TU Delft

Delft University of Technology

MoSeL

A general, extensible modal framework for interactive proofs in separation logic

Krebbers, Robbert; Jourdan, Jacques-Henri; Jung, Ralf; Tassarotti, Joseph; Kaiser, Jan-Oliver; Timany, Amin; Charguéraud, Arthur; Dreyer, Derek

DOI
10.1145/3236772

Publication date
2018

Document Version
Final published version

Published in
Proceedings of the ACM on Programming Languages

Citation (APA)
Krebbers, R., Jourdan, J.-H., Jung, R., Tassarotti, J., Kaiser, J.-O., Timany, A., Charguéraud, A., & Dreyer, D. (2018). MoSeL: A general, extensible modal framework for interactive proofs in separation logic. *Proceedings of the ACM on Programming Languages*, *2*(ICFP), 77:1-77:30. Article 77. https://doi.org/10.1145/3236772

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

MoSeL: A General, Extensible Modal Framework for Interactive Proofs in Separation Logic

ROBBERT KREBBERS, Delft University of Technology, The Netherlands
JACQUES-HENRI JOURDAN, LRI, Univ. Paris-Sud, CNRS, Université Paris-Saclay, France
RALF JUNG, MPI-SWS, Germany
JOSEPH TASSAROTTI, Carnegie Mellon University, USA
JAN-OLIVER KAISER, MPI-SWS, Germany
AMIN TIMANY, imec-Distrinet, KU Leuven, Belgium
ARTHUR CHARGUÉRAUD, Inria & Université de Strasbourg, CNRS, ICube, France
DEREK DREYER, MPI-SWS, Germany

A number of tools have been developed for carrying out separation-logic proofs mechanically using an interactive proof assistant. One of the most advanced such tools is the Iris Proof Mode (IPM) for Coq, which offers a rich set of tactics for making separation-logic proofs look and feel like ordinary Coq proofs. However, IPM is tied to a particular separation logic (namely, Iris), thus limiting its applicability.

In this paper, we propose MoSeL, a general and extensible Coq framework that brings the benefits of IPM to a much larger class of separation logics. Unlike IPM, MoSeL is applicable to both affine and linear separation logics (and combinations thereof), and provides generic tactics that can be easily extended to account for the bespoke connectives of the logics with which it is instantiated. To demonstrate the effectiveness of MoSeL, we have instantiated it to provide effective tactical support for interactive and semi-automated proofs in six very different separation logics.

CCS Concepts: • **Theory of computation → Logic and verification**; **Separation logic**; **Program verification**;

Additional Key Words and Phrases: Separation logic, logic of bunched implications, modal logic, Coq proof assistant, interactive theorem proving

ACM Reference Format:
Robbert Krebbers, Jacques-Henri Jourdan, Ralf Jung, Joseph Tassarotti, Jan-Oliver Kaiser, Amin Timany, Arthur Charguéraud, and Derek Dreyer. 2018. MoSeL: A General, Extensible Modal Framework for Interactive Proofs in Separation Logic. *Proc. ACM Program. Lang.* 2, ICFP, Article 77 (September 2018), 30 pages. https://doi.org/10.1145/3236772

1 INTRODUCTION

Over the past 20 years, *separation logic* [O'Hearn et al. 2001; Reynolds 2002] has come to play an essential role in the program verification toolbox, with a wide range of variations and applications.

Authors' addresses: Robbert Krebbers, Delft University of Technology, mail@robbertkrebbers.nl; Jacques-Henri Jourdan, LRI, Univ. Paris-Sud, CNRS, Université Paris-Saclay, jacques-henri.jourdan@lri.fr; Ralf Jung, MPI-SWS*, jung@mpi-sws.org; Joseph Tassarotti, Carnegie Mellon University, jtassaro@andrew.cmu.edu; Jan-Oliver Kaiser, MPI-SWS*, janno@mpi-sws.org; Amin Timany, imec-Distrinet, KU Leuven, amin.timany@cs.kuleuven.be; Arthur Charguéraud, Inria, arthur.chargueraud@inria.fr; Derek Dreyer, MPI-SWS*, dreyer@mpi-sws.org.
* Saarland Informatics Campus.

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2018 Copyright held by the owner/author(s).
2475-1421/2018/9-ART77
https://doi.org/10.1145/3236772

Much of the early work on separation logic focused on harnessing its potential to facilitate scalable automatic verification and static analysis of sequential pointer-manipulating programs. To support automation, this work only considered restricted fragments of separation logic [Berdine et al. 2004, 2005]. Then, in their seminal development of Concurrent Separation Logic (CSL), O'Hearn [2007] and Brookes [2007] demonstrated that the core concept of separation logic—namely, that propositions describe *ownership of resources*—was just as useful, if not more so, for reasoning modularly about shared state in concurrent programs. There followed a great deal of work on extending the original CSL with rely-guarantee-style reasoning, in order to verify much deeper specifications for concurrent data structures, such as linearizability [da Rocha Pinto et al. 2014; Sergey et al. 2015b; Turon et al. 2013; Vafeiadis and Parkinson 2007]. More recently, higher-order variants of such concurrent separation logics have been used to build semantic models for programming languages with sophisticated type systems like Rust [Jung et al. 2018a].

In general, the more powerful variants of separation logic are expressive enough that they are difficult to fully automate, and proofs in these logics are complex enough that doing them on pencil-and-paper is no longer trustworthy. Consequently, a number of tools have been developed for carrying out separation-logic proofs mechanically using an interactive proof assistant, *e.g.,* [Bengtson et al. 2012; Cao et al. 2018; Chlipala 2013; Jensen et al. 2013; Krebbers 2015; Sergey et al. 2015a; Tuch et al. 2007]. One of the most advanced such tools is the Iris Proof Mode (IPM) for Coq [Krebbers et al. 2017b]. IPM provides a rich set of tactics for doing proofs in higher-order concurrent separation logic in Coq, tactics which mirror the standard Coq tactics (*e.g.,* `intros`, `split`, `destruct`, etc.) but operate in the substructural context of separation logic rather than the meta-logic of Coq. IPM has already proven indispensable to several mechanized verification efforts, including proving contextual refinement via Kripke logical relations [Krebbers et al. 2017b; Tassarotti et al. 2017; Timany et al. 2018], developing program logics for relaxed memory models [Kaiser et al. 2017] and object capability patterns [Swasey et al. 2017], and establishing soundness of the Rust language in the RustBelt project [Jung et al. 2018a].

As its name suggests, however, IPM is not a proof mode for separation logics in general, but rather for a particular concurrent separation logic called Iris [Jung et al. 2016, 2018b, 2015; Krebbers et al. 2017a]. Above and beyond the Coq engineering challenges of making the IPM implementation parametric in the underlying logic, there are several more fundamental reasons why IPM is not directly usable for other separation logics besides Iris:

(1) Iris is an *affine* separation logic, meaning that it allows hypotheses describing ownership of resources to go unused. Affinity helps to simplify proofs when one does not care to precisely track resource usage (*e.g.,* for garbage-collected languages), and the assumption of affinity is correspondingly baked into many of the IPM tactics. However, many separation logics are *linear* rather than affine—they do not allow owned resources to be dropped on the floor—and it is not obvious how to generalize the tactics of IPM to be sound for such logics as well.
(2) Iris extends traditional separation logic with a handful of modalities—such as the persistence ($\Box$) modality and the step-indexed "later" ($\triangleright$) modality—and IPM provides tactical support for reasoning conveniently about those modalities. However, its tactics are hardwired to the Iris modalities, and do not generalize to other bespoke modalities that appear in other logics (*e.g.,* [Charguéraud and Pottier 2017; Kaiser et al. 2017; Tassarotti et al. 2017]).
(3) Iris was designed as a very general logic in which other advanced separation logics—such as the relaxed-memory logic iGPS [Kaiser et al. 2017]—could be *encoded*. However, in order to make use of IPM in an iGPS proof, one is required to carry out the verification at the level of the Iris encoding of iGPS, thus breaking iGPS's logical abstractions. What we really want is

tactical support for reasoning both at the higher level of iGPS and at the lower level of its Iris encoding, and for mixing the two levels of abstraction when needed.

In this paper, we present MoSeL, a general and extensible Coq framework that brings the benefits of IPM to a much larger class of separation logics. In particular, we make the following contributions:

- We present a general interface for higher-order separation logic with a persistence modality. This interface describes the essence of MoSeL: it is all that its tactics really depend on (§2).
- We show that this interface can be inhabited by a wide variety of logics that feature both affine and linear reasoning. As part of this, we present a new logic based on *ordered resource algebras* that generalizes most of the logics we have seen so far (§3).
- We show that not only is MoSeL backwards compatible with the original IPM (and offshoots thereof)—it also provides effective tactical support for interactive and semi-automated proofs in several logics that fall outside the purview of IPM, namely iGPS [Kaiser et al. 2017], CFML [Charguéraud 2011; Charguéraud and Pottier 2017] and CHL [Chen et al. 2015] (§4).
- We explain how MoSeL extends the approach of the original IPM with more aggressive use of Coq's type classes (including the little-known feature of "higher-order type classes"), in order to make the implementations of its tactics parametric in the particular connectives/modalities they are applied to. In this way, when MoSeL is instantiated with a new logic, it can be easily extended with tactical support for the bespoke connectives of that logic (§5).
- MoSeL has been implemented entirely in Coq, and has been demonstrated to work for six different separation logics—see our Coq repository online [Krebbers et al. 2018].

We conclude the paper with a discussion of related work (§6).

2 A TOUR OF MOSEL

In this section we will give a brief introduction to the modal separation logic proof mode (MoSeL). We start with a description of the abstract structure of a *BI logic*, which is used to make MoSeL parametric in the separation logic that one wishes to use (§2.1). We then continue with a brief recap of some basic tactics of the original Iris Proof Mode (IPM) [Krebbers et al. 2017b] (§2.2), show how these tactics generalize to both affine and linear separation logics (§2.3), and finally show how MoSeL accounts for the important notion of persistent propositions (§2.4).

We only focus in this paper on MoSeL's support for proving separation logic entailments. One can also use MoSeL to build tactics for proving program specifications, but such tactics will be very different depending on the programming language one is reasoning about. As such, MoSeL does not provide a unified set of tactics for reasoning about program specifications, and instead requires one to roll one's own custom tactics for each language. IPM already provided primitives to support programming such tactics, and MoSeL is backwards compatible with these primitives.

2.1 BI Logics

While the original IPM is tied to Iris, MoSeL is parametric in the logic that one wishes to use. This parameterization is achieved by abstracting over an interface consisting of the connectives and laws of the logic. MoSeL uses an extension of a *Bunched Implications (BI) logic* [O'Hearn and Pym 1999; Pym 2002], called a MoBI. Before describing the MoBI interface in its full glory in §2.4, we restrict ourselves to ordinary BI logics in §2.1-§2.3.

A BI logic consists of a pre-ordered set $(\mathit{Prop}, \vdash)$ together with operations:

- $\ulcorner\_\urcorner : \mathbf{Prop} \to \mathit{Prop}$
- $\mathsf{True}, \mathsf{False}, \mathsf{emp} : \mathit{Prop}$
- $\wedge,\ \vee, \Rightarrow,\ *,\ -\!\!* : \mathit{Prop} \times \mathit{Prop} \to \mathit{Prop}$
- $\forall,\ \exists : \forall A.\ (A \to \mathit{Prop}) \to \mathit{Prop}$

$$\begin{array}{c}\text{emp} * P \dashv\vdash P \\ P * Q \vdash Q * P \\ (P * Q) * R \vdash P * (Q * R)\end{array} \qquad \text{*-MONO}\ \frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * P_2 \vdash Q_1 * Q_2} \qquad \text{-*-INTRO}\ \frac{P * Q \vdash R}{P \vdash Q \mathrel{-\!\!*} R} \qquad \text{-*-ELIM}\ \frac{P \vdash Q \mathrel{-\!\!*} R}{P * Q \vdash R}$$

Fig. 1. The BI axioms for spatial connectives.

First, the operation $\ulcorner \_ \urcorner$ is used to embed propositions of the meta logic (Coq's **Prop** in our case) into the BI logic. (Following tradition, we call $\ulcorner \phi \urcorner$ a "pure" proposition.) A BI logic also includes all the standard connectives and laws of higher-order intuitionistic logic (*i.e.,* True, False, $\wedge$, $\vee$, $\Rightarrow$, $\forall$, $\exists$). We omit these laws here, as they are standard.

The distinctive part of the BI logic interface is the part dealing with the spatial connectives ($*$, $-\!\!*$), and their associated laws as shown in Figure 1. (We use $\dashv\vdash$ as notation for bidirectional entailment.) To get an intuitive understanding of these connectives, it is perhaps easier to start with their interpretation in separation logic, where propositions denote ownership of *resources*. Exactly what "resources" are is not important right now, but we do need to assume an associative, commutative *composition* operation on resources r_1 and r_2, written $r_1 \cdot r_2$. This composition is typically (though not always) a *partial* operation, and in particular is only well-defined on r_1 and r_2 iff r_1 and r_2 are "separately ownable".

Separating conjunction $P_1 * P_2$ asserts ownership of the composition of two separately ownable resources satisfying P_1 and P_2, respectively—*i.e.,* it asserts ownership of a combined resource $r = r_1 \cdot r_2$, such that P_1 asserts ownership of r_1 and P_2 asserts ownership of r_2. Magic wand $P -\!\!* Q$ asserts ownership of a resource r_1 such that, for any separately ownable resource r_2 satisfying P, the combined resource $r_1 \cdot r_2$ satisfies Q. This is a form of "separating implication", and indeed -*-INTRO and -*-ELIM say that $*$ and $-\!\!*$ are adjoints in the same way as $\wedge$ and $\Rightarrow$. Finally, emp asserts that we do not own any resources. Together with $*$, it forms a commutative monoid.

Later in this paper, we will instantiate MoSeL with a variety of state-of-the-art separation logics. In this section, however, we will motivate the basics of MoSeL using two of the simplest and oldest variants of separation logic.

