nbunsolvars$: the number of unsolved variables,
- $nbpointsto$: the number of formulae $x \overset{p}{\mapsto} y$ in Σ and Σ',
- $nbsym$: the number of non-logical symbols,
- $nbvarslefts$: the number of location variables of Π.

The lexicographic ordering is used on tuples. Note that the normalization of entailments decreases either the number of $x \overset{p}{\mapsto} y$ formulae or the number of symbols or else the number of location variables. A simple check on rules allows to state the following fact.

Lemma 6. *For each inference rule, the complexity measure of the premisses is smaller than the complexity measure of the conclusion.*

The only rule with two premisses is (NEGATION) but the left premiss must be followed by a successful application of either (UNSAT_{pe}) or (UNSAT_{pv}) otherwise the proof fails.

4.3 Completeness

Lemma 7. *Let $\Pi, \Sigma \vdash \Pi', \Sigma'$ be an entailment such that no rule applies (backward). Then the entailment $\Pi, \Sigma \vdash \Pi', \Sigma'$ is not valid.*

Proof. Assume that no rule applies to $\Pi, \Sigma \vdash \Pi', \Sigma'$ (backward). Since (NEGATION) does not apply, Π' is $\top$. Since (TRUE1) does not apply, Σ' is not $\top$. Since (EMP2) does not apply, Σ' is either $\texttt{emp}$ or does not contain $\texttt{emp}$. Finally, since (NORMALIZE) does not apply, Π, Σ is normalised.

- Assume that Σ' is equal to $\texttt{emp}$. So Σ is distinct from $\texttt{emp}$, otherwise the inference rule (EMP1) applies. Since (UNSAT_{pv}) and (UNSAT_{pe}) are not applicable, $\Pi \wedge \mathsf{defined}(\Sigma)$ is satisfiable. Therefore by Lemma 2, Π, Σ has a model (s, h, ι) with a non-empty heap. But since Σ is distinct from $\texttt{emp}$, $(s, h, \iota) \not\models \texttt{emp}$ and therefore $(s, h, \iota) \not\models \Sigma'$. So, $\Pi, \Sigma \vdash \Pi', \Sigma'$ is not valid.
- Otherwise, Σ' contains a $\mapsto$ predicate or $\top$. If Σ is $\texttt{emp}$, then obviously the entailment is not valid ((UNSAT_{pv}) is not applicable).
- For the remaining case, Σ is of the form $x_1 \overset{p_1}{\mapsto} y_1 * \ldots * x_n \overset{p_n}{\mapsto} y_n(*\top)$ with x_i and x_j distinct for $i \neq j$ and $\Pi \not\models x_i = x_j$, and similarly, Σ' is of the form $x'_1 \overset{p'_1}{\mapsto} y'_1 * \ldots * x'_m \overset{p'_m}{\mapsto} y'_m(*\top)$. It is worth noting that it may happen that x'_i is equal to some x'_j with $i \neq j$. The truth constant $\top$ may be present or not in these $*$-conjuncts.
  - If $\top$ is present in Σ but not in Σ', then $\Pi, \Sigma \vdash \Pi', \Sigma'$ is obviously not valid.
  - In general, for all $\Pi_L, \Sigma_L, \Pi_R, \Sigma_R$ if the entailment $\Pi_L, \Sigma_L \vdash \Pi_R, \Sigma_R * \top$ is not valid, then neither are the entailments $\Pi_L, \Sigma_L * \top \vdash \Pi_R, \Sigma_R * \top$ and $\Pi_L, \Sigma_L \vdash \Pi_R, \Sigma_R$.

