The Essence of Higher-Order Concurrent Separation Logic

Robbert Krebbers, Ralf Jung, Aleš Bizjak, Jacques-Henri Jourdan, Derek Dreyer, Lars Birkedal

The Essence of Higher-Order Concurrent Separation Logic

Robbert Krebbers[1], Ralf Jung[2], Aleš Bizjak[3],
Jacques-Henri Jourdan[2], Derek Dreyer[2], and Lars Birkedal[3]

[1] Delft University of Technology, The Netherlands
[2] MPI-SWS, Saarland Informatics Campus, Germany
[3] Aarhus University, Denmark

Abstract. Concurrent separation logics (CSLs) have come of age, and with age they have accumulated a great deal of complexity. Previous work on the Iris logic attempted to reduce the complex logical mechanisms of modern CSLs to two orthogonal concepts: partial commutative monoids (PCMs) and invariants. However, the realization of these concepts in Iris still bakes in several complex mechanisms—such as weakest preconditions and mask-changing view shifts—as primitive notions.

In this paper, we take the Iris story to its (so to speak) logical conclusion, applying the reductionist methodology of Iris to Iris itself. Specifically, we define a small, resourceful *base logic*, which distills the essence of Iris: it comprises only the assertion layer of vanilla separation logic, plus a handful of simple modalities. We then show how the much fancier logical mechanisms of Iris—in particular, its entire program specification layer—can be understood as merely derived forms in our base logic. This approach helps to explain the *meaning* of Iris's program specifications at a much higher level of abstraction than was previously possible. We also show that the step-indexed "later" modality of Iris is an *essential* source of complexity, in that removing it leads to a logical inconsistency. All our results are fully formalized in the Coq proof assistant.

1 Introduction

In his paper *The Next 700 Separation Logics*, Parkinson [26] observed that "separation logic has brought great advances in the world of verification. However, there is a disturbing trend for each new library or concurrency primitive to require a new separation logic." He argued that what is needed is a general logic for concurrent reasoning, into which a variety of useful specifications can be encoded via the abstraction facilities of the logic. "By finding the right core logic," he wrote, "we can concentrate on the difficult problems."

The logic he suggested as a potential candidate for such a core concurrency logic was *deny-guarantee* [12]. Deny-guarantee was indeed groundbreaking in its support for "fictional separation"—the idea that even if threads are concurrently manipulating the same *shared* piece of physical state, one can view them as operating on *logically disjoint* pieces of it and use separation logic to reason modularly about those pieces. It was, however, far from the last word on the subject. Rather,

it spawned a new breed of logics with ever more powerful fictional-separation mechanisms for reasoning modularly about interference [11,16,29,9,30,27]. Several of these also incorporated support for *impredicative invariants* [28,18,17,4], which are needed if one aims to verify code in languages with semantically cyclic features (such as ML or Rust, which support higher-order state).

Although exciting, the progress in this area has come at a cost: as these new separation logics become ever more expressive, each one accumulates increasingly baroque and bespoke proof rules, which are *primitive* in the sense that their soundness is established by direct appeal to the also baroque and bespoke model of the logic. As a result, it is difficult to understand what program specifications in these logics really mean, how they relate to one another, or whether they can be soundly combined in one reasoning framework. In short, we feel, it is high time to renew Parkinson's quest for "the right core logic" of concurrency.

Toward this end, Jung *et al.* [18,17] recently developed **Iris**, a higher-order concurrent separation logic with the goal of simplification and consolidation. The key idea of Iris is that even the fanciest of the interference-control mechanisms in recent concurrency logics can be expressed by a combination of two orthogonal ingredients: *partial commutative monoids* (PCMs) and *invariants*. PCMs enable the user of the logic to roll their own type of fictional (or "logical" or "ghost") state, and invariants serve to tie that fictional state to the underlying physical state of the program. Using just these two mechanisms, Jung *et al.* showed how to take complex primitive proof rules from prior logics and *derive them within* Iris, leading to the slogan: "Monoids and invariants are all you need."

Unfortunately, that slogan does not tell the whole story. Although monoids and invariants do indeed constitute the two main conceptual elements of Iris—and they are arguably "canonical" in their simplicity and universality—the realization of these concepts in Iris involves a number of interacting logical mechanisms, some of which are simple and canonical, others not so much:

- Ownership assertions, $\boxed{a}^{\gamma}$, for logical (ghost) state.
- Named invariant assertions, $\boxed{P}^{\iota}$, asserting that ι is the name of an invariant that enforces that P holds of some piece of the shared state. Invariants in Iris are *impredicative*, which means that $\boxed{P}^{\iota}$ can be used anywhere where normal assertions can be used, *e.g.,* in invariants themselves.
- A necessity modality, $\Box P$, which asserts that P holds *persistently*, as opposed to an assertion describing exclusive ownership of some resource.
- A "later" modality, $\triangleright P$. To support impredicative higher-order quantification and recursively defined assertions, the model of Iris employs the technique of *step-indexing* [2]. This is reflected in the logic in the form of $\triangleright P$, which roughly asserts that P will be true after the next step of computation.
- Invariant masks, $\mathcal{E}$, which are sets of invariant names, ι. Masks are used to track which invariants are enabled (*i.e.,* currently satisfied by some piece of shared state) at a given point in a program proof.
- Mask-changing view shifts, $P \Rrightarrow^{\mathcal{E}_1}_{\mathcal{E}_2} Q$. These describe a kind of logical update operation, asserting (roughly) that, if the invariants in $\mathcal{E}_1$ hold, P can be transformed to Q, after which point the invariants in $\mathcal{E}_2$ hold. These

view shifts are useful for expressing the temporary disabling and re-enabling of invariants within the verification of an atomic step of computation.
– Weakest preconditions, $\mathsf{wp}_{\mathcal{E}}\, e\, \{\Phi\}$, which establish that e is safe to execute assuming the invariants in $\mathcal{E}$ hold, and that if e computes to a value v, then $\Phi(v)$ holds. Hoare triples are encodable in terms of weakest preconditions.

Associated with each of these logical mechanisms are a significant number of primitive proof rules. For certain features, such as the $\Box P$ modality, the rules are mostly standard, and the model is very simple. In contrast, the primitive proof rules for weakest preconditions and view shifts are non-standard, and the model of these features is extremely involved, making the justification of the primitive rules—not to mention the very *meaning* of Iris's Hoare-style program specifications—painfully difficult to understand or explain. Indeed, the previous Iris papers [18,17] have avoided even attempting to present the formal model of program specifications in any detail at all.