Classical separation logic. Classical separation logic [O'Hearn et al. 2001; Reynolds 2002] forms a prototypical instance of a BI logic. Its propositions *Prop* are modeled by predicates over heaps:

$$\sigma \in \mathit{State} \triangleq \mathbb{N} \xrightarrow{\text{fin}} \mathit{Val} \qquad\qquad P, Q \in \mathit{Prop} \triangleq \mathit{State} \to \textbf{Prop}$$

The connectives of classical separation logic are defined in the expected way. Below we show some examples:

$$\begin{array}{ll} \ulcorner \phi \urcorner \triangleq \lambda \sigma.\, \phi & P \wedge Q \triangleq \lambda \sigma.\, P(\sigma) \wedge Q(\sigma) \\ \text{emp} \triangleq \lambda \sigma.\, \sigma = \emptyset & P * Q \triangleq \lambda \sigma.\, \exists \sigma_1, \sigma_2.\, \sigma = \sigma_1 \uplus \sigma_2 \wedge P(\sigma_1) \wedge Q(\sigma_2) \\ \ell \mapsto v \triangleq \lambda \sigma.\, \sigma = [\ell \leftarrow v] & (\exists x : A.\, P) \triangleq \lambda \sigma.\, \exists x : A.\, P(\sigma) \end{array}$$

Note that a BI logic can have more connectives than just those that are present in the BI interface. For example, the $\ell \mapsto v$ connective is not present in the BI interface. Later on in this paper, we will show how one can easily extend MoSeL with support for logic-specific connectives.

Intuitionistic separation logic. Another prototypical instance of a BI logic is *intuitionistic* separation logic [Reynolds 2000]. In this case, the propositions *Prop* are not modeled by just any predicate, but by predicates that are additionally monotone w.r.t. the inclusion order $\subseteq$ on heaps,

and implication on **Prop**:

$$P, Q \in \mathit{Prop} \triangleq \mathit{State} \xrightarrow{\text{mon}} \textbf{Prop}$$

When defining our connectives, we now have an additional proof obligation to establish monotonicity. Most of the connectives can be defined in the same way as in the classical version, but the definition of some connectives needs to be changed to make them monotone:

$$\mathsf{emp} \triangleq \lambda\sigma.\, \mathsf{True} \qquad\qquad \ell \mapsto v \triangleq \lambda\sigma.\, [\ell \leftarrow v] \subseteq \sigma$$

Intuitionistic separation logic is often used to reason about garbage collected languages as it allows one to forget about locations that the program no longer cares about—*i.e.*, $\ell \mapsto v * P \vdash P$ holds for any P. While intuitionistic separation logics can handle languages with manual memory management just fine (see *e.g.*, [Jung et al. 2018a]), in such languages it is sometimes desirable to prove stronger specifications that also establish absence of memory leaks. Intuitionistic separation logics are not suited for this task, but classical separation logics are.

Affine versus general BI logics. As we have just seen, a key difference between classical and intuitionistic separation logic is whether one can forget about locations or not. This notion of being able to forget about resources can be expressed more generally. An *affine BI logic* is a BI logic that enjoys one of the following additional interderivable axioms:

SEP-ELIM	SEP-TRUE	TRUE-EMP
$P * Q \vdash Q$	$P * \mathsf{True} \dashv\vdash P$	$\mathsf{True} \dashv\vdash \mathsf{emp}$

Intuitionistic separation logic is an affine BI logic, but classical separation logic is not.

2.2 Basic Tactics

The original IPM is an example of how interactive proofs in an affine BI logic can look like. With MoSeL, the goal is to support as many of the same tactics as possible in a general BI logic. In this section, we will define the basic context that MoSeL works on, and we will explain some simple IPM tactics that generalize from affine BI logics to general BI logics without restriction.

Let us start with a basic example to show how MoSeL can be used to prove BI entailments:

```
Lemma example_1 {PROP : bi} {A : Type} (P : PROP) (Φ Ψ : A → PROP) :
  (P ∗ (∃ a, Φ a ∨ Ψ a)) -∗ (∃ a, (P ∗ Φ a) ∨ (P ∗ Ψ a)).
```

Notice that `example_1` quantifies over *any* BI logic—we are showing that this law holds in any BI logic. The top-level connective, `-∗`, indicates to the Coq parser that this entire statement should be interpreted using the connectives of some BI logic. (In fact, the outer parentheses are redundant.)

The full proof script is shown in Figure 2. We will step over it carefully. Like an ordinary Coq proof, we start by introducing our hypotheses: `iIntros "[HP H]"`. The *proof mode introduction pattern* `"[HP H]"` eliminates the separating conjunction into its two components.[1] After this first tactic, the context states these two hypotheses separately:

```
"HP" : P
"H" : ∃ a : A, Φ a ∨ Ψ a
--------------------------------------∗
∃ a : A, P ∗ Φ a ∨ P ∗ Ψ a
```

This matches what we would expect from the corresponding Coq tactic `intros [HP H]`. Before we go on, let us discuss how this step can be justified. After all, every MoSeL tactic needs to be proven correct in terms of the basic BI axioms.

[1] Proof mode introduction patterns are similar to vanilla Coq introduction patterns, but have additional features for framing, dealing with modalities, etc. Details can be found in the original IPM paper [Krebbers et al. 2017b].

First of all, we have to translate the context and goal shown by MoSeL into a statement in BI. Formally speaking, if Π is the list of our current hypotheses and Q the current goal, we define the *MoSeL entailment* $\Pi \Vdash Q$ as follows:

$$\Pi \Vdash Q \triangleq \mathop{\Large *}\Pi \vdash Q \qquad \text{(MoSeL-entail-1)}$$

Here, the big $*$ is the *iterated separating conjunction* on a list Π of propositions (the empty list turns into emp). This stands in contrast to ordinary intuitionistic logic (wherein the context represents the ordinary conjunction of the hypotheses) and reflects that, when conducting separation logic proofs, we mostly deal with hypotheses about separately owned resources.

It is important to notice that, just like IPM, MoSeL is *not* a general proof mode for bunched implications. We are using BI as the interface because MoSeL supports both conjunction and separating conjunction, but there are tautologies in BI that cannot be proven in the proof mode because MoSeL does not support arbitrary bunches. For example, one cannot prove $\mathrm{emp} \vdash P \mathrel{-\!\!*} (Q \Rightarrow (P \wedge Q))$ in MoSeL—the proof will get stuck at the intermediate goal $P \Vdash Q \Rightarrow (P \wedge Q)$, where Q cannot be introduced into the context. Our experience with IPM is that arbitrary bunches are not typically needed to handle the kinds of entailments one encounters in separation logic proofs.

The combination of the following rules now state that the tactic `iIntros "[HP H]"` is sound:

$$\textsc{tac-wand-intro}\ \frac{\Pi, P \Vdash Q}{\Pi \Vdash P \mathrel{-\!\!*} Q} \qquad \textsc{tac-sep-elim}\ \frac{\Pi, P_1, P_2 \Vdash Q}{\Pi, P_1 * P_2 \Vdash Q}$$

Using MoSeL-entail-1 and the laws in Figure 1, these rules can be derived without much difficulty.

The next step is to extract the witness from the existential quantifier in `H`, and perform case distinction on the disjunction using `iDestruct "H" as (x) "[H1|H2]"`; this creates two new goals:

```
"HP" : P
"H1" : Φ x
--------------------------------------*
∃ a : A, P * Φ a ∨ P * Ψ a
```

```
"HP" : P
"H2" : Ψ x
--------------------------------------*
∃ a : A, P * Φ a ∨ P * Ψ a
```

Again, this step works in any BI, based on the axioms of the BI logic:

$$\textsc{tac-exist-elim}\ \frac{\forall x.\ (\Pi, P \Vdash Q)}{\Pi, \exists x.\ P \Vdash Q} \qquad \textsc{tac-or-elim}\ \frac{\Pi, P_1 \Vdash Q \qquad \Pi, P_2 \Vdash Q}{\Pi, P_1 \vee P_2 \Vdash Q}$$

Notice that in the premise of tac-exist-elim, the $\forall x$ is a *meta-level* quantification. This reflects the fact that `iDestruct` moved the `x` to the Coq context.

Now that we are done eliminating our hypotheses, we turn our attention to the goal. The tactics `iExists` and `iLeft`/`iRight` are used to introduce existential quantifiers and disjunctions, respectively. These tactics works just like those in plain Coq. Things get interesting when we have to prove a separating conjunction `P * Φ a` or `P * Ψ a`. This is done using the `iSplitL "H1 ... Hn"` tactic, which transforms the goal into two goals: one in which the hypotheses `H1 ... Hn` are available to prove the left-hand side, and another in which the remaining hypotheses are available to prove the right-hand side. Formally speaking, this tactic corresponds to the rule tac-sep-intro, which follows from $*$-mono and associativity of separating conjunction:

$$\textsc{tac-sep-intro}\ \frac{\Pi_1 \Vdash Q_1 \qquad \Pi_2 \Vdash Q_2}{\Pi_1, \Pi_2 \Vdash Q_1 * Q_2} \qquad \textsc{tac-assumption}\ P \Vdash P$$

After splitting the context using `iSplitL`, both goals match the only hypothesis, so the `iAssumption` tactic trivially solves the goal.

```
Lemma example_1 {PROP : bi} {A : Type} (P : PROP) (Φ Ψ : A → PROP) :
  P ∗ (∃ a, Φ a ∨ Ψ a) -∗ ∃ a, (P ∗ Φ a) ∨ (P ∗ Ψ a).
Proof.
  iIntros "[HP H]". iDestruct "H" as (x) "[H1|H2]".
  - iExists x. iLeft. iSplitL "HP"; iAssumption.
  - iExists x. iRight. iSplitL "HP"; iAssumption.
Qed.
Lemma example {PROP : bi} {A : Type} (P : PROP) (Φ Ψ : A → PROP) :
  P ∗ (∃ a, Φ a ∨ Ψ a) -∗ ∃ a, (P ∗ Φ a) ∨ (P ∗ Ψ a).
Proof. iIntros "[HP H]". iFrame "HP". iAssumption. Qed.
```

Fig. 2. A verbose and short proof of `example_1` in MoSeL.

The proof we have described is rather verbose because we eliminated all hypotheses step-by-step, and then introduced them step-by-step. A more efficient approach of performing proofs in separation logic is by *framing*: canceling out a hypothesis in the goal. As shown in Figure 2, a proof based on framing is much shorter. To see how this proof works, let us take a look at the goal we obtained after introducing the hypotheses again:

```
"HP" : P
"H" : ∃ a : A, Φ a ∨ Ψ a
--------------------------------------∗
∃ a : A, P ∗ Φ a ∨ P ∗ Ψ a
```

The `iFrame "HP"` tactic "cancels" the hypothesis `HP : P` in the goal, and thus turns the goal into:

```
"H" : ∃ a : A, Φ a ∨ Ψ a
-------------------------------------∗
∃ a : A, Φ a ∨ Ψ a
```

As before, `iAssumption` trivially solves the goal that we have obtained after framing because it matches the only hypothesis. Formally, the `iFrame` tactic corresponds to the following rule:

$$\textsc{tac-frame}\ \frac{\Pi \Vdash Q}{\Pi, P \Vdash P * Q}$$

The actual `iFrame` tactic needs to do more work as it needs to transform the goal into the appropriate shape. As our simple example already indicates, `P` needs to be distributed over a disjunction and existential quantifier. The original IPM paper [Krebbers et al. 2017b, Section 4.2] explains how this can be implemented using logic programming with type classes in Coq.

2.3 Affine and Absorbing Propositions

So far, we have seen how tactics like `iIntros` or `iExists` in MoSeL work for any general BI in the same way they worked in the original IPM for Iris. However, not all IPM tactics generalize to any BI logic without restrictions: Some of them rely on the fact that Iris is *affine*, *i.e.,* it enjoys $P * Q \vdash Q$ (SEP-ELIM) for *all* P, Q. In this section, we will see how MoSeL adds side-conditions to these tactics such that they still work in specific cases (*i.e.,* for specific choices of P, Q).

To see where things will be different, let us consider a variant of the lemma we proved before:

```
Lemma example_2_wrong {PROP : bi} {A : Type} (P : PROP) (Φ Ψ : A → PROP) :
  P ∗ (∃ a, Φ a ∨ Ψ a) -∗ ∃ a, Φ a ∨ (P ∗ Ψ a).
```

Contrary to the previous version of this lemma, `P` no longer appears in the left disjunct. As such, after performing the necessary eliminations and introductions, we end up with:

```
"HP" : P
"H1" : Φ x
--------------------------------------*
Φ x
```

When presented with this goal, the original Iris Proof Mode allowed one to use the tactic `iClear` to remove the hypothesis `HP`, which would subsequently allow one to solve the goal by assumption. Alternatively, one could just use the tactic `iAssumption` directly, since that tactic implicitly removes the hypotheses that are not needed to solve the goal. In MoSeL, however, these tactics are not sound since we deal with arbitrary BI logics, which, contrary to Iris, do not necessarily enjoy SEP-ELIM.

Fortunately, there are classes of propositions that satisfy SEP-ELIM even in an arbitrary BI: The rule holds if the proposition to be thrown away is *affine* or if the goal is *absorbing*, two notions that we will define in this section:

$$\textsc{sep-elim-general}\quad \frac{\mathsf{affine}(P)\ \mathit{or}\ \mathsf{absorbing}(Q)}{P * Q \vdash Q}$$

Intuitively, a proposition is *affine* if it can be "thrown away". Formally, this class of propositions can be defined as:

$$\mathsf{affine}(P) \triangleq P \vdash \mathsf{emp}$$

Together with the fact that emp is a neutral element for separating conjunction, it is easy to show that for affine propositions P, SEP-ELIM-GENERAL holds. Note that all propositions of intuitionistic separation logic are affine, and a proposition P of classical separation logic is affine iff $P(\sigma)$ implies $\sigma = \emptyset$ for any heap σ.

Dually to being affine, a proposition is *absorbing* if it can "suck up" any additional resources that happen to lie around. Formally, this class of propositions can be defined as:

$$\mathsf{absorbing}(Q) \triangleq Q * \mathsf{True} \vdash Q$$

Again, it is easy to show SEP-ELIM-GENERAL from this and $P \vdash \mathsf{True}$. Note that all propositions of intuitionistic separation logic are absorbing, and a proposition P of classical separation logic is absorbing iff $P(\sigma_1)$ implies $P(\sigma_2)$ for any heaps $\sigma_1 \subseteq \sigma_2$.