  Therefore, without loss of generality, we assume that Σ is of the form $x_1 \overset{p_1}{\mapsto} y_1 * \ldots * x_n \overset{p_n}{\mapsto} y_n$ and Σ' is of the form $x'_1 \overset{p'_1}{\mapsto} y'_1 * \ldots * x'_m \overset{p'_m}{\mapsto} y'_m * \top$. Given a memory state (s, h, ι) satisfying $\Pi, x_1 \overset{p_1}{\mapsto} y_1 * \ldots * x_n \overset{p_n}{\mapsto} y_n$, we can build an alternative memory state (s', h', ι) satisfying the entailment and such that distinct location variables have distinct values. We proceed as follows.
  1. for $j = 1, \ldots, n$, $s'(y_j) \overset{\text{def}}{=} s(y_j)$ if y_j is distinct from x_i for all $i = 1, \ldots, n$,

2. $s'(x_i) \neq s'(x_j)$ if $i \neq j$,
3. $s'(x_i) \neq s'(y_j)$ if y_j is distinct from x_i,
4. $h'(s'(x_i)) = (\iota(p_i), s'(y_i))$.

Again, several cases may occur depending on the form of $x'_1 \overset{p'_1}{\mapsto} y'_1 * \ldots * x'_m \overset{p'_m}{\mapsto} y'_m * \top$.

- There is some x'_j different from any x_i for $i = 1, \ldots, n$. Given a model (s, h, ι) of $\Pi, x_1 \overset{p_1}{\mapsto} y_1 * \ldots * x_n \overset{p_n}{\mapsto} y_n$, the model (s', h', ι) described above in which we impose that $s(x'_j) \neq s(x_i)$ for all i, cannot be a model of $x'_1 \overset{p'_1}{\mapsto} y'_1 * \ldots * x'_m \overset{p'_m}{\mapsto} y'_m * \top$ since $s(x'_j)$ is different from $s(x_i)$ for $i = 1, \ldots, n$.
- There is some x_i different from any x'_j for $i = j, \ldots, m$. Using arguments similar to the ones in the previous case, we can conclude that the entailment is not valid.
- In the remaining case, we have $\{x_i \mid i = 1, \ldots, n\} = \{x'_j \mid i = 1, \ldots, m\}$. So, $m \geq n$. For each $i \in [1, n]$, let $S_i = \{\{p'_j \mid x'_j \text{ equal to } x_i\}\}$.
  Again several cases occur depending on the form of $\Sigma' * \top$.
  - Case 1: $S_i = \{\{p'_j\}\}$ is a singleton for some i. Since (Align) is not applicable, either $\Pi \wedge \mathsf{defined}(\Sigma) \not\models p_i = p'_j$ or $\Pi_{pv} \not\models y_i = y'_j$. In the first case, we can find a model (s, h, ι) of Π, Σ with $\iota(p_i) \neq \iota(p'_j)$. From this model, we get a new model (s', h', ι) as above which cannot be a model of $\top, \Sigma'$. A similar reasoning can be used when $\Pi_{pv} \not\models y_i = y'_j$.
  - Case 2: for all i, $|S_i| \geq 2$. Just pick an arbitrary $i \in [1, n]$. Let $p'_j \in S_i$. Since (Substract) is not applicable, either $\Pi \wedge \mathsf{defined}(\Sigma) \not\models p_i > p'_j$ or $\Pi_{pv} \not\models x_i = x'_j \wedge y_i = y'_j$. In the first case, there is a memory state satisfying the entailment Π, Σ such that $\iota(p_i) \not> \iota(p'_j)$. This memory state does not satisfy $\top, \Sigma'$ and therefore $\Pi, \Sigma \vdash \Pi', \Sigma'$ is not valid. As x_i is equal to x'_j, for the remaning case $\Pi_{pv} \not\models y_i = y'_j$, we can provide a similar analysis.

These lemmas allow to state the following theorem.

Theorem 1 (Correctness and Completeness). *The entailment $\Pi, \Sigma \vdash \Pi', \Sigma'$ is valid iff there is a proof tree for $\Pi, \Sigma \vdash \Pi', \Sigma'$.*

Proof. The previous lemmas state that any application of the rules preserves the entailment, that all proof trees are finite, and that if a proof branch cannot be terminated by an axiom nor a rule application, it contains an invalid entailment.