In the present paper, we rectify this situation by taking the Iris story to its (so to speak) logical conclusion—that is, by applying the reductionist Iris methodology to Iris itself! Specifically, we present a small, resourceful *base logic*, which distills the essence—the minimal, primitive core—of Iris: it comprises only the assertion layer of vanilla separation logic (*i.e.,* including $P * Q$ but not Hoare triples) extended with $\Box P$, $\triangleright P$, and a simple, novel, monadic *update* modality, $\Rrightarrow P$. Using these basic mechanisms, the fancier mechanisms of mask-changing view shifts and weakest preconditions—and their associated proof rules—*can all be derived within the logic.* And by expressing the fancier mechanisms as derived forms, we can now explain the meaning of Iris's program specifications at a much higher level of abstraction than was previously possible.

In §2, we begin by presenting from first principles the reduced base logic that constitutes the primitive core of our new version of Iris (version 3.0). Then, in §3, we explain step-by-step how to encode weakest preconditions in the Iris 3.0 base logic. Next, in §4, we show how our base logic is sufficient to derive the remaining mechanisms and proof rules of full Iris, including named invariants and mask-changing view shifts.

On the negative side, there is one point of unfortunate complexity that Iris 3.0 inherits from earlier versions without simplification: the aforementioned "later" modality, $\triangleright P$. The Iris rule for accessing an invariant $\boxed{P}^{\iota}$ says that when we gain control of the resource satisfying the invariant, we only learn $\triangleright P$, not P. It has proven very difficult to explain to users of Iris the role of $\triangleright$ here because it boils down to "the model made me do it": the $\triangleright$ reflects a corresponding place in the existing step-indexed model of Iris where the step-index is decreased to ensure a well-founded construction. Moreover, $\triangleright P$ is in general strictly weaker than P, and experience working with Iris has shown that in certain cases this weakness forces the user of the logic into painful workarounds. In §5, we show that in the proof rule for accessing an invariant, the use of $\triangleright$ (or something like it) is in fact *essential*, because if $\triangleright$ is removed from the rule, Iris becomes inconsistent. This provides evidence that $\triangleright$ is a kind of necessary evil.

Finally, in §6, we discuss related work, and in §7, we conclude.

All results in this paper have been formalized in the Coq proof assistant [1].

2 The Iris 3.0 base logic

The goal of this section is to introduce the *Iris 3.0 base logic*, which is the core logic that all of Iris rests on: all its program-logic mechanisms will be defined in terms of just the primitive assertions of our base logic.

The Iris base logic is a higher-order logic with a couple of extensions, most of which are standard. We will discuss each of these extensions in turn. The primitive logical assertions are defined by the following grammar:

$$\begin{aligned} P, Q, R \in \mathsf{Prop} ::= {} & \mathsf{True} \mid \mathsf{False} \mid t = u \mid P \wedge Q \mid P \vee Q \mid P \Rightarrow Q \mid \forall x.\, P \mid \exists x.\, P \\ & \mid P * Q \mid P \mathrel{-\!\!*} Q \mid \mathsf{Own}(a) \mid \mathcal{V}(a) \mid \Box P \mid \Rrightarrow P \mid \mu\, x.\, P \mid \triangleright P \end{aligned}$$

Since the logic is higher-order, the full grammar of (multi-sorted) terms also involves the usual connectives of the simply-typed lambda calculus. This is common practice; the full details are spelled out in the technical appendix [1].

The rules for the logical entailment[4] $P \vdash Q$ are displayed in Figure 1. Note that $P \dashv\vdash Q$ is shorthand for having both $P \vdash Q$ and $Q \vdash P$.

We omit the ordinary rules for intuitionistic higher-order logic with equality, which are standard and displayed in the appendix [1]. The remaining connectives and proof principles fall into two broad categories: those dealing with ownership of resources (§2.1-§2.5) and those related to step-indexing (§2.6-§2.7).

2.1 Separation logic

The connectives $*$ and $\mathrel{-\!\!*}$ of bunched implications [25] make our base logic a *separation logic*: they let us reason about *ownership of resources*. The key point is that $P * Q$ describes ownership of a resource that can be *separated* into two disjoint pieces, one satisfying P and one satisfying Q. This is in contrast to $P \wedge Q$, which describes ownership of a resource satisfying both P and Q.

For example, consider the resources owned by different threads in a concurrent program. Because these threads operate concurrently, it is crucial that their ownership is *disjoint*. As a consequence, separating conjunction is the natural operator to combine the ownership of concurrent threads.

Together with separating conjunction, we have a second form of implication: the *magic wand* $P \mathrel{-\!\!*} Q$. It describes ownership of "Q minus P", *i.e.,* it describes resources such that, if you (disjointly) add resources satisfying P, you obtain resources satisfying Q.

2.2 Resource algebras

The purpose of the $\mathsf{Own}(a)$ connective is to assert ownership of the resource a. Before we go on introducing this connective, we need to answer the following question: *what is a resource*?

[4] The full judgment is of the shape $\Gamma \mid P \vdash Q$, where Γ assigns types to free variables. However, since Γ only plays a role in the rules for quantifiers, we omit it.

Laws of (affine) bunched implications.

$$\mathsf{True} * P \dashv\vdash P \qquad P * Q \vdash Q * P \qquad (P * Q) * R \vdash P * (Q * R)$$

$$\text{*-MONO}\ \frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * P_2 \vdash Q_1 * Q_2} \qquad \text{-*-INTRO}\ \frac{P * Q \vdash R}{P \vdash Q \mathrel{-\!\!*} R} \qquad \text{-*-ELIM}\ \frac{P \vdash Q \mathrel{-\!\!*} R}{P * Q \vdash R}$$

Laws for resources and validity.

$$\text{OWN-OP}\ \mathsf{Own}(a) * \mathsf{Own}(b) \dashv\vdash \mathsf{Own}(a \cdot b) \qquad \text{OWN-UNIT}\ \mathsf{True} \vdash \mathsf{Own}(\varepsilon) \qquad \text{OWN-CORE}\ \mathsf{Own}(a) \vdash \Box\,\mathsf{Own}(|a|)$$

$$\text{OWN-VALID}\ \mathsf{Own}(a) \vdash \mathcal{V}(a) \qquad \text{VALID-OP}\ \mathcal{V}(a \cdot b) \vdash \mathcal{V}(a) \qquad \text{VALID-ALWAYS}\ \mathcal{V}(a) \vdash \Box\,\mathcal{V}(a)$$

Laws for the basic update modality.

$$\text{UPD-MONO}\ \frac{P \vdash Q}{\Rrightarrow P \vdash \Rrightarrow Q} \qquad \text{UPD-INTRO}\ P \vdash \Rrightarrow P \qquad \text{UPD-TRANS}\ \Rrightarrow\Rrightarrow P \vdash \Rrightarrow P \qquad \text{UPD-FRAME}\ Q * \Rrightarrow P \vdash \Rrightarrow(Q * P)$$

$$\text{UPD-UPDATE}\ \frac{a \rightsquigarrow B}{\mathsf{Own}(a) \vdash \Rrightarrow \exists b \in B.\, \mathsf{Own}(b)}$$

Laws for the always modality.