Coming back to the MoSeL tactics, from SEP-ELIM-GENERAL, we can easily derive the cases in which the tactics `iClear` or `iAssumption` are applicable:

$$\textsc{tac-clear}\quad \frac{\Pi \Vdash Q \qquad \mathsf{affine}(P)\ \mathit{or}\ \mathsf{absorbing}(Q)}{\Pi, P \Vdash Q} \qquad\qquad \textsc{tac-assumption}\quad \frac{\mathsf{affine}(\Pi)\ \mathit{or}\ \mathsf{absorbing}(Q)}{\Pi, Q \Vdash Q}$$

Modalities. In addition to defining the classes of affine and absorbing propositions, we define two modalities that can be used to *make* any proposition affine or absorbing, respectively:

$$\langle \mathit{affine} \rangle\, P \triangleq \mathsf{emp} \wedge P \qquad\qquad \langle \mathit{absorb} \rangle\, P \triangleq \mathsf{True} * P$$

Both of these modalities are examples of *derived connectives*: We can define them generically for all BI using the primitive connectives, but their interpretation in terms of resources varies from BI to BI (*e.g.*, on the BI's choice for what emp and $\wedge$ mean).

The first modality is useful, for example, when dealing with pure propositions $\ulcorner \phi \urcorner$. In a general BI, pure propositions are not affine, but can be made affine by wrapping them in the $\langle \mathit{affine} \rangle$ modality. Similarly, magic wands $P \mathrel{-\!\!*} Q$ are in general not affine, even if both P and Q are affine.

```
Lemma example_2 {PROP : bi} {A : Type} (P : PROP) (Φ Ψ : A → PROP) :
  <affine> P ∗ (∃ a, Φ a ∨ Ψ a) -∗ ∃ a, Φ a ∨ (P ∗ Ψ a).
Proof.
  iIntros "[HP H]". iDestruct "H" as (x) "[H1|H2]".
  - iExists x. iLeft. iAssumption.
  - iExists x. iRight. iSplitL "HP"; iAssumption.
Qed.
```

Fig. 3. A verbose proof of `example_2` in MoSeL.

So we can use $\langle \mathit{affine} \rangle\, (P \mathrel{-\!\!*} Q)$ to say that we want an affine proof of this magic wand, thus giving us the freedom to drop the magic wand on the floor if we do not need it.

The modalities and the classes are related as follows:

$$\mathrm{affine}(P) \quad \mathit{iff} \quad P \vdash \langle \mathit{affine} \rangle\, P \qquad\qquad \mathrm{absorbing}(P) \quad \mathit{iff} \quad \langle \mathit{absorb} \rangle\, P \vdash P$$

Notice that the other directions hold universally: $\langle \mathit{affine} \rangle\, P \vdash P$ and $P \vdash \langle \mathit{absorb} \rangle\, P$ hold for any P. So, the affinity modality can be universally eliminated, but only introduced freely for affine propositions; and vice versa for the absorbing modality.

Affinity in the proof mode. Using these modalities, we now come back to our second example. The lemma `example_2_wrong`, as stated before, clearly does not hold in a general BI logic. However, using the affinity modality, we can state the following variant which *does* hold:

```
Lemma example_2 {PROP : bi} {A : Type} (P : PROP) (Φ Ψ : A → PROP) :
  <affine> P ∗ (∃ a, Φ a ∨ Ψ a) -∗ ∃ a, Φ a ∨ (P ∗ Ψ a).
```

The proof begins just like the first example. After introducing the existentials and picking the correct side of the disjunction, we are left with the following two goals:

```
"HP" : <affine> P
"H1" : Φ x
--------------------------------------∗
Φ x
```

```
"HP" : <affine> P
"H2" : Ψ x
--------------------------------------∗
P ∗ Ψ x
```

The first goal can be solved by `iAssumption`; MoSeL automatically determines that `HP` is affine and hence can be dropped. The second goal can be completed just as before; `iAssumption` automatically makes use of $\mathrm{affine}(P) \vdash P$. The full (verbose) proof script is given in Figure 3.

2.4 Persistent and Intuitionistic Propositions

Working in a BI logic requires more care than working in an ordinary intuitionistic logic since one has to ensure that each hypothesis is used the right number of times. In this section, we will see how to characterize the class of *intuitionistic* propositions—those that can be used any number of times, including not at all.

MoSeL, like IPM, has special support for such propositions, making them more convenient to use: For example, when introducing a separating conjunction $P_1 * P_2$ using the `iSplitL` tactic, we thus far had to subdivide the hypotheses between the goals for P_1 and P_2, but MoSeL's special support for intuitionistic propositions will allow us to easily use all intuitionistic hypotheses for proving *both* separating conjuncts P_1 and P_2. The key difference from IPM is that MoSeL supports non-affine BIs as well as affine ones. This requires a re-design of the axioms governing intuitionistic propositions, since some of the axioms satisfied by Iris (and relied upon by IPM) are not valid in the presence of non-affine propositions.

$\boxdot$-MONO
$$\frac{P \vdash Q}{\boxdot P \vdash \boxdot Q}$$

$\boxdot$-IDEM
$$\boxdot P \dashv\vdash \boxdot \boxdot P$$

$$\boxdot \forall x.\, P \dashv\vdash \forall x.\, \boxdot P \qquad \boxdot \exists x.\, P \dashv\vdash \exists x.\, \boxdot P$$

$$\boxdot(P \land Q) \dashv\vdash \boxdot P \land \boxdot Q \qquad \boxdot(P \lor Q) \dashv\vdash \boxdot P \lor \boxdot Q$$

$\boxdot$-ABSORB
$$\boxdot P * \mathsf{True} \dashv\vdash \boxdot P$$

$\boxdot$-EMP
$$\mathsf{emp} \vdash \boxdot \mathsf{emp}$$

$\boxdot$-CONJ-ELIM
$$\boxdot P \land Q \vdash P * Q$$

$\boxdot$-ELIM
$$\boxdot P \vdash P * \boxdot P$$

$\boxdot$-EMP-ELIM
$$\boxdot P \land \mathsf{emp} \vdash P$$

Fig. 4. Primitive rules and selected derived rules of the persistence modality.

Iris, and hence IPM, define a class of *persistent* propositions that can be "used without being used up"—they can be consumed multiple times.[2] One example of persistent propositions is the pure proposition $\ulcorner \phi \urcorner$, which embeds a meta-level proposition ϕ in the logic. In an affine BI logic, persistence is a sufficient condition for being intuitionistic.

However, when generalizing to non-affine BI logics, it is possible to have propositions that are persistent, but still have to be used at least once—and indeed, though $\ulcorner \phi \urcorner$ is persistent, it is not affine. Instead, $\ulcorner \phi \urcorner$ is *absorbing* (this is clearly visible in the definition of $\ulcorner \_ \urcorner$ in the simple separation logic in §2.1, where the heap σ is ignored). A proof of $\ulcorner \phi \urcorner$ could "suck up" arbitrary additional resources. Dropping $\ulcorner \phi \urcorner$ would drop these resources, which is not allowed—so $\ulcorner \phi \urcorner$ is not affine, and hence not intuitionistic.

We now proceed to define both of these classes of proposition: intuitionistic and persistent. To account for persistence, we will extend the logical signature required by MoSeL to include a persistence modality, and use that to define the notion of persistent propositions. Then we will use that to define an intuitionistic modality and intuitionistic propositions. Finally, we will see how MoSeL itself treats intuitionistic propositions specially.

MoBI: BI logic with a persistence modality. The fundamental intuition behind persistent propositions is that we can obtain an unbounded number of (separately ownable) copies of them. To turn this intuition into a part of the signature of MoSeL, we first follow the idea of the previous section and define persistent propositions in terms of a new modality, the *persistence modality* $\boxdot$:

$$\mathsf{persistent}(P) \triangleq P \vdash \boxdot P$$

Now, instead of defining $\boxdot$ in the logic (as we did for $\langle \mathit{affine} \rangle$), we instead extend the signature of BI logics to a *MoBI*—a BI logic with an accompanying *persistence modality* $\boxdot$. The reason we take this approach is that it is not clear how to define a persistence modality that works well for all logics. For example, in Iris, persistence is a primitive modality that cannot be defined in terms of the remaining connectives [Bizjak and Birkedal 2018]. Hence, it is incumbent upon the user of MoSeL to supply an appropriate definition of the persistence modality when they instantiate the framework.

A new connective comes with new rules. The rules for the persistence modality are shown in Figure 4. Aside from some of the usual laws that are desirable for a modality (commuting with the quantifiers, idempotency, monotonicity), we have some laws that are expressing the specific properties we require of persistence. The basic intuition for this modality is reflected in $\boxdot$-ELIM: $\boxdot P$ can be used to obtain a fresh (separately ownable) copy of P. Because emp does not require any resources, we can easily obtain copies of emp any time, and thus we demand that emp be persistent

[2]Note that all persistent propositions P are *duplicable*, in the sense that $P \dashv\vdash P * P$, but not the other way around. The proposition $\exists q.\, \ell \overset{q}{\mapsto} 0$ is an example of a duplicable, but non-persistent, proposition.

(⊡-EMP). Moreover, following pure propositions $\ulcorner\phi\urcorner$, persistent propositions $\boxdot P$ are assumed to be absorbing (⊡-ABSORB).

Finally, we need a way to eliminate the persistence modality entirely (rather than just creating copies of its content). Unfortunately, $\boxdot P \vdash P$ (as present in Iris) is not an axiom we can add: Together with ⊡-EMP and ⊡-ABSORB, that would allow us to derive $\mathsf{True} \vdash \mathsf{True} * \mathsf{emp} \vdash \boxdot \mathsf{emp} \vdash \mathsf{emp}$, which only affine BI logics (like Iris) can satisfy. Instead, we slightly weaken the elimination rule to $\boxdot P \land \mathsf{emp} \vdash P$ (⊡-EMP-ELIM). While this may seem somewhat ad-hoc, it actually turns out that both ⊡-EMP-ELIM and ⊡-ELIM can be derived from the more general ⊡-CONJ-ELIM:

$$\boxdot P \land Q \vdash P * Q$$

The intuition for this rule is that from the $\boxdot P$ we can obtain a fresh (separately ownable) copy of P, but by retaining the Q on the right-hand side of the entailment, we ensure that we have not dropped any resources on the floor. To obtain ⊡-EMP-ELIM, we simply pick Q to be emp, and to obtain ⊡-ELIM, we pick Q to be $\boxdot P$.

Some of the other rules can also be derived. In fact, the only primitive laws are ⊡-MONO, ⊡-CONJ-ELIM, ⊡-EMP, ⊡-IDEM (left-to-right), ⊡-ABSORB (left-to-right), commuting with universal quantification (right-to-left) and commuting with existential quantification (left-to-right). All remaining laws, and the other directions of the mentioned laws, can be derived from these.

In a simple separation logic like either the classical or intuitionistic separation logics we saw in §2.1, the persistence modality can be defined as follows:

$$\boxdot P \triangleq \lambda\sigma.\, P(\emptyset) \qquad \text{(PERSISTENCE-SIMPLE)}$$

That is, $\boxdot P$ holds if P holds for the empty heap. This definition satisfies all the laws in Figure 4.

Intuitionistic propositions in a linear world. Now that we have defined persistence, we turn our attention to the notion of intuitionistic propositions. Intuitively, a proposition is intuitionistic if it is both persistent (we can create copies) and affine (we can drop it). Wrapping persistent propositions in the affinely modality nicely handles the problem of not being able to drop them. For example, while we have seen that $\ulcorner\phi\urcorner$ is not intuitionistic in all BIs, $\langle \mathit{affine} \rangle \ulcorner\phi\urcorner$ is in fact both affine and persistent and hence intuitionistic. The additional $\langle \mathit{affine} \rangle$ ensures that this proposition does not "suck up" arbitrary resources the way $\ulcorner\phi\urcorner$ does.

Indeed, this is exactly how we define the corresponding *intuitionistic modality* □:

$$\square P \triangleq \langle \mathit{affine} \rangle \boxdot P \qquad\qquad \mathsf{intuitionistic}(P) \triangleq P \vdash \square P$$

It can be easily verified that P is intuitionistic if and only if it is persistent and affine.

Like IPM, MoSeL provides support for intuitionistic propositions by splitting the context into two parts: the *spatial context* that we have already seen above, and the *intuitionistic context* which will only contain intuitionistic propositions. We extend the *MoSeL entailment* with this extra context:

$$\Gamma; \Pi \Vdash Q \quad \triangleq \quad \square\left(\bigwedge \Gamma\right) * \mathbin{\text{✻}} \Pi \vdash Q \qquad \text{(MOSEL-ENTAIL)}$$

Notice how the intuitionistic modality is used to represent the fact that the propositions in Γ can be used as often as needed (or not at all). IPM employed a similar definition of the entailment, but using just the persistence modality ⊡, since in Iris all propositions are affine.[3]

Figure 5 shows some of the rules that can be derived for the intuitionistic modality. In particular, it is affine (□-AFFINE), can be freely eliminated (□-ELIM), and is duplicable (□-DUP), matching the intuition that intuitionistic propositions can be used not at all, once, or multiple times. Furthermore,

[3]In fact, Iris used □ as notation for the persistence modality; but since both modalities coincide for affine BIs, that is consistent with our usage of the symbols.

$$\frac{P \vdash Q}{\Box P \vdash \Box Q}\ \Box\text{-MONO} \qquad \Box P \dashv\vdash \Box\Box P\ \Box\text{-IDEM}$$

$$\Box(P \wedge Q) \dashv\vdash \Box P \wedge \Box Q \qquad \Box(P \vee Q) \dashv\vdash \Box P \vee \Box Q$$

$$\Box \forall x.\, P \vdash \forall x.\, \Box P \qquad \Box \exists x.\, P \dashv\vdash \exists x.\, \Box P$$

$$\Box P \vdash \mathrm{emp}\ \Box\text{-AFFINE} \qquad \Box P \vdash P\ \Box\text{-ELIM} \qquad \Box P \dashv\vdash \Box P * \Box P\ \Box\text{-DUP} \qquad \Box P * \Box P \dashv\vdash \Box P \wedge \Box P\ \Box\text{-SEP-AND}$$

Fig. 5. Selected derived rules of the intuitionistic modality.

```
Lemma example_3 {PROP : bi} {A : Type} (P : PROP) (Φ Ψ : A → PROP) :
  □ P ∗ (∃ a, Φ a ∨ Ψ a) -∗ ∃ a, Φ a ∨ (P ∗ P ∗ Ψ a).
Proof.
  iIntros "[#HP H]". iDestruct "H" as (x) "[H1|H2]".
  - iExists x. iLeft. iAssumption.
  - iExists x. iRight. iSplitR; [|iSplitR]; iAssumption.
Qed.
```

Fig. 6. A verbose proof of `example_3` in MoSeL.

it commutes with conjunction, disjunction, and existential quantification, which permits standard manipulation of hypotheses in the intuitionistic context like eliminating an existential quantifier. It only commutes one-way with universal quantification, but that is sufficient to eliminate universal quantification (via "specialization") in the intuitionistic context.[4]

Intuitionistic propositions in MoSeL. To see how intuitionistic propositions work in MoSeL in practice, let us consider the following variant of our running example:

```
Lemma example_3 {PROP : bi} {A : Type} (P : PROP) (Φ Ψ : A → PROP) :
  □ P ∗ (∃ a, Φ a ∨ Ψ a) -∗ ∃ a, Φ a ∨ (P ∗ P ∗ Ψ a).
```

This time, we use P not at all in the left disjunct in the conclusion, and use it *twice* in the right disjunct. The introduction part of the proof script changes slightly: We start with `iIntros "[#HP H]"`. The # sign tells MoSeL to move the proposition to the intuitionistic context. This corresponds to the following rule, using MoSeL-ENTAIL extended with an intuitionistic context:[5]

$$\frac{\Gamma, P; \Pi \Vdash Q \qquad \mathrm{intuitionistic}(P)}{\Gamma; \Pi, P \Vdash Q}$$

This rule follows from □-SEP-AND and □ commuting with conjunction.