Conclusion and future work We proposed a proof system for basic symbolic heaps with permissions and we established its soundness and completeness. To achieve this goal, we considered deduction rules that are "invertible" *à la* sequent calculus, and therefore require some particular side conditions and are particular cases of the frame rule or the permission axiom.

As said in the introduction, this is a very preliminary work, and we would be interested to reformulate our proof system with these more standard rules

while keeping the completeness. We are also interested in complete proof systems for some related logics for permission-based separation logics, in particular with abstract predicates *à la* Boolean BI equipped with an external product with a permission algebra (as defined by Le Bach and Hobor [14]), or with concrete, recursively defined predicates, first of which is the singly-linked list predicate.

References

1. J. Berdine, C. Calcagno, and P. O'Hearn. A decidable fragment of separation logic. In *FSTTCS'04*, volume 3328 of *LNCS*, pages 97–109. Springer, 2004.
2. J. Berdine, C. Calcagno, and P. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO'05*, volume 4111 of *LNCS*, pages 115–137. Springer, 2005.
3. J. Berdine, C. Calcagno, and P. W. O'Hearn. Symbolic execution with separation logic. In *APLAS'05*, pages 52–68, 2005.
4. R. Bornat, C. Calcagno, P. O'Hearn, and M. Parkinson. Permission accounting in separation logic. In *POPL'05*, pages 259–270. ACM, 2005.
5. J. Boyland. Checking interference with fractional permissions. In *SAS'03*, number 2694 in LNCS, pages 55–72. Springer, 2003.
6. J. Brotherston. Cyclic proofs for first-order logic with inductive definitions. In *TABLEAUX'05*, volume 3702 of *LNCS*, pages 78–92. Springer, 2005.
7. J. Brotherston, D. Distefano, and R. L. Petersen. Automated cyclic entailment proofs in separation logic. In *CADE'11*, volume 6803 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2011.
8. M. Dahlweid, M. Moskal, Th. Santen, S. Tobies, and W. Schulte. VCC: contract-based modular verification of concurrent C. In *ICSE'09*, pages 429–430. IEEE, 2009.
9. S. Demri, E. Lozes, and D. Lugiez. On symbolic heaps modulo permission theories. In *FSTTCS'17*, volume 93 of *LIPIcs*, pages 25:1–25:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.
10. R. Dockins, A. Hobor, and A.W. Appel. A fresh look at separation algebras and share accounting. In *APLAS'09*, volume 5904 of *LNCS*, pages 161–177. Springer, 2009.
11. G. He, S. Qin, C. Luo, and W.N. Chin. Memory Usage Verification Using Hip/Sleek. In *ATVA'09*, number 5799 in LNCS, pages 166–181. Springer, 2009.
12. B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens. Verifast: A powerful, sound, predictable, fast verifier for C and Java. In *NFM'11*, volume 6617 of *LNCS*, pages 41–55. Springer, 2011.
13. X. Bach Le, C. Gherghina, and A. Hobor. Decision procedures over sophisticated fractional permissions. In *APLAS'12*, pages 368–385, 2012.
14. X. Bach Le and A. Hobor. Logical reasoning for disjoint permissions. In *ESOP'18*, volume 10801 of *LNCS*, pages 385–414. Springer, 2018.
15. K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *LPAR'10*, volume 6355 of *LNCS*, pages 348–370. Springer, 2010.
16. P. Müller, M. Schwerhoff, and A.J. Summers. Viper: A verification infrastructure for permission-based reasoning. In *VMCAI'16*, volume 9583 of *LNCS*, pages 41–62. Springer, 2016.
17. J. Villard, E. Lozes, and C. Calcagno. Tracking heaps that hop with Heap-Hop. In *TACAS'10*, volume 6015 of *LNCS*, pages 275–279. Springer, 2010.