$$\Box\text{-MONO}\ \frac{P \vdash Q}{\Box P \vdash \Box Q} \qquad \Box\text{-ELIM}\ \Box P \vdash P \qquad \mathsf{True} \vdash \Box\,\mathsf{True} \qquad \Box(P \wedge Q) \vdash \Box(P * Q) \qquad \Box P \wedge Q \vdash \Box P * Q$$

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

Laws for the later modality.

$$\triangleright\text{-MONO}\ \frac{P \vdash Q}{\triangleright P \vdash \triangleright Q} \qquad \text{Löb}\ (\triangleright P \Rightarrow P) \vdash P \qquad \forall x.\, \triangleright P \vdash \triangleright \forall x.\, P \qquad \triangleright \exists x.\, P \vdash \triangleright\,\mathsf{False} \vee \exists x.\, \triangleright P$$

$$\triangleright(P * Q) \dashv\vdash \triangleright P * \triangleright Q \qquad \Box \triangleright P \dashv\vdash \triangleright \Box P$$

Laws for timeless assertions.

$$\triangleright\text{-TIMELESS}\ \triangleright P \vdash \triangleright\,\mathsf{False} \vee (\triangleright\,\mathsf{False} \Rightarrow P) \qquad \triangleright\text{-OWN}\ \triangleright\,\mathsf{Own}(a) \vdash \exists b.\, \mathsf{Own}(b) \wedge \triangleright(a = b)$$

Fig. 1. Proof rules of the Iris 3.0 base logic.

The Iris base logic does not answer this question by fixing a particular set of resources. Instead, the set of resources is kept general, and it is up to the user of the logic to make a suitable choice. All the logic demands is that the set of resources forms a *unital resource algebra* (uRA), as defined in Figure 2.

Resource algebras are similar to *partial commutative monoids* (PCMs), which are often used to describe ownership in concurrent separation logics because:

A *resource algebra* (RA) is a tuple $(M, \mathcal{V} \subseteq M, |-| : M \to M^?, (\cdot) : M \times M \to M)$ satisfying:

$$\forall a, b, c.\, (a \cdot b) \cdot c = a \cdot (b \cdot c) \qquad \forall a, b.\, a \cdot b = b \cdot a$$
$$\forall a, b.\, (a \cdot b) \in \mathcal{V} \Rightarrow a \in \mathcal{V} \qquad \forall a.\, |a| \in M \Rightarrow |a| \cdot a = a$$
$$\forall a.\, |a| \in M \Rightarrow ||a|| = |a| \qquad \forall a, b.\, |a| \in M \land a \preccurlyeq b \Rightarrow |b| \in M \land |a| \preccurlyeq |b|$$

where $M^? \triangleq M \uplus \{\bot\}$ with $a^? \cdot \bot \triangleq \bot \cdot a^? \triangleq a^?$

$$a \preccurlyeq b \triangleq \exists c \in M.\, b = a \cdot c$$
$$a \rightsquigarrow B \triangleq \forall c^? \in M^?.\, a \cdot c^? \in \mathcal{V} \Rightarrow \exists b \in B.\, b \cdot c^? \in \mathcal{V}$$
$$a \rightsquigarrow b \triangleq a \rightsquigarrow \{b\}$$

A *unital resource algebra* (uRA) is a resource algebra M with an element ε satisfying:

$$\varepsilon \in \mathcal{V} \qquad \forall a \in M.\, \varepsilon \cdot a = a \qquad |\varepsilon| = \varepsilon$$

Fig. 2. Resource algebras.

- Ownership of different threads can be *composed* using the $\cdot$ operator.
- Composition of ownership is *associative and commutative*, reflecting the associative and commutative semantics of parallel composition.
- Combinations of ownership that do not make sense are ruled out by *partiality*, *e.g.,* multiple threads claiming to have ownership of an exclusive resource.

However, there are some differences between RAs and PCMs:

1. Instead of partiality, RAs use *validity* to rule out invalid combinations of ownership. Specifically, there is a subset $\mathcal{V}$ of *valid* elements. As shown previously [17], this take on partiality is necessary when defining *higher-order* ghost state, which we will need for modeling invariants in §4.3.
2. Instead of having one "unit" that acts as the identity for *every* element, RAs have a partial function $|-|$ assigning the *(duplicable) core* $|a|$ to each element a. The core of an RA is a strict generalization of the unit of a PCM: the core can be *different* for different elements, and since the core is *partial*, there can actually be elements of the RA for which there is no identity element.

Although the Iris base logic is parameterized by a uRA (that is, an RA with a single, global unit), we do not demand that every RA have a unit because we typically compose RAs from smaller parts. Requiring all of these "intermediate" RAs to be unital would render many of our compositions impossible [17].

Let us now give some examples of RAs; more appear in §3.3 and §4.2.

Exclusive. Given a set X, the task of the *exclusive RA* $\textsc{Ex}(X)$ is to make sure that one party *exclusively* owns a value $x \in X$. (We are using a datatype-like notation to declare the possible elements of $\textsc{Ex}(X)$.)

$$\textsc{Ex}(X) \triangleq \mathsf{ex}(x : X) \mid \lightning \qquad \mathcal{V} \triangleq \{\mathsf{ex}(x) \mid x \in X\} \qquad |\mathsf{ex}(x)| \triangleq \bot$$

Composition is always undefined (using the invalid dummy element $\lightning$) to ensure that ownership is *exclusive*, *i.e.,* exactly one party has full control over the resource. This RA does not have a unit.

Finite partial function. Given a set of keys K and an RA M, the *finite partial function uRA* $K \xrightarrow{\text{fin}} M$ is defined by lifting the core and the composition operator pointwise, and by defining validity as the conjunction of pointwise validities. The unit ε is defined to be the empty partial function $\emptyset$.

2.3 Resource ownership

Having completed the discussion of RAs, we now come back to the base logic and its connective $\mathsf{Own}(a)$, which describes ownership of the RA element a. It forms the "primitive" form of ownership in our logic, which can then be composed into more interesting assertions using the previously described connectives. The most important fact about ownership is that separating conjunction "reflects" the composition operator of RAs into the logic (OWN-OP).

Besides the $\mathsf{Own}(a)$ connective, we have the primitive connective $\mathcal{V}(a)$, which reflects validity of RA elements into the logic. Note that ownership is connected to validity: the rule OWN-VALID says that only valid elements can be owned.

2.4 Resource updates

So far, resources have been *static*: the logic provides assertions to reason about resources you own, the consequences of that ownership, and how ownership can be disjointly separated. The *(basic) update modality* $\Rightarrow P$, however, lets you talk about what you *could own* after performing an update to what you do own.