In Coq, the goal state now looks as follows:

```
"HP" : P
--------------------------------------□
"H" : ∃ a : A, Φ a ∨ Ψ a
--------------------------------------∗
∃ a : A, Φ a ∨ P ∗ P ∗ Ψ a
```

Notice how, as in IPM, the two contexts (intuitionistic and spatial) are rendered separately.

[4]The trouble with commuting □ "out of" a ∀ is that, in case the domain of quantification is empty, we would have to prove True ⊢ □ True which does not hold. For non-empty domains, the two connectives do in fact commute.

[5]IPM called this the "persistent context", again because it was restricted to the affine setting and did not have to distinguish persistent and intuitionistic propositions.

```
Lemma example_4 {PROP : bi} {A : Type} (P Q : PROP) : □ P ∧ □ Q -∗ □ (P ∗ Q).
Proof. iIntros "[#HP #HQ]". iModIntro. iSplitL; iAssumption. Qed.
```

Fig. 7. A verbose proof of `example_4` in MoSeL.

The proof proceeds as previously, and for the first case, completes in the same way, except that now the side-condition of `iAssumption` only requires all remaining hypotheses in the *spatial* context (of which there are none) to be affine, since the intuitionistic hypotheses are affine by definition. For the second case, we end up facing the following goal:

```
"HP" : P
--------------------------------------□
"H2" : Ψ x
--------------------------------------∗
P ∗ P ∗ Ψ x
```

We now run `iSplitR`, taking all our assumptions to the right-hand side of the separating conjunction. However, this only refers to the spatial assumptions—everything in the intuitionistic context gets moved to *both* goals. This matches the following generalization of the introduction rule for separating conjunction, where Γ can be used in both subgoals:

$$\textsc{tac-sep-intro}\quad \frac{\Gamma; \Pi_1 \Vdash Q_1 \qquad \Gamma; \Pi_2 \Vdash Q_2}{\Gamma; \Pi_1, \Pi_2 \Vdash Q_1 * Q_2}$$

Soundness of this rule easily follows from $\Box$-DUP.

The proof then completes in the same way as before. The full proof script is given in Figure 6.

2.5 Introducing Modalities

We have seen how MoSeL can exploit the fact that a proposition is intuitionistic, and we have shown the modality $\Box$ to express that we can use a proposition intuitionistically. This begs the question: How can we *prove* $\Box P$ in MoSeL? And what about other modalities, like $\langle \textit{affine} \rangle P$? IPM had built-in support for introducing $\Box$; MoSeL generalizes this in the form of the `iModIntro` tactic, which can handle a wide variety of modalities with similar properties.

The key proof rule for introducing $\Box$ is the following, derived from $\Box$-MONO and $\Box$-IDEM:

$$\Box\textsc{-intro}\quad \frac{\Box P \vdash Q}{\Box P \vdash \Box Q}$$

Read in a bottom-up way, this rule says that if our context is wrapped in $\Box$, we can remove a leading $\Box$ from our goal.

Let us now see this rule in action when proving the following lemma:

```
Lemma example_4 {PROP : bi} {A : Type} (P Q : PROP) : □ P ∧ □ Q -∗ □ (P ∗ Q).
```

After introducing the assumptions with `iIntros "[#HP #HQ]"`, we have the following goal:

```
"HP" : P
"HQ" : Q
--------------------------------------□
□ (P ∗ Q)
```

Now we can use the `iModIntro` tactic to introduce the $\Box$ modality. This tactic will make sure that we can only proceed with an empty spatial context: If there are any assumptions in the spatial context, it will try to clear them (if they are affine), and fail if that is impossible. Then it uses $\Box$-intro to change the goal to $P * Q$. The overall effect is described by the following rule:

$$\frac{\Gamma \vdash Q \qquad \mathsf{affine}(\Pi)}{\Gamma; \Pi \vdash \Box Q}\ \textsc{tac-intuitionistic-intro}$$

The proof is then trivial to complete; the complete proof script is shown in Figure 7.

The tactic `iModIntro` works not only for the $\Box$ modality. For example, it also supports introducing the $\boxdot$ and $\langle \mathit{affine} \rangle$ modalities based on the following derivable rules:

$$\frac{\langle \mathit{affine} \rangle\ P \vdash Q}{\langle \mathit{affine} \rangle\ P \vdash \langle \mathit{affine} \rangle\ Q}\ \langle \mathit{affine} \rangle\textsc{-intro} \qquad \frac{\boxdot P \vdash Q}{\boxdot P \vdash \boxdot Q}\ \boxdot\textsc{-intro}$$

In fact, `iModIntro` is based on an extensible infrastructure that can be taught about logic-specific modalities, so long as they satisfy similar rules. We will describe this infrastructure in §5.3.

3 INSTANTIATING MOSEL'S MOBI INTERFACE FOR VARIANTS OF IRIS

In the previous section, we have shown that MoSeL's interface can be inhabited by two of the simplest and oldest variants of separation logic. Since MoSeL is derived from IPM, the obvious first candidate for a significant logic instantiating the interface is Iris itself. In this section, we show that not only does Iris satisfy the axioms of a MoBI logic required by MoSeL (§3.1), but a variant of Iris by Tassarotti et al. [2017] that has been modified to support reasoning about linear resources is also a MoBI logic (§3.2). Finally, we show that both the original Iris and this linear variant are subsumed by an even more general model (§3.3). In particular, the logics in this section make use of more interesting persistence modalities, showing that the "trivial" instance (persistence-simple, as used for classical separation logic in §2.4) is not the only choice, even for non-affine logics.

3.1 Iris

With MoSeL being a strict generalization of the original IPM that works for Iris, the first instance we will look at is Iris itself. Iris already comes with a persistence modality, which we can use directly for MoSeL.[6] It is easy to verify that Iris is a MoBI logic.

Recovering the full IPM. However, being a MoBI logic is far from enough. First of all, Iris is an affine BI logic; cases like `example_2` in §2.3 should not require adding this $\langle \mathit{affine} \rangle$ modality. MoSeL supports this seamlessly because the side-conditions incurred, *e.g.*, by `iClear` are trivially satisfied for any proposition. This restores backwards compatibility with IPM w.r.t. exploiting affinity.

Moreover, apart from the ordinary separation logic connectives and the persistence modality that are part of the MoBI interface, Iris has many additional connectives with their own reasoning principles that we wish MoSeL to support. For example, Iris has various modalities for step-indexing, and *update modalities* to reason about changes to the resources owned by a thread. It turns out that the original IPM already laid the groundwork necessary for supporting additional connectives: IPM was extensible in the sense that users could define derived connectives in Iris, and "register" them with IPM to get proper support in interactive proofs. In §5 we will describe various extensions

[6] Iris uses the notation $\Box$ for its persistence modality, while we use $\boxdot$. However, Iris is an affine BI logic, so both modalities coincide—thus, our interpretation of $\Box$ as the intuitionistic modality is compatible with Iris's use of it.

A *unital resource algebra* (uRA) is a tuple
$(M : \mathbf{Type}, \mathcal{V} : M \to \mathbf{Prop}, |-| : M \to M, (\cdot) : M \times M \to M, \varepsilon : M)$ satisfying:

$$\forall a, b, c.\ (a \cdot b) \cdot c = a \cdot (b \cdot c) \quad \text{(RA-ASSOC)} \qquad \forall a, b.\ \mathcal{V}(a \cdot b) \Rrightarrow \mathcal{V}(a) \quad \text{(RA-VALID-OP)}$$
$$\forall a, b.\ a \cdot b = b \cdot a \quad \text{(RA-COMM)} \qquad \forall a.\ \varepsilon \cdot a = a \quad \text{(RA-UNIT-OP)}$$
$$\forall a.\ |a| \cdot a = a \quad \text{(RA-CORE-ID)} \qquad \mathcal{V}(\varepsilon) \quad \text{(RA-UNIT-VALID)}$$
$$\forall a.\ ||a|| = |a| \quad \text{(RA-CORE-IDEM)}$$
$$\forall a, b.\ a \preccurlyeq b \Rrightarrow |a| \preccurlyeq |b| \quad \text{(RA-CORE-MONO)} \qquad \text{where} \quad a \preccurlyeq b \triangleq \exists c \in M.\ b = a \cdot c$$

Fig. 8. Resource algebras.

of that infrastructure to completely decouple MoSeL from Iris while providing full backwards compatibility with IPM.

Before we go on to discuss new logics that IPM could not handle, we briefly review a simplified version of the model of Iris. We will entirely ignore step-indexing. The purpose of this model is to be able to compare the logics we will discuss later (descendants of Iris) with Iris itself.

A simplified model of Iris. We have shown a model of a simple intuitionistic separation logic in §2.1. That model was based on predicates over heaps, but it is straightforward to generalize this to predicates over an arbitrary partial commutative monoid (PCM). Iris takes one more step and uses a *unital resource algebra* [Jung et al. 2018b] (uRA) as its model of resources. The axioms for uRAs are shown in Figure 8. uRAs differ from PCMs in the following two respects:

(1) Composition $(\cdot)$ is a total operation. To encode the fact that some resources are incompatible and hence cannot coexist, uRAs come with a *validity predicate* $\mathcal{V}$ that carves out a subset of elements that are considered "valid". Two elements a and b are *compatible* iff $\mathcal{V}(a \cdot b)$. For example, two overlapping heaps would compose to some "error" element.
The reason for this is twofold: First of all, pragmatically speaking, total functions are easier to handle in Coq than partial functions. Secondly, this approach of encoding compatibility via a validity predicate scales better to the step-indexed setting [Jung et al. 2018b].
(2) To model the persistence modality, RAs also have a function $|-|$ that assigns to each element a its *(duplicable) core* $|a|$, as demanded by RA-CORE-ID. We further demand that $|-|$ is idempotent (RA-CORE-IDEM) and monotone (RA-CORE-MONO) with respect to the *extension order* $\preccurlyeq$.

Based on this, we define the set of *Iris propositions* as a quotient of the monotone predicates (w.r.t. $\preccurlyeq$) over some uRA M:

$$\mathit{Prop} \triangleq M \xrightarrow{\text{mon}_{\preccurlyeq}} \mathbf{Prop} \big/_{\equiv}$$
$$P \vdash Q \triangleq \forall a.\ (\mathcal{V}(a) \wedge P(a)) \Rrightarrow Q(a) \qquad P \equiv Q \triangleq (P \vdash Q) \wedge (Q \vdash P)$$

The quotient equates propositions if they agree on all *valid* elements of M.

We can now define all the connectives required by MoSeL, some of which we show below. Moreover, as a generalization of $\ell \mapsto v$, we define a notion of "ownership of a uRA element".

$$P * Q \triangleq \lambda a.\ \exists a_1, a_2.\ a = a_1 \cdot a_2 \wedge P(a_1) \wedge Q(a_2) \qquad \mathsf{emp} \triangleq \lambda a.\ \mathsf{True}$$
$$\boxdot P \triangleq \lambda a.\ P(|a|) \qquad \mathsf{Own}\,(b) \triangleq \lambda a.\ b \preccurlyeq a$$

For each of these definitions, we have to prove that they are indeed monotone as required by the definition of *Prop*. For $\boxdot$, this relies on RA-CORE-MONO. Note that persistence is generalized to be able to retain some duplicable part of the resources a, showing that MoSeL does not require $\boxdot$ to

strip away all resources (like it did in intuitionistic separation logic, as shown in §2.1). The axiom ⊡-IDEM is justified by RA-CORE-IDEM, and the axiom ⊡-CONJ-ELIM is justified by RA-CORE-ID.

Iris provides a very general model that subsumes not only the simple intuitionistic separation logic from §2.1 but also extensions of it with fractions, permissions, tokens and other kinds of ghost state. The simple logic can be recovered by picking M to be finite maps with disjoint union, $|a| \triangleq \varepsilon$, and defining $\ell \mapsto v \triangleq \mathsf{Own}([\ell \leftarrow v])$ (using the singleton map where ℓ is assigned v).

3.2 Iris with Linear Ownership

We are going to look at a variant of Iris developed by Tassarotti et al. [2017] that features *linear ownership—i.e.,* this logic is *not* an affine BI logic. Linear ownership in this logic is used to ensure that program refinements preserve termination behavior under *fair* scheduling [Lehmann et al. 1981] of concurrent threads. Tassarotti et al. did not give their linear version of Iris a name, but we will call it Fairis to avoid confusion with the many other logics discussed in this paper.

Tassarotti et al. adapted the original IPM to support Fairis. However, unlike MoSeL, this version of IPM was specific to Fairis and was not backwards compatible with Iris. (We discuss some further limitations of their proof mode in §6.) We now describe the model of Fairis and show how MoSeL can be instantiated with Fairis in order to replace the latter's bespoke proof mode.