Updates to resources are called *frame-preserving updates* and can be performed using the rule UPD-UPDATE. We can perform a frame-preserving update $a \rightsquigarrow B$ if for any resource (called a *frame*) a_f such that $a \cdot a_\mathrm{f} \in \mathcal{V}$, there exists a resource $b \in B$ such that $b \cdot a_\mathrm{f} \in \mathcal{V}$. If we think of those frames as being the resources owned by other threads, then a frame-preserving update is guaranteed not to invalidate the resources of concurrently-running threads. By doing only frame-preserving updates, we know we will never "step on anybody else's toes".

Before discussing how frame-preserving updates are reflected into the logic, we give some examples of frame-preserving updates. Since ownership in the exclusive RA is exclusive, there is nobody whose assumptions could be invalidated by changing the value of the resource. To that end, we have $\mathsf{ex}(x) \rightsquigarrow \mathsf{ex}(y)$ for any x and y. The updates for the finite partial functions $K \xrightarrow{\text{fin}} M$ are as follows:

$$\text{FPFN-UPDATE}\ \frac{a \rightsquigarrow_M B}{f[i := a] \rightsquigarrow \{f[i := b] \mid b \in B\}} \qquad \text{FPFN-ALLOC}\ \frac{a \in \mathcal{V} \qquad K \text{ infinite}}{\emptyset \rightsquigarrow \{[i := a] \mid i \in K\}}$$

The first rule witnesses pointwise lifting of updates on M. The second rule is more interesting: it allows us to *allocate* a fresh slot in the finite partial function.

This is always possible because only finitely many indices $i \in K$ will be used at any given point in time.

The update modality reflects frame-preserving updates into the logic, in the sense that $\Rightarrow\!\!\!\mid P$ asserts ownership of resources *that can be updated to* resources satisfying P. The rule UPD-UPDATE witnesses this relationship, while the remaining proof rules essentially say that $\Rightarrow\!\!\!\mid$ is a *strong monad* with respect to separating conjunction [19,20].

This gives rise to an alternative interpretation of the basic update modality: we can think of $\Rightarrow\!\!\!\mid P$ as a *thunk* that captures some resources in its environment and that, when executed, will "return" resources satisfying P. The various proof rules then let us perform additional reasoning on the result of the thunk (UPD-MONO), create a thunk that does nothing (UPD-INTRO), compose two thunks into one (UPD-TRANS), and add resources to those captured by the thunk (UPD-FRAME).

2.5 The always modality

The intuition for the *always modality* $\Box P$ is that P holds *without asserting any exclusive ownership*. This is useful because an assumption $\Box P$ can be used arbitrarily often, *i.e.,* it cannot be "used up". In particular, while $P \mathrel{-\!\!*} Q$ is a "linear implication" and can only be applied once, $\Box(P \mathrel{-\!\!*} Q)$ can be applied arbitrarily often. We use this in the encoding of Hoare triples in §3.2.

We call an assertion P *persistent* if proofs of P can never assert exclusive ownership, which formally means it enjoys $P \vdash \Box P$. As soon as either P or Q is persistent, their separating conjunction $(P * Q)$ and normal conjunction $(P \wedge Q)$ coincide, thus enabling one to use "normal" intuitionistic reasoning.

Under which circumstances is $\mathsf{Own}(a)$ persistent? RAs provide a flexible answer to this: the core $|a|$ defines the *duplicable* part of a, and hence $\mathsf{Own}(|a|)$ does not assert any exclusive ownership, which is reflected into the logic using the rule OWN-CORE. In §4.2, we will consider an example of an RA with a non-trivial core, and we will make use of the fact that $\mathsf{Own}(|a|)$ is persistent.

2.6 The later modality and guarded fixed-points

Although RAs provide a powerful way to instantiate our logic with the user's custom type of resources, they have an inherent limitation: the user-chosen RA must be defined *a priori*. But what if the user wants to define their resources *in terms of* the assertions of the logic? In prior work [17], we called this phenomenon *higher-order ghost state*, and showed how to incorporate it into the Iris 2.0 logic. Iris 3.0 inherits higher-order ghost state from Iris 2.0 without change.

The challenge of supporting higher-order ghost state is that the user-chosen RA depends on the type of propositions of our logic, which in turn depends on the user-chosen RA. In Iris 2.0, we showed how to cut this circularity using a novel algebraic structure called a *CMRA* ("camera"), which synthesizes the features of an RA together with a *step-indexed* structure [2]. Since a proper understanding of CMRAs is not needed in order to appreciate the contribution

of the present paper, we refer the reader to the Iris 2.0 paper [17] for details, and instead focus briefly here on how the presence of higher-order ghost state affects our base logic. (We will see a concrete instance of higher-order ghost state in §4.2, where we use it to encode impredicative invariants.)

The step-indexing aspect of CMRAs is internalized into the logic by adding a new modality: the *later modality*, $\triangleright P$ [23,3]. Intuitively, $\triangleright P$ asserts that P holds "at the next step-index" (or "one step later"). In the definition of weakest preconditions in §3.3, we connect $\triangleright$ to computation steps, allowing one to think of $\triangleright P$ as saying that P holds at the next step of computation.

Beyond higher-order ghost state, step-indexing allows us to include a fixed-point operator $\mu\, x.\, P$ into the logic, which can be used to define recursive predicates *without any restriction on the variance* of the recursive occurrences of x in P. Instead, all recursive occurrences must be *guarded*: they have to appear below a later modality $\triangleright$. In §3, we will show how guarded recursion is used for defining weakest preconditions. Moreover, as shown in [28], guarded recursion is useful to define specifications for higher-order concurrent data structures.

A crucial proof rule for $\triangleright$ is Löb, which facilitates proving properties about fixed-points: we can essentially assume that the recursive occurrences are already proven correct (as they are under a later). Note that many of the usual rules for later, such as introduction ($P \vdash \triangleright P$) and commutativity with other operators ($\triangleright(P \land Q) \dashv\vdash \triangleright P \land \triangleright Q$) are derivable from the rules in Figure 1.

2.7 Timeless assertions

There are some occasions where we inevitably end up with hypotheses below a later. An example is the Iris rule for accessing invariants (WP-INV in §4). Although one can always introduce a later, one cannot just eliminate a later, so the later may make certain reasoning steps impossible. However, as we will prove in §5, it is crucial for logical consistency that the later is present in WP-INV.

Still, for many assertions, their semantics is independent of step-indexing, so adding a $\triangleright$ in front of them does not really "change" anything. When accessing an invariant containing such an assertion, we thus do not want the later to be in the way. Ideally, for such assertions, we would like to have $\triangleright P \vdash P$. However, that does not work: indeed, *at step-index* 0, $\triangleright P$ trivially holds and, consequently, does not imply P. Instead, we say that an assertion P is *timeless* when $\triangleright P \vdash \diamond P$, where the modality $\diamond$ is defined by $\diamond P \triangleq P \lor \triangleright \mathsf{False}$. We call this new $\diamond$ modality "except 0": it states that the given assertion holds at all step-indices greater than 0. Under this modality, we can strip away a later from a timeless assertion, *i.e.,* given a timeless P, to prove $\triangleright P \vdash \diamond Q$, it is sufficient to prove $P \vdash \diamond Q$.

Using the rules for timeless assertions in Figure 1, we can prove that some frequently occurring assertions are timeless. In particular, if a CMRA is *discrete—i.e.,* if it degenerates to a plain RA that ignores the step-indexing structure, as is the case for many types of resources—then equality, ownership and validity of such resources are timeless. Furthermore, most of the connectives of our logic (not including $\triangleright$) preserve timelessness.

2.8 Consistency

Logical consistency is usually stated as $\mathsf{True} \not\vdash \mathsf{False}$, *i.e.,* from a closed context one cannot prove a contradiction. However, when building a program logic within our base logic, we wish to prove that the postconditions of our Hoare triples actually represent program behavior (§4.6), so we need a stronger statement:

Theorem 2.1 (Soundness of first-order interpretation). *Given a first-order proposition ϕ (not involving ownership, higher-order quantification, nor any of the modalities) and $\mathsf{True} \vdash (\Rrightarrow\triangleright)^n\,\phi$, then the "standard" (meta-logic) interpretation of ϕ holds. Here, $(\Rrightarrow\triangleright)^n$ is notation for nesting $\Rrightarrow\triangleright$ n times.*

The proposition ϕ should be a first-order predicate to ensure it can be used both inside our logic and at the meta-level. Furthermore, the theorem makes sure that even when reasoning below any combination of modalities, we cannot prove a contradiction. Consistency, *i.e.,* $\mathsf{True} \not\vdash \mathsf{False}$, is a trivial consequence of this theorem: just pick $\phi = \mathsf{False}$ and $n = 0$.

Theorem 2.1 is proven by defining a suitable semantic domain of assertions, interpreting all connectives into that domain, and proving soundness of all proof rules. For further details, we refer the reader to [17,1].

3 Weakest preconditions

This section shows how to encode a program logic in the Iris base logic. Usually, program logics are centered around Hoare triples, but instead of directly defining Hoare triples in the base logic, we first define the notion of a *weakest precondition*. There are two reasons for defining Hoare triples in terms of the weakest precondition connective: First, weakest preconditions are more primitive and, as such, more natural to encode. Second, weakest preconditions are more convenient for performing interactive proofs with Iris [21].

We will first give some intuition about weakest preconditions and how to work with them. After that, we present the encoding of weakest preconditions in three stages, gradually adding support for reasoning about state and concurrency. For simplicity, we use a concrete programming language in this section. The version including all features of Iris for an arbitrary language is given in §4.

3.1 Programming language

For the purpose of this example, we use a call-by-value λ-calculus with references and fork. The syntax and semantics are given in Figure 3.

Head-reduction $(e, \sigma) \to_{\mathsf{h}} (e', \sigma', \vec{e}_f)$ is defined on pairs (e, σ) consisting of an expression e and a shared heap σ (a finite partial map from locations to values). Moreover, $\vec{e}_f$ is a list of forked off expressions, which is used to define the semantics of `fork` $\{e\}$. The head-reduction is lifted to a per-thread reduction $(e, \sigma) \to (e', \sigma', \vec{e}_f)$ using evaluation contexts. We define an expression e to be reducible in a shared heap σ, and we note $\mathrm{red}(e, \sigma)$, if it can make a thread-local step. The thread-pool reduction $(T, \sigma) \to_{\mathsf{tp}} (T', \sigma')$ is an interleaving semantics where the *thread-pool* T denotes the existing threads as a list of expressions.

Syntax:

$$v \in \mathit{Val} ::= () \mid \ell \mid \lambda x.e$$
$$e \in \mathit{Expr} ::= v \mid x \mid e_1(e_2) \mid \texttt{fork}\ \{e\} \mid \texttt{ref}(e) \mid !\,e \mid e_1 \leftarrow e_2$$
$$K \in \mathit{Ctx} ::= \bullet \mid K(e) \mid v(K) \mid \texttt{ref}(K) \mid !\,K \mid K \leftarrow e \mid v \leftarrow K$$

Head reduction:

$$\begin{aligned}
((\lambda x.e)v, \sigma) &\to_{\mathsf{h}} (e[v/x], \sigma, \epsilon) \\
(!\,\ell, \sigma) &\to_{\mathsf{h}} (v, \sigma, \epsilon) && \text{if } \sigma(\ell) = v \\
(\ell \leftarrow w, \sigma) &\to_{\mathsf{h}} ((), \sigma[\ell := w], \epsilon) && \text{if } \sigma(\ell) = v \\
(\texttt{ref}(v), \sigma) &\to_{\mathsf{h}} (\ell, \sigma[\ell := v], \epsilon) && \text{if } \sigma(\ell) = \bot \\
(\texttt{fork}\ \{e\}, \sigma) &\to_{\mathsf{h}} ((), \sigma, e)
\end{aligned}$$

Thread-local reduction:

$$\frac{(e, \sigma) \to_{\mathsf{h}} (e', \sigma', \vec{e}_f)}{(K[\,e\,], \sigma) \to (K[\,e'\,], \sigma', \vec{e}_f)}$$

Thread-pool semantics:

$$\frac{(e, \sigma) \to (e', \sigma', \vec{e}_f)}{(T_1; e; T_2, \sigma) \to_{\mathsf{tp}} (T_1; e'; T_2; \vec{e}_f, \sigma')}$$

Fig. 3. Lambda calculus with references and fork.

3.2 Proof rules

Before coming to the actual contribution of this section—which is the encoding of weakest preconditions using our base logic in §3.3—we give some idea of how to reason using weakest preconditions by discussing its proof rules. These proof rules are inspired by [15], but presented in weakest precondition style.

Given a predicate $\Phi : \mathit{Val} \to \mathsf{Prop}$, called the postcondition, the connective $\mathsf{wp}\, e\, \{\Phi\}$ gives the *weakest precondition* under which all executions of e are *safe*, and all return values v of e satisfy the postcondition $\Phi(v)$. For an execution to be safe, we demand that it *does not get stuck*, which in the case of our language means the program must never access invalid locations.

Figure 4 shows some rules of the $\mathsf{wp}\, e\, \{\Phi\}$ connective. To reason about state, we use the well-known *points-to assertion* $\ell \mapsto v$, which states that we *exclusively own* the location ℓ, and that it currently stores value v. As part of defining weakest preconditions, we will also have to define the points-to assertion.