In Fairis, propositions are predicates over *pairs* of uRA elements. (Just as in our presentation of the Iris model, we simplify the presentation by ignoring step-indexing.) The first component represents *affine* ownership of a resource that can be dropped, while the second component represents *linear* ownership that must be tracked precisely. In the model, this is reflected by requiring the predicates to be monotone with respect to the first component, but not the second:

$$\mathit{Prop} \triangleq \{P \in M \times M \to \mathbf{Prop} \mid \forall a_1, a_2, b.\ a_1 \preccurlyeq a_2 \Rightarrow (P(a_1, b) \Rightarrow P(a_2, b))\}_{/\equiv}$$

$$P \vdash Q \triangleq \forall a, b.\ (\mathcal{V}(a) \wedge \mathcal{V}(b) \wedge P(a, b)) \Rightarrow Q(a, b) \qquad P \equiv Q \triangleq (P \vdash Q) \wedge (Q \vdash P)$$

The quotient only considers the value of $P(a, b)$ relevant if both the affine and linear resources are valid.

Most connectives operate component-wise, for example:

$$P * Q \triangleq \lambda(a, b).\ \exists a_1, a_2, b_1, b_2.\ a = a_1 \cdot a_2 \wedge b = b_1 \cdot b_2 \wedge P(a_1, b_1) \wedge Q(a_2, b_2)$$

Below we display the definitions of the more interesting connectives:

$$\mathsf{emp} \triangleq \lambda(a, b).\ b = \varepsilon \qquad \mathsf{OwnAffine}(a') \triangleq \lambda(a, b).\ a' \preccurlyeq a \wedge b = \varepsilon$$

$$\boxdot P \triangleq \lambda(a, b).\ P(|a|, \varepsilon) \qquad \mathsf{OwnLinear}(b') \triangleq \lambda(a, b).\ b = b'$$

Since Fairis features both affine and linear resources, it has a connective $\mathsf{OwnAffine}(a')$ for ownership of an affine resource a', and a connective $\mathsf{OwnLinear}(b')$ for ownership of a linear resource b'. The former requires the linear resource to be ε while the latter requires the linear resource to be exactly b'. Since the emp connective is defined to hold when the linear resource is empty, this gives that the $\mathsf{OwnAffine}(a')$ connective is indeed affine.

The persistence modality behaves like Iris's persistence modality on the affine resource, while it acts like the one of classical separation logic (see §2.4) on the linear resource. It is tempting to define persistence as taking the core of both the affine and the linear resource, but that is incompatible with ⊡-EMP, from which we can derive that $(\boxdot \mathsf{emp})(a, b)$ has to hold for all resources.

Originally, Fairis did not have a persistence modality as defined above, and instead defined the notion of intuitionistic propositions differently. This notion was obtained by combining affinity with another modality, the *relevance modality* $\langle \mathit{relv} \rangle$. The details of the $\langle \mathit{relv} \rangle$ modality do not matter so much, but what is important is that the old and new definition of intuitionistic propositions

An *ordered unital resource algebra* (uORA) is a tuple
$(M : \mathbf{Type}, \sqsubseteq : M \times M \to \mathbf{Prop}, \mathcal{V} : M \to \mathbf{Prop}, |-| : M \to M, (\cdot) : M \times M \to M, \varepsilon : M)$ satisfying:

$$\forall a, b, c.\ (a \cdot b) \cdot c = a \cdot (b \cdot c) \quad \text{(ORA-ASSOC)}$$
$$\forall a, b.\ a \cdot b = b \cdot a \quad \text{(ORA-COMM)}$$
$$\forall a.\ |a| \cdot a = a \quad \text{(ORA-CORE-ID)}$$
$$\forall a.\ ||a|| = |a| \quad \text{(ORA-CORE-IDEM)}$$
$$\forall a, b.\ \mathcal{V}(a \cdot b) \Rightarrow \mathcal{V}(a) \quad \text{(ORA-VALID-OP)}$$
$$\forall a, b.\ a \sqsubseteq b \Rightarrow (\mathcal{V}(b) \Rightarrow \mathcal{V}(a)) \quad \text{(ORA-VALID-MONO)}$$
$$\mathcal{V}(\varepsilon) \quad \text{(ORA-UNIT-VALID)}$$
$$\forall a.\ a \sqsubseteq a \quad \text{(ORA-}\sqsubseteq\text{-REFL)}$$
$$\forall a, b, c.\ a \sqsubseteq b \land b \sqsubseteq c \Rightarrow a \sqsubseteq c \quad \text{(ORA-}\sqsubseteq\text{-TRANS)}$$
$$\forall a, b, c.\ a \sqsubseteq b \Rightarrow a \cdot c \sqsubseteq b \cdot c \quad \text{(ORA-}\sqsubseteq\text{-OP)}$$
$$\forall a, b.\ a \sqsubseteq b \Rightarrow |a| \sqsubseteq |b| \quad \text{(ORA-CORE-MONO)}$$
$$\forall a, b.\ |a| \sqsubseteq |a \cdot b| \quad \text{(ORA-}\sqsubseteq\text{-OP-CORE)}$$
$$\forall a.\ \varepsilon \sqsubseteq |a| \quad \text{(ORA-}\sqsubseteq\text{-UNIT-CORE)}$$
$$\forall a.\ \varepsilon \cdot a = a \quad \text{(ORA-UNIT-OP)}$$

Fig. 9. Ordered resource algebras.

(using $\langle rel v \rangle$ and $\boxdot$, respectively) coincide. Since, when doing proofs, we are primarily interested in the intuitionistic modality, and not so much in the persistence or relevance modality, this means that most proofs were not affected fundamentally when changing from Fairis to MoSeL.

We updated all of the proofs of examples from Fairis to use MoSeL. Many of the proofs went through with only minor notational changes. In other cases, we were able to shorten and improve the readability of proofs by using MoSeL's better support for introduction and elimination patterns.

3.3 General Linear and Affine Resources

While Fairis shows that the axioms of MoSeL (in particular, those for persistence) are compatible with general (non-affine) BI logics, the treatment of linear resources in Fairis is somewhat ad-hoc. In particular, the resources of this logic are always a product of linear and affine pieces.

In this section, we validate that MoSeL can handle an even more general class of logics that live anywhere on the gradient between an affine and a fully linear BI logic (where no ownership is affine). Concretely, we generalize the notion of uRAs (unital resource algebras) such that all the logics we have discussed so far are instances of this more general model.

The basic idea behind this generalized model is to equip uRAs with a *pre-order* that explains which resources can be "thrown away": If $a \sqsubseteq b$, then one can "throw away" parts of b and only keep a around. In particular, if $\varepsilon \sqsubseteq b$, then b can be thrown away entirely. The idea of adding a pre-order to a "PCM-like structure" has been used before to give general models of separation logic [Cao et al. 2017], so it was a natural choice to do the same with Iris's resource algebras. We call these structures *unital ordered resource algebras*, uORAs. The axioms are given in Figure 9. The axioms on the left-hand side are the same as in Figure 8 for uRAs; the axioms on the right-hand side govern the newly added pre-order. Most importantly, $\sqsubseteq$ is a pre-order that is compatible with composition (ORA-$\sqsubseteq$-OP). The remaining axioms will be motivated by the MoBI proof rules. Again, we ignore step-indexing to keep the discussion focused on the handling of resources.

When modeling the logic, the pre-order $\sqsubseteq$ controls the monotonicity requirement: If a proposition P holds for some resource a and we have $a \sqsubseteq b$, then P must also hold for b. In other words, affine resources can always be "added" to any proposition without invalidating it. This has the effect that those resources can be "thrown away" in the logic. In fact, it is a well-kept secret of affine BI logics that they do not actually ever really "throw away" the resources that are being dropped. Instead, they just forget about them, stop tracking them, and pick some other unsuspecting proposition to carry them along (SEP-ELIM in §2.1). Because all propositions are upwards-closed w.r.t. extension

order, that proposition cannot defend itself; it is forced to accept the additional resources. Linear propositions on the other hand do not satisfy an up-closure property; they do not have to put up with additional resources, forcing the logic to track resources precisely. By controlling the up-closure, we have control over which resources can be silently added to propositions (and thus seemingly dropped), and which have to be carried along explicitly.

Based on this, we define the set of propositions as a quotient of the monotone predicates (w.r.t. $\sqsubseteq$) over some uORA M:

$$\mathit{Prop} \triangleq M \xrightarrow{\mathrm{mon}_\sqsubseteq} \mathbf{Prop}/_{\equiv}$$

$$P \vdash Q \triangleq \forall a.\, (\mathcal{V}(a) \land P(a)) \Rrightarrow Q(a) \qquad\qquad P \equiv Q \triangleq (P \vdash Q) \land (Q \vdash P)$$

This is just like the definition of Iris in §3.1, except that the extension order ($\preccurlyeq$) got replaced by the user-defined order ($\sqsubseteq$).

We can now define all the connectives required by MoSeL as well as a generic notion of "ownership of a resource"; some examples are shown below:

$$P * Q \triangleq \lambda a.\, \exists a_1, a_2.\, a = a_1 \cdot a_2 \land P(a_1) \land Q(a_2) \qquad \mathsf{emp} \triangleq \lambda a.\, \varepsilon \sqsubseteq a$$

$$\boxdot P \triangleq \lambda a.\, P(|a|) \qquad \mathsf{Own}\,(b) \triangleq \lambda a.\, b \sqsubseteq a$$

For these definitions to be well-formed, we have to show that $\boxdot$ is monotone with respect to $\sqsubseteq$. This relies on ORA-CORE-MONO.

These definitions of the connectives are obtained from the ones used for Iris by replacing $\preccurlyeq$ with $\sqsubseteq$. As a consequence, we can recover Iris from these definitions by taking a uRA and turning it into a uORA where we let $\sqsubseteq$ be the extension order $\preccurlyeq$. Notice that $\varepsilon \preccurlyeq a$ trivially holds, corresponding to the fact that emp is equivalent to True in Iris.

In fact, we can recover every logic that we have discussed so far in the paper: The simple intuitionistic separation logic from §2.1 is just a special case of Iris, as discussed in §3.1. The classical separation logic can be recovered by picking $\sqsubseteq$ to be equality, which makes every resource linear. Both emp and Own (_) then effectively use equality to compare resources, just as classical separation logics do. Finally, Fairis can be recovered by taking a uRA M and constructing a uORA on *pairs* of elements as follows:

$$(a_1, b_1) \cdot (a_2, b_2) \triangleq (a_1 \cdot a_2, b_1 \cdot b_2) \qquad \mathcal{V}((a, b)) \triangleq \mathcal{V}(a) \land \mathcal{V}(b)$$

$$(a_1, b_1) \sqsubseteq (a_2, b_2) \triangleq a_1 \preccurlyeq a_2 \land b_1 = b_2 \qquad |(a, b)| \triangleq (|a|, \varepsilon)$$

We can then define $\mathsf{OwnAffine}(a) \triangleq \mathsf{Own}\,((a, \varepsilon))$ and $\mathsf{OwnLinear}(b) \triangleq \mathsf{Own}\,((\varepsilon, b))$ to recover both affine and linear ownership.

Instantiating MoSeL. Next, we have to show the rules of a MoBI logic are satisfied for all instances of our uORA model. This will also motivate the axioms we require for the order ($\sqsubseteq$). Showing the laws for the BI connectives is straightforward. For the laws of $\boxdot$, the following table shows which of the uORA axioms is required to show the individual proof rules:

Proof rule	$\boxdot$-MONO	$\boxdot$-IDEM	$\boxdot$-ABSORB	$\boxdot$-EMP	$\boxdot$-CONJ-ELIM
uORA axiom	–	ORA-CORE-IDEM	ORA-$\sqsubseteq$-OP-CORE	ORA-$\sqsubseteq$-UNIT-CORE	ORA-CORE-ID

Together with ORA-CORE-MONO being used to show that $\boxdot P$ is monotone, all uORA axioms concerning the core are needed exactly once in this model, and every proof rule of $\boxdot$ (except for $\boxdot$-MONO) reflects one uORA axiom.

Altogether, this shows that MoSeL can handle a large variety of separation logics spanning from linear to affine. In particular, the axioms that uORAs impose on the core provide sufficient

conditions for satisfying the laws of persistence. This has been helpful when instantiating MoSeL with existing logics that already define all the BI connectives, but do not already come with a persistence modality.

4 GOING BEYOND IRIS

In this section, we show that MoSeL can be instantiated for logics beyond Iris. First of all, we describe the instance for iGPS [Kaiser et al. 2017], a logic defined *inside* Iris that uses its own notion of propositions (§4.1). This turns out to give rise to a general construction for defining richer and richer MoBI logics. Second, we describe two MoSeL instances for logics that have nothing to do with Iris at all: CFML [Charguéraud 2018] (§4.2) and CHL [Chen et al. 2015] (§4.3).

4.1 iGPS and Monotone Predicates

In prior work, Kaiser et al. [2017] presented iGPS: a separation logic for reasoning about programs under weak (or relaxed) memory models. iGPS is based on Iris, but instead of plain Iris propositions, iGPS propositions are monotone Iris predicates over a notion of *views*. Views describe the knowledge of the current thread with regard to the state of shared memory, and they have a canonical pre-order ($\sqsubseteq$) expressing how this knowledge may evolve over time.

All the usual separation logic connectives can be defined on these view predicates, permitting iGPS to be used as a higher-level logic without reasoning explicitly about views. Most of the iGPS connectives just perform pointwise lifting of their Iris counterparts. The exceptions are implication and magic wand, which need to be upward closed w.r.t. $\sqsubseteq$, as usual in a Kripke-style model:

$$\Phi \Rightarrow \Psi \triangleq \lambda i.\, \forall j.\, \ulcorner i \sqsubseteq j \urcorner \Rightarrow \Phi(j) \Rightarrow \Psi(j) \qquad \Phi \mathrel{-\!\!*} \Psi \triangleq \lambda i.\, \forall j.\, \ulcorner i \sqsubseteq j \urcorner \Rightarrow \Phi(j) \mathrel{-\!\!*} \Psi(j)$$

View predicates indeed form a proper logic, and one can reason directly inside that logic instead of appealing to the underlying model.