As usual in a weakest precondition style system [10], the postcondition of the conclusion of each rule involves an arbitrary predicate Φ. For example, imagine we want to prove $\ell \mapsto v * P \vdash \mathsf{wp}\, (\ell \leftarrow w)\, \{\Phi\}$. The rule WP-STORE tells us what we have to show about Φ for this to hold:

$$\dfrac{\dfrac{\dfrac{P * \ell \mapsto w \vdash \Phi()}{P \vdash \ell \mapsto w \mathrel{-\!\!*} \Phi()}\ {\scriptstyle \mathrel{-\!\!*}\text{-INTRO}}}{\ell \mapsto v * P \vdash \ell \mapsto v * \triangleright(\ell \mapsto w \mathrel{-\!\!*} \Phi())}\ {\scriptstyle *\text{-MONO},\ \triangleright\text{-INTRO}}}{\ell \mapsto v * P \vdash \mathsf{wp}\, (\ell \leftarrow w)\, \{\Phi\}}\ {\scriptstyle \text{WP-STORE}}$$

Here, we use $*$-MONO to show that we own the location ℓ – this should not be surprising; in a separation logic, we have to demonstrate ownership of a location to access it. Furthermore, using our remaining resources P we have to prove $\ell \mapsto w \mathrel{-\!\!*} \Phi()$. It does not matter what Φ says for values other than $()$, which corresponds to the fact that the store expression terminates with $()$.

Notice the end-to-end effect of applying this little derivation: we had to show that we own $\ell \mapsto v$, and it got replaced in our context with $\ell \mapsto w$. However, this

$$\textsc{wp-mono}\ \frac{\forall v.\, \Phi(v) \vdash \Psi(v)}{\mathsf{wp}\, e\, \{\Phi\} \vdash \mathsf{wp}\, e\, \{\Psi\}} \qquad \textsc{wp-frame}\ \frac{P * \mathsf{wp}\, e\, \{\Phi\}}{\mathsf{wp}\, e\, \{P * \Phi\}} \qquad \textsc{wp-val}\ \frac{\Phi(v)}{\mathsf{wp}\, v\, \{\Phi\}} \qquad \textsc{wp-bind}\ \frac{\mathsf{wp}\, e\, \{v.\, \mathsf{wp}\, K[\, v\,]\, \{\Phi\}\}}{\mathsf{wp}\, K[\, e\,]\, \{\Phi\}}$$

$$\textsc{wp-fork}\ \frac{\triangleright \Phi() * \triangleright\, \mathsf{wp}\, e\, \{v.\, \mathsf{True}\}}{\mathsf{wp}\, \mathtt{fork}\, \{e\}\, \{\Phi\}} \qquad \textsc{wp-}\lambda\ \frac{\triangleright\, \mathsf{wp}\, e[v/x]\, \{\Phi\}}{\mathsf{wp}\, (\lambda x.e)v\, \{\Phi\}} \qquad \textsc{wp-load}\ \frac{\ell \mapsto v * \triangleright(\ell \mapsto v \mathrel{-\!\!*} \Phi(v))}{\mathsf{wp}\, !\ell\, \{\Phi\}}$$

$$\textsc{wp-store}\ \frac{\ell \mapsto v * \triangleright(\ell \mapsto w \mathrel{-\!\!*} \Phi())}{\mathsf{wp}\, (\ell \leftarrow w)\, \{\Phi\}} \qquad \textsc{wp-alloc}\ \frac{\triangleright(\forall \ell.\, \ell \mapsto v \mathrel{-\!\!*} \Phi(\ell))}{\mathsf{wp}\, \mathtt{ref}(v)\, \{\Phi\}}$$

Fig. 4. Rules for weakest preconditions.

was all expressed in the *premise* of WP-STORE (and similarly for the other rules), with the conclusion applying to an *arbitrary* postcondition Φ. We could have equivalently written the rule as $\ell \mapsto v \mathrel{-\!\!*} \mathsf{wp}\, (\ell \leftarrow w)\, \{\ell \mapsto w\}$, but applying rules in such a style requires using the rules of framing (WP-FRAME) and monotonicity (WP-MONO) for every instruction. We thus prefer the style of rules in Figure 4.

Hoare triples. Traditional Hoare triples can be defined in terms of weakest preconditions as $\{P\}\, e\, \{\Phi\} \triangleq \Box(P \mathrel{-\!\!*} \mathsf{wp}\, e\, \{\Phi\})$. The $\Box$ modality ensures that the triple asserts no exclusive ownership, and as such, can be used multiple times.

3.3 Definition of weakest preconditions

We now discuss how to *define* weakest preconditions using the Iris base logic, proceeding in three stages of increasing complexity.

First stage. To get started, let us assume the program we want to verify makes no use of fork or shared heap access. The idea of $\mathsf{wp}\, e\, \{\Phi\}$ is to ensure that given any reduction $(e, \sigma) \to \cdots \to (e_n, \sigma_n)$, either (e_n, σ_n) is reducible, or the program terminated, *i.e.,* e_n is a value v for which we have $\Phi(v)$. The natural candidate for encoding this is using the fixed-point operator $\mu\, x.\, P$ of our logic. Consider the following:

$$\begin{aligned} \mathsf{wp}\, e\, \{\Phi\} \triangleq\ & (e \in \mathit{Val} \wedge \Phi(e)) && \textit{(return value)} \\ & \vee \big(e \notin \mathit{Val} \wedge \forall \sigma.\ \mathrm{red}(e, \sigma) && \textit{(safety)} \\ & \quad \wedge \triangleright\, (\forall e_2, \sigma_2.\, (e, \sigma) \to (e_2, \sigma_2, \epsilon) \mathrel{-\!\!*} \mathsf{wp}\, e_2\, \{\Phi\}) && \textit{(preservation)} \quad \big) \end{aligned}$$

Weakest precondition is defined by case-distinction: either the program has already terminated (e is a value), in which case the postcondition should hold. Alternatively, the program is *not* a value, in which case there are two requirements. First, for any possible heap σ, the program should be reducible (called

program safety). Second, *if* the program makes a step, then the weakest precondition of the reduced program e_2 must hold (called *preservation*).

Note that the recursive occurrence $\mathsf{wp}\, e_2\, \{\Phi\}$ appears under a $\triangleright$-modality, so the above can indeed be defined using the fixed-point operator μ. In some sense, this "ties" the steps of the program to the step-indices implicit in the logic, by adding another $\triangleright$ for every program step.

So, how useful is this definition? The rules WP-VAL and WP-λ are almost trivial, and using LÖB induction we can prove WP-MONO, WP-FRAME and WP-BIND. We can thus reason about programs that do not fork or make use of the heap.

But unfortunately, this definition cannot be used to verify programs involving heap accesses: the states σ and σ_2 are universally quantified and not related to anything. The program must always be able to proceed under *any* heap, so we cannot possibly prove the rules of the load, store and allocation constructs.

The usual way to proceed in constructing a separation logic is to define the pre- and post-conditions as *predicates* over states, but that is not the direction we take. After all, our base logic *already* has a notion of "resources that can be updated"—*i.e.,* a notion of state—built in to its model of assertions. Of course we want to make use of this power in building our program logic.

Second stage: Adding state. We now consider programs that access the shared heap but still do not fork. To use the resources provided by the Iris base logic, we have to start by thinking about the right RA. An obvious candidate would be to use $\mathit{Loc} \xrightarrow{\text{fin}} \textsc{Ex}(\mathit{Val})$ (which is isomorphic to finite partial functions with composition being disjoint union) and define $\ell \mapsto v$ as $\mathsf{Own}([\ell := \mathsf{ex}(v)])$. However, that leaves us with a problem: how do we tie those resources to the actual heap that the program executes on? We have to make sure that from *owning* $\ell \mapsto v$, we can actually deduce that ℓ is allocated in σ.

To this end, we will actually have *two* heaps in our resources, *both* elements of $\mathit{Loc} \xrightarrow{\text{fin}} \textsc{Ex}(\mathit{Val})$. The *authoritative heap* $\bullet\,\sigma$ is managed by the weakest precondition, and tied to the physical state occurring in the program reduction. There will only ever be one authoritative heap resource, *i.e.,* we want $\bullet\,\sigma \cdot \bullet\,\sigma'$ to be invalid. At the same time, the *heap fragments* $\circ\,\sigma$ will be owned by the program itself and used to give meaning to $\ell \mapsto v$. These fragments can be composed the usual way ($\circ\,\sigma \cdot \circ\,\sigma' = \circ\,(\sigma \uplus \sigma')$). Finally, we need to tie these two pieces together, making sure that the fragments are always a "part" of the authoritative state: if $\bullet\,\sigma \cdot \circ\,\sigma'$ is valid, then $\sigma' \preccurlyeq \sigma$ should hold.

This is called the *authoritative RA*, $\textsc{Auth}(\mathit{Loc} \xrightarrow{\text{fin}} \textsc{Ex}(\mathit{Val}))$ [18]. Before we explain how to define the authoritative RA, let us see why it is useful in the definition of weakest preconditions. The new definition is (changes are in red):

$$
\begin{aligned}
\mathsf{wp}\, e\, \{\Phi\} \triangleq\ & (e \in \mathit{Val} \wedge \Rrightarrow \Phi(e)) \\
& \vee \big(e \notin \mathit{Val} \wedge \forall \sigma.\, \mathsf{Own}(\bullet\,\sigma) \mathrel{-\!\!*} \Rrightarrow \mathrm{red}(e, \sigma) \\
& \quad \wedge \triangleright\, (\forall e_2, \sigma_2.\, (e, \sigma) \to (e_2, \sigma_2, \epsilon) \mathrel{-\!\!*} \Rrightarrow \mathsf{Own}(\bullet\,\sigma_2) * \mathsf{wp}\, e_2\, \{\Phi\})\big) \\
\ell \mapsto v \triangleq\ & \mathsf{Own}(\circ\,[\ell := v])
\end{aligned}
$$

The difference from the first definition is that the second disjunct (the one covering the case of a program that can still reduce) requires proving safety and preservation under the assumption that the authoritative heap $\bullet\,\sigma$ *matches the physical one*. Moreover, when the program makes a step to some new state σ_2, the proof must be able to *produce* a matching authoritative heap. Finally, the basic update modality permits the proof to perform frame-preserving updates.

To see why this is useful, consider proving WP-LOAD, the weakest precondition of $!\,\ell$. After picking the right disjunct and introducing all assumptions, we can combine the assumptions made by the rule, $\ell \mapsto v$, with the assumptions provided by the definition of weakest preconditions to obtain $\mathsf{Own}(\bullet\,\sigma \cdot \circ\,[\ell := v])$. By OWN-VALID, we learn that this RA element is valid, which (as discussed above) implies $[\ell := v] \preccurlyeq \sigma$, so $\sigma(\ell) = v$. In other words, because the RA ties the authoritative heap and the heap fragments together, and because weakest precondition ties the authoritative heap and the physical heap used in program reduction together, we can make a connection between $\ell \mapsto v$ and the physical heap.

Completing the proof of safety and progress now is straightforward. Since all possible reductions of $!\,\ell$ do not change the heap, we can produce the authoritative heap $\bullet\,\sigma_2$ by just "forwarding" the one we got earlier in the proof. In this case, we did not even make use of the fact that we are allowed to perform frame-preserving updates. This is, however, necessary to prove weakest preconditions of operations that actually *change* the state (like allocation or storing), because in these cases, the authoritative heap needs to be changed likewise.

Authoritative RA. To complete the definition, we need to define the *authoritative RA* [18]. We can do so in general (*i.e.,* the definition is not specific to heaps), so assume we are given some uRA M and let:

$$\begin{aligned}
\textsc{Auth}(M) &\triangleq \textsc{Ex}(M)^? \times M \\
\mathcal{V} &\triangleq \{(\bot, b) \mid b \in \mathcal{V}\} \cup \left\{ (\mathsf{ex}(a), b) \;\middle|\; a \in \mathcal{V} \wedge b \preccurlyeq a) \right\} \\
(x_1, b_1) \cdot (x_2, b_2) &\triangleq (x_1 \cdot x_2, b_2 \cdot b_2) \\
|(x, b)| &\triangleq (\bot, |b|)
\end{aligned}$$

With $a \in M$, we write $\bullet\, a$ for $(\mathsf{ex}(a), \varepsilon)$ to denote *authoritative* ownership of a and $\circ\, a$ for $(\bot, a)$ to denote *fragmentary* ownership of a.

It can be easily verified that this RA has the three key properties discussed above: ownership of $\bullet\, a$ is exclusive, ownership of $\circ\, a$ composes like that of a, and the two are tied together in the sense that validity of $\bullet\, a \cdot \circ\, b$ implies $b \preccurlyeq a$. Beyond this, it turns out that we can show the following frame-preserving updates that are needed for WP-STORE and WP-ALLOC:

$$\begin{aligned}
\bullet\,\sigma \cdot \circ\,[\ell := v] &\rightsquigarrow \bullet\,\sigma[\ell := w] \cdot \circ\,[\ell := w] \\
\bullet\,\sigma &\rightsquigarrow \bullet\,\sigma[\ell := w] \cdot \circ\,[\ell := w] \qquad \text{if } \ell \notin \mathrm{dom}(\sigma)
\end{aligned}$$

Third stage: Adding fork. Our previous definition of $\mathsf{wp}\, e\, \{\Phi\}$ only talked about reductions $(e, \sigma) \to (e_2, \sigma_2, \epsilon)$ which do *not* fork off threads, and hence

one could not prove WP-FORK. This new definition lifts this limitation:

$$\begin{aligned}\mathsf{wp}\, e\, \{\Phi\} \triangleq\ & (e \in \mathit{Val} \wedge \Rrightarrow \Phi(e)) \\ & \vee \big(e \notin \mathit{Val} \wedge \forall \sigma.\, \mathsf{Own}(\bullet\, \sigma) \mathrel{-\!\!*} \Rrightarrow \mathrm{red}(e, \sigma) \\ & \quad \wedge \triangleright (\forall e_2, \sigma_2, \vec{e}_f.\, (e, \sigma) \to (e_2, \sigma_2, \vec{e}_f) \ \mathrel{-\!\!*} \Rrightarrow \\ & \qquad \mathsf{Own}(\bullet\, \sigma_2) * \mathsf{wp}\, e_2\, \{\Phi\} * \mathop{\Large *}_{e' \in \vec{e}_f} \mathsf{wp}\, e'\, \{v.\mathsf{True}\})\big) \\ \ell \mapsto v \triangleq\ & \mathsf{Own}(\circ\, [\ell := v])\end{aligned}$$

Instead of just demanding a proof of the weakest precondition of the thread e under consideration, we also demand proofs that all the forked-off threads $\vec{e}_f$ are safe. We do not care about their return values, so the postcondition is trivial.