However, in the Coq development of Kaiser et al. [2017], mechanized iGPS proofs were done by unfolding the connectives into their Iris definitions and proving these using IPM. So, for example, to prove the iGPS entailment $\Phi * (\Phi \mathrel{-\!\!*} \Psi) \vdash \Psi$, one would unfold it into the Iris entailment

$$\forall i.\, \Phi(i) * (\forall j.\, \ulcorner i \sqsubseteq j \urcorner \Rightarrow \Phi(j) \mathrel{-\!\!*} \Psi(j)) \vdash \Psi(i) \qquad \text{(EXAMPLE-MONPRED-UNFOLD)}$$

and prove that using IPM. For proving the primitive rules of iGPS, a proof by unfolding is sometimes inevitable, but when using iGPS as a logic (*e.g.*, to prove Hoare-style program specifications), this approach is rather unsatisfactory because views (like i and j) and relations between them show up everywhere. So, while view predicates provided a good abstraction on paper, this abstraction was violated in Coq. Using MoSeL one can prove this statement at the appropriate level of abstraction:

```
Lemma example_monpred {I PROP} (Φ Ψ : monPred I PROP) : Φ ∗ (Φ -∗ Ψ) ⊢ Ψ.
Proof. iIntros "[H1 H2]". iApply "H2". iAssumption. Qed.
```

After the `iIntros` tactic, the goal looks like (note that no views are visible):

```
"H1" : Φ
"H2" : Φ -∗ Ψ
--------------------------------------∗
Ψ
```

Let us now see how MoSeL supports this functionality—not only for iGPS, but for any logic in which propositions represent monotone predicates. (In fact, the Coq proof of `example_monpred` above was generic w.r.t. the choice of monotone predicate.)

General monotone predicates. Let $(\mathcal{I}, \sqsubseteq)$ be a pointed (*i.e.,* inhabited) pre-ordered set, and *Prop* a MoBI. Then, the MoBI of monotone predicates over $\mathcal{I}$ and *Prop* is given by:

$$\mathcal{I} \xrightarrow{\text{mon}} \mathit{Prop} \triangleq \{\Phi \in \mathcal{I} \to \mathit{Prop} \mid \forall i, j \in \mathcal{I}.\, i \sqsubseteq j \Rightarrow \Phi(i) \vdash \Phi(j)\}$$

The connectives are defined just as explained above for iGPS. With these definitions, one can prove that all the MoBI axioms are verified, including those of the persistence modality $\boxdot$ (Figure 4).

Proofs in the model of monotone predicates. When using monotone predicates, it is sometimes necessary to unfold the definition of the connectives of $\mathcal{I} \xrightarrow{\text{mon}} \mathit{Prop}$ into a statement in *Prop*. Unlike the way we have done that in EXAMPLE-MONPRED-UNFOLD, one typically does not want to unfold the statement entirely, but rather in a lazy fashion. This can be done in MoSeL as follows:

```
Lemma example_monpred {I PROP} (Φ Ψ : monPred I PROP) : Φ ∗ (Φ -∗ Ψ) ⊢ Ψ.
Proof. iStartProof PROP. iIntros (i) "[H1 H2]". (* ... *)
```

Using the `iStartProof` tactic, one can instruct MoSeL to translate the statement into another logic (this may fail, if the statement cannot be translated). The result of calling that tactic is:

```
∀ i : I, Φ i ∗ (Φ -∗ Ψ) i -∗ Ψ i
```

Note that only the outer layer of the statement has been unfolded. This lazy approach to unfolding keeps the goals short and readable, and minimizes explicit reasoning about views.

Interfacing with the underlying logic. One of the key features of MoSeL's support for monotone predicates is that it allows one to piggyback on the features of the underlying logic. For example, iGPS inherits the assertions for resource ownership and impredicative invariants from Iris. This is done using the embedding $\lceil \_ \rceil : \mathit{Prop} \to (\mathcal{I} \xrightarrow{\text{mon}} \mathit{Prop})$, defined as $\lceil P \rceil \triangleq \lambda i.\, P$.

When using the embedding, one often ends up with goals and premises where a connective appears below a $\lceil \_ \rceil$, for example $\lceil P \mathrel{-\!\!*} Q \rceil$. In order to conveniently carry out proofs, it is essential that one does not need to manually distribute the embedding. Let us consider a generic example:

```
Lemma example_monpred_2 {I PROP} (Φ : monPred I PROP) (P Q : PROP) :
  ⌜ P -∗ Q ⌝ -∗ ⌜ □ P ⌝ -∗ <affine> Φ -∗ ⌜ P ∗ Q ⌝.
Proof. iIntros "H1 #H2 H3". iFrame "H2". iApply "H1". iAssumption. Qed.
```

After the `iIntros` the goal is as follows:

```
"H2" : ⌜ P ⌝
--------------------------------------□
"H1" : ⌜ P -∗ Q ⌝
"H3" : <affine> Φ
--------------------------------------∗
⌜ P ∗ Q ⌝
```

Note that `iIntros` automatically recognizes that `⌜ □ P ⌝` is intuitionistic, even though the $\Box$ modality appears below the embedding. Furthermore, as the next step shows, the `iFrame` tactic will automatically distribute the embedding to frame `⌜ P ⌝`, *i.e.,* use the rule $\lceil P * Q \rceil \dashv\vdash \lceil P \rceil * \lceil Q \rceil$. The resulting goal after framing is:

```
"H2" : ⌜ P ⌝
--------------------------------------□
"H1" : ⌜ P -∗ Q ⌝
"H3" : <affine> Φ
--------------------------------------∗
⌜ Q ⌝
```

In much the same way, the `iApply` tactic will automatically distribute the magic wand over the embedding, making use of the rule $\lceil P \mathrel{-\!\!*} Q \rceil \dashv\vdash \lceil P \rceil \mathrel{-\!\!*} \lceil Q \rceil$ to implicitly turn the goal into $\lceil \texttt{P} \rceil$. As we will see in §5.3, MoSeL's support for automatically distributing connectives is not limited to the embedding $\lceil \_ \rceil$ but can be used for any connective.

Modalities. As usual when adding a new index to a Kripke-style semantics, we can define two modalities on monotone predicates expressing that a proposition holds for *all views* (we call this *objective*), or that it holds for *some view* (we call this *subjective*):

$$\langle obj \rangle\, \Phi \triangleq \lambda i.\, \forall j.\, \Phi(j) \qquad\qquad \langle subj \rangle\, \Phi \triangleq \lambda i.\, \exists j.\, \Phi(j)$$

We again use the modality to define a class of propositions: Φ is *objective* if $\Phi \vdash \langle obj \rangle\, \Phi$.[7] Objective propositions are used in iGPS when reasoning about propositions that are transferred between threads: Objective propositions, unlike general iGPS propositions, hold in *all* threads.

This begs the question: How can we prove $\langle obj \rangle\, P$ in MoSeL? In §2.5 we have already seen how one can use MoSeL's `iModIntro` tactic to introduce various other modalities, like $\langle affine \rangle$, $\boxdot$ and $\square$. In fact, `iModIntro` can also be used to introduce $\langle obj \rangle$. The corresponding rule for this tactic is:

$$\textsc{tac-}\langle obj \rangle\textsc{-intro}\quad \frac{\Gamma; \Pi \vdash \Psi \qquad \text{objective}(\Gamma) \qquad \text{objective}(\Pi)}{\Gamma; \Pi \vdash \langle obj \rangle\, \Psi}$$

That is, to introduce $\langle obj \rangle\, \Psi$, which amounts to saying that Ψ is independent of the view, it must be the case that all hypotheses are objective, too, *i.e.,* they must also be independent of the view. All of this is provided automatically by using the general construction for monotone predicates.

The actual version of `iModIntro` is in fact a bit more liberal than the above rule: it will automatically clear the hypotheses that are not objective (provided the cleared hypotheses are affine, or the goal is absorbing). We will come back to this in §5.3.

4.2 CFML

CFML [Charguéraud 2011, 2018] is a verification framework for higher-order imperative programs, which scales up to handle realistic data structures and algorithms implemented in OCaml, *e.g.,* Union Find, Hashtable, DFS, Dijsktra's shortest path, etc. Like Iris, CFML implements its heap assertions via a shallow embedding in Coq. However, unlike Iris, which is an affine separation logic based on the manipulation of weakest preconditions, CFML is a linear separation logic based on the manipulation of Hoare triples.[8] The fact that CFML was designed prior to and completely independently from Iris makes CFML a particularly interesting target for MoSeL.

In what follows, we first explain how to exploit MoSeL to improve CFML's `hsimpl` simplification tactic for entailments, and then explain how to apply the same methodology to contribute a new tactic able to automate entailments involving *read-only permissions* [Charguéraud and Pottier 2017].

CFML for plain separation logic. As CFML is based on Hoare triples, program verification involves numerous applications of the frame and consequence rules. Typically, one wants to prove a triple of the form $\{P\}\ e\ \{\Phi\}$ from a hypothesis of the form $\{P'\}\ e\ \{\Phi'\}$. To relate the two, CFML exploits

[7] This turns out to be equivalent with $\langle subj \rangle\, \Phi \vdash \Phi$, so there is no separate class for the subjectivity modality.

[8] CFML 1.0 [Charguéraud 2011] parses OCaml code and produces Coq axioms for describing the *characteristic formulae* capturing the semantics of the code. In contrast, CFML 2.0 [Charguéraud 2018] manipulates code expressed in a deep embedding and computes characteristic formulae inside Coq, in an axiom-free fashion. In both cases, CFML's separation logic is entirely formalized in Coq. The present work is based on CFML 2.0.

the following rule, which combines framing with the rule of consequence:

$$\textsc{consequence-frame}\quad \frac{\{P'\}\ e\ \{\Phi'\} \qquad P \vdash P' * Q \qquad \forall v.\ \Phi'(v) * Q \vdash \Phi(v)}{\{P\}\ e\ \{\Phi\}}$$

CFML then relies on a tactic called `hsimpl` for proving the two entailments and instantiating Q. Indeed, observe that, in the rule above, Q cannot be inferred from either the goal or the Hoare triple hypothesis. CFML's solution is to introduce a unification variable (*evar*), which subsequently gets instantiated with the result of subtracting P' from P. Using such a unification variable is the poor man's approach for computing a subtraction without a magic wand. This approach falls short when extending the logic with more advanced features such as read-only permissions.

Contrary to CFML's `hsimpl` tactic, MoSeL does support the magic wand. Thus, with MoSeL we can exploit the *ramified frame rule* [Hobor and Villard 2013], which uses a magic wand to *eliminate* the auxiliary variable Q. Concretely, the ramified frame rule merges the second and third premises from rule CONSEQUENCE-FRAME into a single premise as follows:

$$\textsc{ramified-frame}\quad \frac{\{P'\}\ e\ \{\Phi'\} \qquad P \vdash P' * (\forall v.\ \Phi'(v) \mathrel{-\!\!*} \Phi(v))}{\{P\}\ e\ \{\Phi\}}$$

The entailment from the ramified frame rule can be easily and automatically processed by MoSeL's tactics, in three stages:

(1) A tactic performing basic simplifications is first invoked, mainly for turning the entailment into a curried form. For example, the tactic would turn $(\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2) \vdash (\ell_1 \mapsto v_1 * (\forall v.\ \mathsf{emp} \mathrel{-\!\!*} \ell_2 \mapsto v_2))$ into $\mathsf{emp} \vdash \ell_1 \mapsto v_1 \mathrel{-\!\!*} (\ell_2 \mapsto v_2 \mathrel{-\!\!*} (\ell_1 \mapsto v_1 * (\forall v.\ \mathsf{emp} \mathrel{-\!\!*} \ell_2 \mapsto v_2)))$. The curried form makes it easier to subsequently introduce the heap predicates one by one.
(2) Multiple calls to the `iIntros` and `iFrame` tactics are then issued for introducing hypotheses in the spatial context, and for cancelling them out in the goal when possible.
(3) After all hypotheses are introduced, the remaining goal could be a unification variable. Such a variable might originate, *e.g.*, from the application of the Hoare logic rule for a sequence $e_1; e_2$, for which the intermediate state needs to be inferred. When the goal is a unification variable, MoSeL's `iAccu` tactic is invoked for solving the goal by instantiating the unification variable as the separating conjunction of the resources remaining in the spatial context.

Instantiating MoSeL with CFML's assertion logic was relatively straightforward. All the connectives needed by MoSeL were either already existing or easy to add to CFML. In order to instantiate MoSeL, we used the "trivial" definition $\boxdot P \triangleq \lambda\sigma.\ P(\emptyset)$, which we also used for the simple classical and intuitionistic separation logics (PERSISTENCE-SIMPLE in §2.4). As we have seen, this definition enjoys all the axioms of the persistence modality (see Figure 4).

As a case study for our CFML instantiation of MoSeL, we ported a CFML file containing proofs of functions manipulating mutable linked lists, replacing calls to CFML tactics (including `hsimpl`) with uses of MoSeL tactics. The fact that CFML uses Hoare triples, instead of using a weakest precondition connective as Iris does, prevented us from directly proving Hoare triples using MoSeL, but we were able to put MoSeL to good use in verifying side-conditions of the ramified frame rule.

CFML for separation logic with read-only permissions. Charguéraud and Pottier [2017] present a variant of separation logic that supports *temporary read-only permissions*. Concretely, for any separation logic proposition P, the proposition $\mathrm{RO}(P)$ describes a duplicable, read-only version of P. Charguéraud and Pottier showed that read-only permissions were particularly useful for verifying programs that rely on some memory regions not being modified, and that, at the same time, they avoid the bookkeeping of fractional permissions.

Read-only permissions are introduced by the *read-only frame rule*:

$$\textsc{read-only-frame}\quad \frac{\{P * \mathrm{RO}(Q)\}\ e\ \{\Phi\} \qquad \mathrm{normal}(Q)}{\{P * Q\}\ e\ \{v.\ \Phi(v) * Q\}}$$

This rule allows one to turn a proposition Q into a read-only version $\mathrm{RO}(Q)$ (called a *read-only permission*) for the scope of the frame. The side-condition $\mathrm{normal}(Q)$ asserts that Q does not contain any read-only permissions—it is needed for soundness [Charguéraud and Pottier 2017]. The read-only permission $\mathrm{RO}(Q)$ can be duplicated or discarded at will, but it can never "come back" through the postcondition Φ. Indeed, a key invariant for soundness is that read-only permissions never appear in postconditions. Otherwise, one could end up with the proposition $Q * \mathrm{RO}(Q)$, which is unsound.[9] The side-condition $\mathrm{normal}(Q)$ prevents such situations from occurring.

Trying to generalize CFML's `hsimpl` approach to automatically instantiate the read-only frame rule, we run into problems because, unlike in the case of plain separation logic, the framed proposition Q can no longer be computed by subtraction.

Fortunately, exploiting the flexibility of MoSeL, we are able to devise a version of the read-only frame rule that is amenable to automation. First, we introduce a modality, written "$\langle \mathit{normal} \rangle\ P$", which describes a resource that satisfies P and contains no read-only permissions. Using this modality, we can encode $\mathrm{normal}(P)$ as $P \vdash \langle \mathit{normal} \rangle\ P$ and leverage MoSeL's built-in support for reasoning conveniently about modalities. Second, we can give a ramified version of the read-only frame rule. Its side condition is an entailment involving magic wands and the $\langle \mathit{normal} \rangle\ P$ modality, and to solve such entailments, we extend MoSeL's `iFrame` tactic with a carefully-crafted set of type class instances for guiding the simplification process.

4.3 CHL

Chen et al. [2015] describe *Crash Hoare Logic* (CHL), a separation logic they used to verify a crash-safe file system called FSCQ in Coq. CHL has many tactics for automatically framing out propositions and unfolding steps of Hoare triple proofs which are used throughout the Coq development. We instantiated the MoSeL interface with CHL, and used the proof mode to re-do the proofs of the "buffer cache" component of FSCQ. As with CFML, our goal was to determine whether some of the automation in CHL could be reproduced by building on the primitive tactics of MoSeL.

For instance, CHL features a tactic `cancel`, which (among other things) acts on a goal of the form $P \vdash Q$ by (1) finding any pure propositions in P and eliminating them into the Coq context, (2) eliminating existentials $(\exists x.\ R)$ in P, (3) framing propositions in P, and (4) introducing existentials in Q with Coq unification variables. To implement this functionality, the tactic repeatedly applies lemmas to rearrange the order of separating conjuncts in P and Q and commute existentials with separating conjunction in order to obtain a form where elimination rules can be applied.

We can reproduce much of what this tactic does using a combination of `iDestruct`, `iFrame`, and `iExists`, much like in the tactic for the ramified frame rule described in §4.2. One benefit of writing the tactic using the MoSeL primitives is extensibility. For example, in FSCQ there is a predicate on propositions, `crash_xform`, which distributes over separating conjunction, *i.e.,* `crash_xform(P * Q) ⊣⊢ crash_xform(P) * crash_xform(Q)`. The authors of FSCQ define a tactic `xform_norm` which repeatedly rewrites goals involving `crash_xform` using this and other distributivity rules to put them in a form where the `cancel` tactic can do framing. However, with MoSeL we can instead extend `iDestruct` and `iFrame` to automatically make use of this distributivity property without needing to first apply a tactic like `xform_norm`. We will see in §5.3 how this extensibility is supported.

[9] With the proposition $Q * \mathrm{RO}(Q)$, one could modify Q, then read a deprecated state using $\mathrm{RO}(Q)$.

5 IMPLEMENTATION IN COQ

Like the original IPM, MoSeL is implemented entirely in Coq and does not involve any OCaml plugins or modifications of the Coq source code. To achieve that, it uses *canonical structures* [Garillot et al. 2009] to parameterize MoSeL by a MoBI logic, *computational reflection* to efficiently implement operations on the proof mode contexts, *logic programming* using type classes [Sozeau and Oury 2008; Spitters and van der Weegen 2011] to make the tactics very modular, and the *Ltac tactic definition language* [Delahaye 2000] to wrap things up.

In this section we briefly describe how the Coq implementation of MoSeL is parameterized by a MoBI (§5.1), followed by a recap of the way tactics are implemented in IPM and MoSeL (§5.2), to finally show how we have made MoSeL extensible (§5.3).

5.1 MoBI Interfaces

In order to make MoSeL independent of Iris, we have parameterized all definitions and tactics by the following canonical structure, describing a MoBI logic with a persistence modality:

```
Structure bi := Bi {
  bi_car :> Type;                          bi_pure : Prop → bi_car;
  bi_entails : bi_car → bi_car → Prop;     bi_sep : bi_car → bi_car → bi_car;
  bi_emp : bi_car;                         bi_persistently : bi_car → bi_car;
  (* other operations and axioms *)
}.
```

This canonical structure bundles the carrier (type of propositions) of the MoBI logic `bi_car` together with all MoBI operations and axioms. For each BI logic (*e.g.*, Iris, CFML, CHL), we declare a canonical instance, so canonical structure search can be used to infer this instance given its carrier.

5.2 Implementation of Tactics

The cornerstone of MoSeL (and IPM before it) is the deeply-embedded representation of separation logic contexts. These contexts are represented by association lists `env` with strings as keys:

```
Record envs (PROP : bi) :=
  Envs { env_intuitionistic : env PROP; (* the intuitionistic context Γ *)
         env_spatial : env PROP;        (* the spatial context Π *)
         env_counter : positive         (* a counter for fresh name generation *) }.
```

Using this representation, we can now formalize the MoSeL entailment ⊩ in Coq:

```
Definition envs_entails {PROP} (Δ : envs PROP) (Q : PROP) : Prop :=
  ⌜envs_wf Δ⌝ ∧ □ [∧] env_intuitionistic Δ ∗ [∗] env_spatial Δ ⊢ Q.
```

This definition is nearly the same as MoSeL-entail, but includes the condition `envs_wf Δ` to ensure that the contexts are well-formed, *i.e.,* all hypotheses have unique names. The connectives `[∧]` and `[∗]` fold a conjunction and separating conjunction, respectively, over a list.

As in IPM, most MoSeL tactics are defined as Coq lemmas justifying context manipulations. For example, the lemma corresponding to the tactics `iSplitL` and `iSplitR` (see §2.2) for introduction of separating conjunction is as follows:

```
Class FromSep {PROP : bi} (P Q1 Q2 : PROP) := from_sep : Q1 ∗ Q2 ⊢ P.
Instance from_sep_sep P1 P2 : FromSep (P1 ∗ P2) P1 P2.
Lemma tac_sep_split Δ Δ1 Δ2 d js P Q1 Q2 :
  FromSep P Q1 Q2 → envs_split d js Δ = Some (Δ1,Δ2) →
  envs_entails Δ1 Q1 → envs_entails Δ2 Q2 → envs_entails Δ P.
```

The first premise `FromSep P Q1 Q2` makes sure that the goal is indeed a separating conjunction (we will explain the flexibility of using a type class for this purpose in §5.3). The second premise comprises the function `envs_split`, which splits the context Δ into contexts Δ_1 and Δ_2 for both separating conjuncts. The context Δ_1 has the spatial hypotheses named `js`, while Δ_2 has the remaining spatial hypotheses (or vice versa, depending on $\mathsf{d} \in \{\texttt{Left}, \texttt{Right}\}$). The function `envs_split` is written in Coq, which enables us to prove the second premise efficiently by computation. The last two premises are the new goals for the conjuncts `Q1` and `Q2`.

The end-user tactics `iSplitL` and `iSplitR` are defined as wrappers that simply apply the lemma `tac_sep_split`. These wrappers are written in Ltac and are responsible for input handling (*e.g.,* parsing of arguments of the tactic) and error handling.

5.3 Making MoSeL Extensible

Although MoSeL provides off-the-shelf tactical support for the MoBI connectives of any logic, this is usually not enough. In order to do proofs in a specific logic, one needs tactical support for the bespoke connectives of that logic. For example, Iris comes with modalities for step-indexing and resource ownership, iGPS comes with embeddings and the modalities for views, and so on.

Of course, one could write custom MoSeL tactics for these connectives, but that would be rather cumbersome. Instead, as we will show in this section, MoSeL tactics can be extended to take new connectives into account. MoSeL's support for extensibility comes in the following ways:

- Many tactics demand certain propositions to be intuitionistic, persistent or affine. We represent these requirements using type classes, so that one can easily define additional instances to make MoSeL aware that logic-specific connectives are intuitionistic, persistent or affine.
- All tactics for introduction and elimination of logical connectives (*e.g.,* `iDestruct`, `iIntros`, `iSplitL`) are parameterized by a type class. This type class is used to turn the goal (in the case of introduction) or hypotheses (in the case of elimination) into the right shape. By declaring appropriate instances of these type classes, one can extend the behavior of these tactics.
- MoSeL comes with a generic tactic for introduction and elimination of modalities, which subsumes several ad-hoc tactics for dealing with modalities in IPM, and supports a variety of modalities that appear in other logics.

Parameterization of tactics by type classes. Let us demonstrate this by an example. The tactics `iSplitL`/`iSplitR` do not demand the goal to be a separating conjunction. Instead, if we let `P` be the goal, it demands an instance of the following type class:

```
Class FromSep {PROP : bi} (P Q1 Q2 : PROP) := from_sep : Q1 ∗ Q2 ⊢ P.
```

This surprisingly simple trick, which was already present in the original IPM, allows for a lot of flexibility. By declaring additional instances of this type class, one can easily extend `iSplitL`/`iSplitR` to distribute connectives. Below we list some instances of this type class that make this possible:

```
Instance from_sep_sep P1 P2 : FromSep (P1 ∗ P2) P1 P2.
Instance from_sep_affinely P Q1 Q2 :
  FromSep P Q1 Q2 → FromSep (<affine> P) (<affine> Q1) (<affine> Q2).
Instance from_sep_embed `{BiEmbed PROP PROP'} P Q1 Q2 :
  FromSep P Q1 Q2 → FromSep ⌈P⌉ ⌈Q1⌉ ⌈Q2⌉.
```

Using these instances, `iSplitL`/`iSplitR` is able to introduce separating conjunctions, even if they appear below $\langle \mathit{affine} \rangle$ modalities or below an embedding $\lceil \_ \rceil$. Using similar instances, one can handle all modalities that distribute over separating conjunctions.

In the original IPM, we only scratched the surface of what was possible using this approach. In MoSeL, we have made *every* tactic parametric by such a type class, and that way, we are even able

Modality	Action on intuitionistic context	Action on spatial context
$\langle \mathit{affine} \rangle$	None	All hypotheses must be affine
$\boxdot$	None	Remove all hypotheses
$\square$	None	All hypotheses must be affine, and are removed
$\lceil \_ \rceil$	All hypotheses must appear under an embedding, which is stripped	
$\langle \mathit{obj} \rangle$	All hypotheses must be objective	
$\triangleright$	A later modality $\triangleright$ may be stripped off each hypothesis	

Fig. 10. The action performed on the the proof mode context for introducing a modality.

to handle complicated features such as lazy unfolding of monotone view predicates as we have seen in §4.1 without much difficulty. After all, since every tactic is parameterized by such a type class, we have the full power of logic programming via type classes at our disposal for extending the functionality of MoSeL tactics. Moreover, since these type classes are parameterized by arbitrary MoBI logics, we can easily support connectives such as $\lceil \_ \rceil$ that involve multiple MoBI logics.

Generic tactics for modalities. MoSeL provides generic support for introduction and elimination of modalities in the form of the `iModIntro` and `iMod` tactics. In this section, we will focus on the tactic `iModIntro` for introduction of modalities. In §2.5 and §4.1 we have already seen how this tactic operates on the $\boxdot$, $\square$, $\langle \mathit{affine} \rangle$ and $\langle \mathit{obj} \rangle$ modalities. What all of these modalities M have in common is that, to introduce $(M\ P)$, a certain action should be performed on the proof mode context. Figure 10 gives an overview of these actions for the modalities we have seen in this paper, as well as the later $\triangleright$ modality for step-indexing.

In order to register a modality so that `iModIntro` will recognize and introduce it, one has to declare an instance of the type class `FromModal` that contains the following information:

- An action that should be performed on the intuitionistic context.
- An action that should be performed on the spatial context.
- A proof of a set of laws that should hold to justify said actions on the contexts.

These actions on the contexts can be of the following kinds:

(1) Introduction is only allowed when the context is empty.
(2) Introduction is only allowed when all hypotheses satisfy a property described by a type class.
(3) Introduction will perform a transformation on each hypothesis, described by a type class.
(4) Introduction will remove all hypotheses from the context.
(5) Introduction will perform no action on the context.

In Coq, a modality is represented as a dependently typed record called `modality`:

```
Inductive modality_action (PROP₁ : bi) : bi → Type :=
  | MIEnvIsEmpty {PROP₂ : bi} : modality_action PROP₁ PROP₂
  | MIEnvForall (C : PROP₁ → Prop) : modality_action PROP₁ PROP₁
  | MIEnvTransform {PROP₂ : bi} (C : PROP₂ → PROP₁ → Prop) : modality_action PROP₁ PROP₂
  | MIEnvClear {PROP₂} : modality_action PROP₁ PROP₂
  | MIEnvId : modality_action PROP₁ PROP₁.
Record modality (PROP₁ PROP₂ : bi) := Modality {
  modality_car :> PROP₁ → PROP₂;
  modality_intuitionistic_action : modality_action PROP₁ PROP₂;
  modality_spatial_action : modality_action PROP₁ PROP₂;
  (* The modality laws, which depend on the fields modality_intuitionistic_action
     and modality_spatial_action *) }.
```

This record is parameterized by the domain $\texttt{PROP}_1$ and co-domain $\texttt{PROP}_2$ of the modality. It contains the carrier `modality_car : PROP₁→PROP₂`, together with the actions that should be performed when the modality is introduced. Actions are represented using the inductive type `modality_action`, which contains constructors for the actions 1–5 that we have described before.

For the case of the ⟨*affine*⟩ modality, the actions are `MIEnvId` (action 5) for the intuitionistic context, and `MIEnvForall Affine` (action 2) for the spatial context. Here, `Affine` is a predicate (in the form of a type class), which describes that a proposition is affine. In order to register this modality, we need to define an instance of the `FromModal` type class:

```
Class FromModal {PROP₁ PROP₂ : bi} (M : modality PROP₁ PROP₂) (P : PROP₂) (Q : PROP₁) :=
  from_modal : M Q ⊢ P.
Instance from_modal_affinely P : FromModal modality_affinely (<affine> P) P.
```

The `FromModal` type class is very similar to the `FromSep` type class—it states how a goal `P` can be turned into a modality `M` and proposition `Q`. When presented with a goal `P`, the `iModIntro` tactic will look for an instance `FromModal M P Q`, to transform the goal into `Q` and to perform the actions of `M` on the MoSeL contexts. Since the actions `MIEnvForall` and `MIEnvTransform` take a type class `C` as their argument, *and* they themselves also appear (indirectly via `modality`) as the argument of the type class `FromModal`, we dub the technique that we use here "higher-order type classes".

6 RELATED WORK

Tactics for separation logic. In the past decade, there has been much progress on mechanized versions of separation logic in proof assistants, with a focus that ranges from strong proof automation to the ability to deal with realistic programming languages. We will mainly discuss tactic languages for separation logic, rather than approaches for performing proofs in the model of separation logic (*e.g.,* [Nanevski et al. 2010; Sergey et al. 2015a]), since the latter are usually logic-specific.

Some of the earliest work on tactics for separation logic is by Appel [2006], who created a family of tactics that help out with the basic bookkeeping that is involved while reasoning in a separation logic. This work was later extended by McCreight [2009].

Bengtson et al. [2012] have continued this line of work by implementing similar tactics in a more language-independent fashion; they acheived this by parameterizing the framework by a separation algebra [Calcagno et al. 2007; Dockins et al. 2009]. An interesting aspect of their work is that most of their tactics operated on propositions that were in a so-called *normal form* that consists of a list of pure propositions and a list of spatial propositions. By keeping propositions in such a normal form, it turned out to be easier to implement tactics. A more recent direction in this line of work is by Cao et al. [2018], who developed an improved version of this approach for VST, a framework based on separation logic for proving correctness of C programs.

Most of these implementations, however, considered a fairly limited fragment of the assertion language of separation logic. In contrast, Krebbers et al. [2017b] developed a tactic language for Iris in Coq that supports a variety of modalities and other features [Jung et al. 2016, 2018b, 2015; Krebbers et al. 2017a], including extensive support for magic wands, higher-order quantification, the persistence and later modality, ghost ownership, and impredicative invariants. An important aspect of the work of Krebbers et al. [2017b] was to represent contexts not just as lists, but as association lists so that users can easily refer to hypotheses by name. On top of that, it uses computational reflection to improve efficiency of basic operations on the proof context.

Tassarotti et al. [2017] modified the proof mode of Krebbers et al. [2017b] to support Fairis, their linear variant of Iris. Like MoSeL, the Fairis proof mode divided the proof context into intuitionistic and spatial parts, and various tactics tracked when propositions could be duplicated, dropped, and split between subgoals of a proof. They decomposed the intuitionistic modality as the composition

of the affine modality and another modality, which they called relevance. However, they did not carefully delineate what properties of this modality were needed to support the features of the proof mode, as we have done here with our axiomatization of persistence. Moreover, their changes were tailored to their particular logic and were not compatible with the original Iris or with any of the other MoSeL instances we have described here. Also, because their proof mode was based on an incompatible fork of an early version of IPM, it lacked many of the miscellaneous improvements that were subsequently made to IPM. Indeed, one benefit of the logic-independent approach taken by MoSeL is that improvements to the proof mode can be shared across different logics.

Generic models of BI and separation logic. There is a large literature on semantic models of BI [Pym 2002]. The most common type of model axiomatizes an abstract notion of "resource" as some algebraic structure, which is then used to give a Kripke-style possible worlds model of propositions. Pym et al. [2004] give such a model of BI in which resources are elements of a pre-ordered partial commutative monoid, where the monoid operation is monotone with respect to the pre-order. Subsequent work on the semantics of separation logic has considered variants and restrictions of this general notion of resource [Atkey 2011; Calcagno et al. 2007; Dockins et al. 2009] with additional axioms often motivated by properties of the "canonical" heap model of separation logic. Many recent separation logics have either not considered an order structure on resources or just use the extension order, instead of the arbitrary pre-order considered by Pym et al. [2004].

Most recently, Cao et al. [2017] have argued for the importance of considering arbitrary orders, and have given generic accounts of both general/classical and affine/intuitionistic separation logics, unifying many previously given models for these kinds of logics. They study in detail how imposing axioms on the order affects the proof rules of the resulting logic. This has inspired us to also extend resource algebras RAs with an order, leading us to *ordered resource algebras* (ORAs, see §3.3).[10] Our goals with ORAs are somewhat different, however. We do not attempt to give a general model of *every* separation logic; rather, we were looking for a uniform treatment of the various affine and non-affine logics we had instantiated MoSeL with. In particular, our focus was on studying the interaction between the axioms for persistence in MoBIs and the axioms for the core in ORAs. Cao et al. [2017], on the other hand, do not consider any modalities.

Modalities are pervasive in separation logics. This is not surprising, since any relation on resources gives rise to a modality, a fact used by Dockins et al. [2008]. Courtault et al. [2016] study BI with modalities that are indexed by resources. These modalities are used to reason about how the state of a system (*e.g.,* a Petri net) evolves over time. In contrast, we are considering the resources available at a single point in time (*e.g.,* during program execution) and use modalities to single out classes of propositions that satisfy structural properties, like being intuitionistic or being affine.

In his soundness proof of a type-and-capability system, Pottier [2013] considers resources equipped with a "core" operation, which he uses to model a duplicability modality and which has in fact inspired the Iris authors [Jung et al. 2016] to adopt a similar notion. The axioms of Pottier's core, however, are significantly different from the ones for RAs/ORAs [Jung et al. 2018b].

ACKNOWLEDGMENTS

We wish to thank Aleš Bizjak and Daniel Gratzer for their feedback on an early version of MoSeL.

This research was supported in part by a European Research Council (ERC) Consolidator Grant for the project "RustBelt", funded under the European Union's Horizon 2020 Framework Programme (grant agreement no. 683289), the French National Research Agency (grant no. ANR-15-CE25-0008), a gift from Oracle Labs, and a Flemish Research Fund (grant no. G.0962.17N).

[10] RAs and ORAs are generalizations of their unital variants, uRAs and uORAs, that do not require a unit ε and where the core $|-|$ is a partial function [Jung et al. 2016, 2018b].

REFERENCES

Andrew W. Appel. 2006. Tactics for separation logic. Available at http://www.cs.princeton.edu/~appel/papers/septacs.pdf.

Robert Atkey. 2011. Amortised resource analysis with separation logic. *LMCS* 7, 2 (2011).

Jesper Bengtson, Jonas Braband Jensen, and Lars Birkedal. 2012. Charge! - A framework for higher-order separation logic in Coq. In *ITP (LNCS)*, Vol. 7406. 315–331.

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

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

Aleš Bizjak and Lars Birkedal. 2018. On models of higher-order separation logic. *ENTCS* 336 (2018), 57–78.

Stephen Brookes. 2007. A semantics for concurrent separation logic. *TCS* 375, 1-3 (2007), 227–270.

Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. 2007. Local action and abstract separation logic. In *LICS*. 366–378.

Qinxiang Cao, Lennart Beringer, Samuel Gruetter, Josiah Dodds, and Andrew W. Appel. 2018. VST-Floyd: A separation logic tool to verify correctness of C programs. *JAR* 61, 1-4 (2018), 367–422.

Qinxiang Cao, Santiago Cuellar, and Andrew W. Appel. 2017. Bringing order to the separation logic jungle. In *APLAS (LNCS)*, Vol. 10695. 190–211.

Arthur Charguéraud. 2011. Characteristic formulae for the verification of imperative programs. In *ICFP*. 418–430.

Arthur Charguéraud. 2018. CFML 2.0. http://www.charglueraud.org/softs/cfml/.

Arthur Charguéraud and François Pottier. 2017. Temporary read-only permissions for separation logic. In *ESOP (LNCS)*, Vol. 10201. 260–286.

Haogang Chen, Daniel Ziegler, Tej Chajed, Adam Chlipala, M. Frans Kaashoek, and Nickolai Zeldovich. 2015. Using Crash Hoare logic for certifying the FSCQ file system. In *SOSP*. 18–37.

Adam Chlipala. 2013. The Bedrock structured programming system: combining generative metaprogramming and Hoare logic in an extensible program verifier. In *ICFP*. 391–402.

Jean-René Courtault, Didier Galmiche, and David J. Pym. 2016. A logic of separating modalities. *TCS* 637 (2016), 30–58.

Pedro da Rocha Pinto, Thomas Dinsdale-Young, and Philippa Gardner. 2014. TaDA: A logic for time and data abstraction. In *ECOOP (LNCS)*, Vol. 8586. 207–231.

David Delahaye. 2000. A tactic language for the system Coq. In *LPAR (LNCS)*, Vol. 1955. 85–95.

Robert Dockins, Andrew W. Appel, and Aquinas Hobor. 2008. Multimodal separation logic for reasoning about operational semantics. In *MFPS*.

Robert Dockins, Aquinas Hobor, and Andrew W. Appel. 2009. A fresh look at separation algebras and share accounting. In *APLAS (LNCS)*, Vol. 5904. 161–177.

François Garillot, Georges Gonthier, Assia Mahboubi, and Laurence Rideau. 2009. Packaging mathematical structures. In *TPHOLs (LNCS)*, Vol. 5674. 327–342.

Aquinas Hobor and Jules Villard. 2013. The ramifications of sharing in data structures. In *POPL*. 523–536.

Jonas Braband Jensen, Nick Benton, and Andrew Kennedy. 2013. High-level separation logic for low-level code. In *POPL*. 301–314.

Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. 2018a. Rustbelt: Securing the foundations of the Rust programming language. *PACMPL* 2, POPL, 66:1–66:34.

Ralf Jung, Robbert Krebbers, Lars Birkedal, and Derek Dreyer. 2016. Higher-order ghost state. In *ICFP*. 256–269.

Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. 2018b. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Accepted for publication in JFP* (2018).

Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. In *POPL*. 637–650.

Jan-Oliver Kaiser, Hoang-Hai Dang, Derek Dreyer, Ori Lahav, and Viktor Vafeiadis. 2017. Strong logic for weak memory: Reasoning about release-acquire consistency in Iris. In *ECOOP (LIPIcs)*, Vol. 74. 17:1–17:29.

Robbert Krebbers. 2015. *The C standard formalized in Coq.* Ph.D. Dissertation. Radboud University.

Robbert Krebbers, Jacques-Henri Jourdan, Ralf Jung, Joseph Tassarotti, Jan-Oliver Kaiser, Amin Timany, Arthur Charguéraud, and Derek Dreyer. 2018. Coq repository for MoSeL. http://iris-project.org/mosel/.

Robbert Krebbers, Ralf Jung, Aleš Bizjak, Jacques-Henri Jourdan, Derek Dreyer, and Lars Birkedal. 2017a. The essence of higher-order concurrent separation logic. In *ESOP (LNCS)*, Vol. 10201. 696–723.

Robbert Krebbers, Amin Timany, and Lars Birkedal. 2017b. Interactive proofs in higher-order concurrent separation logic. In *POPL*. 205–217.

Daniel J. Lehmann, Amir Pnueli, and Jonathan Stavi. 1981. Impartiality, justice and fairness: The ethics of concurrent termination. In *Automata, Languages and Programming*. 264–277.

Andrew McCreight. 2009. Practical tactics for separation logic. In *TPHOLs (LNCS)*, Vol. 5674. 343–358.

Aleksandar Nanevski, Viktor Vafeiadis, and Josh Berdine. 2010. Structuring the verification of heap-manipulating programs. In *POPL*. 261–274.

Peter W. O'Hearn. 2007. Resources, concurrency, and local reasoning. *TCS* 375, 1 (2007), 271–307.

Peter W. O'Hearn and David J. Pym. 1999. The logic of bunched implications. *Bulletin of Symbolic Logic* 5, 2 (1999), 215–244.

Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. 2001. Local reasoning about programs that alter data structures. In *CSL (LNCS)*, Vol. 2142. 1–18.

François Pottier. 2013. Syntactic soundness proof of a type-and-capability system with hidden state. *JFP* 23, 1 (2013), 38–144.

David J. Pym. 2002. *The semantics and proof theory of the logic of bunched implications*. Springer.

David J. Pym, Peter W. O'Hearn, and Hongseok Yang. 2004. Possible worlds and resources: the semantics of BI. *TCS* 315, 1 (2004), 257–305.

John C. Reynolds. 2000. Intuitionistic reasoning about shared mutable data structure. In *Millennial Perspectives in Computer Science*. 303–321.

John C. Reynolds. 2002. Separation logic: A logic for shared mutable data structures. In *LICS*. 55–74.

Ilya Sergey, Aleksandar Nanevski, and Anindya Banerjee. 2015a. Mechanized verification of fine-grained concurrent programs. In *PLDI*. 77–87.

Ilya Sergey, Aleksandar Nanevski, and Anindya Banerjee. 2015b. Specifying and verifying concurrent algorithms with histories and subjectivity. In *ESOP*. 333–358.

Matthieu Sozeau and Nicolas Oury. 2008. First-class type classes. In *TPHOLs (LNCS)*, Vol. 5170. 278–293.

Bas Spitters and Eelis van der Weegen. 2011. Type classes for mathematics in type theory. *MSCS* 21, 4 (2011), 795–825.

David Swasey, Deepak Garg, and Derek Dreyer. 2017. Robust and compositional verification of object capability patterns. *PACMPL* 1, OOPSLA (2017), 89:1–89:26.

Joseph Tassarotti, Ralf Jung, and Robert Harper. 2017. A higher-order logic for concurrent termination-preserving refinement. In *ESOP (LNCS)*, Vol. 10201. 909–936.

Amin Timany, Léo Stefanesco, Morten Krogh-Jespersen, and Lars Birkedal. 2018. A logical relation for monadic encapsulation of state: Proving contextual equivalences in the presence of `runST`. *PACMPL* 2, POPL (2018), 64:1–64:28.

Harvey Tuch, Gerwin Klein, and Michael Norrish. 2007. Types, bytes, and separation logic. In *POPL*. 97–108.

Aaron Turon, Derek Dreyer, and Lars Birkedal. 2013. Unifying refinement and Hoare-style reasoning in a logic for higher-order concurrency. In *ICFP*. 377–390.

Viktor Vafeiadis and Matthew J. Parkinson. 2007. A marriage of rely/guarantee and separation logic. In *CONCUR (LNCS)*, Vol. 4703. 256–271.