This encoding shows how much mileage we get out of building on top of the Iris base logic. Because said logic supports ownership and step-indexing, we can get around explicitly managing resources and step-indices in the weakest precondition definition. We do not have to explicitly account for the way resources are subdivided between the current thread and the forked-off thread. Instead, all we have to do is surgically place some update modalities, a single $\triangleright$, and some standard separation logic connectives. This keeps the definition of, and the reasoning about, weakest preconditions nice and compact.

4 Recovering the Iris program logic

In this section, we show how to encode the reasoning principles of *full Iris* [18,17] within our base logic. The main remaining challenge is to encode *invariants*, which are the key feature for reasoning about *sharing* in concurrent programs [5].

An invariant is simply a property that holds at all times: each thread accessing the state may assume the invariant holds before each step of its computation, but it must also ensure that it continues to hold after each step. Since we work in a separation logic, the invariant does not just "hold"; it expresses ownership of some resources, and threads accessing the invariant get access to those resources. The rule that realizes this idea looks as follows:

$$\text{WP-INV}\quad \frac{\triangleright P \vdash \mathsf{wp}_{\mathcal{E}\setminus\{\iota\}}\, e\, \{v.\, \triangleright P * \Phi(v)\} \qquad \mathsf{atomic(e)} \qquad \iota \in \mathcal{E}}{\boxed{P}^{\iota} \vdash \mathsf{wp}_{\mathcal{E}}\, e\, \{\Phi\}}$$

This rule is quite a mouthful, so we will go over it carefully. First of all, there is a new assertion $\boxed{P}^{\iota}$, which states that P (an arbitrary assertion) is maintained as an invariant. The rule says that having this assertion in the context permits us to access the invariant, which involves acquiring ownership of $\triangleright P$ *before* the verification of e and giving back ownership of $\triangleright P$ *after* said verification. Crucially, we require that e is *atomic*, meaning that computation is guaranteed to complete in a single step. This is essential for soundness: the rule allows us to temporarily use and even break the invariant, but after a single atomic step (*i.e.*, before any other thread could take a turn), we have to establish it again.

The $\triangleright$ modality arises because of the inherently cyclic nature (*i.e., impredicativity*) of our invariants: P can be *any* assertion, including assertions about invariants. We will show in §5 that removing the $\triangleright$ leads to an unsound logic.

Finally, we come to the *mask* $\mathcal{E}$ and *invariant name* ι: they avoid the issue of *reentrancy*. We have to make sure that the same invariant is not accessed twice at the same time, as that would incorrectly duplicate the underlying resource. To this end, each invariant has a *name* ι identifying it. Furthermore, weakest preconditions are annotated with a *mask* to keep track of which invariants are still enabled. Accessing an invariant removes its name from the mask, ensuring that it cannot be accessed again in a nested fashion.

In order to recover the full power of the Iris program logic (including WP-INV), we start this section by lifting a limitation of the base logic, namely, that it is restricted to a *single* uRA of resources (§4.1). Then we explain how resources are used to keep track of invariants (§4.2), and define *world satisfaction*, a protocol enforcing how invariants are maintained (§4.3). We follow on by defining the *fancy update modality*, which supports accessing invariants (§4.4), before finally giving an enriched version of weakest preconditions that validates WP-INV (§4.5).

4.1 Dynamic composable resources

The base logic as described in §2 is limited to resources formed by a *single* RA. However, for the construction in this section, we will need multiple RAs, so we need to find a way to lift this limitation. Furthermore, we frequently need to use not just a single instance of an RA, but multiple, entirely independent instances (*e.g.,* one instance of the RA per instance of a data structure).

As prior work already observed [18,17], it turns out that RAs themselves are already flexible enough to solve this, we just have to pick the right RA. Concretely, assume we are given a *family* of RAs $(M_i)_{i\in\mathcal{I}}$ indexed by some finite index set $\mathcal{I}$. Then, we instantiate our base logic with the following global resource algebra:

$$M \triangleq \prod_{i\in\mathcal{I}} \mathbb{N} \xrightarrow{\text{fin}} M_i$$

First of all, we use a finite partial function to obtain an arbitrary number of instances of any of the given RAs. Furthermore, we take the product over the entire family to make all the chosen RAs available inside the logic.

Typically, we will only own some resource a in one particular instance named $\gamma \in \mathbb{N}$ of a given RA M_i. To express that, we introduce the following notation:

$$\boxed{a : M_i}^{\gamma} \triangleq \mathsf{Own}((\ldots, \emptyset, i : [\gamma := a], \emptyset, \ldots))$$

Often, we will even leave away the M_i because it is clear from context.

All the rules about $\mathsf{Own}(\cdot)$ can now also be derived for $\boxed{\cdot}^{\cdot}$. In addition, we obtain a rule to create new instances of RAs with an arbitrary valid initial state:

$$a \in \mathcal{V}_{M_i} \vdash \Rrightarrow \exists \gamma. \boxed{a : M_i}^{\gamma}$$

Obtaining modular proofs. Even with multiple RAs at our disposal, it may still seem like we have a modularity problem: every proof is done in an instantiation of Iris with some particular family of RAs. As a result, if two proofs make different choices about the RAs, they are carried out in entirely different logics and hence cannot be composed.

To solve this problem, we *generalize* our proofs over the family of RAs that Iris is instantiated with. So in the following, all proofs are carried out in Iris instantiated with some unknown $(M_i)_{i \in \mathcal{I}}$. If the proof needs a particular RA, it further assumes that there exists some j *s.t.* M_j is the desired RA. Composing two proofs is thus easily possible; the resulting proof works in any family of RAs that contains all the particular RAs needed by either proof. Finally, if we want to obtain a "closed form" of some particular proof in a concrete instance of Iris, we simply construct a family of RAs that contains everything the proof needs.

4.2 A registry of invariants

Since we wish to be able to share the $\boxed{P}^{\iota}$ assertion among threads, we will need a central "invariant registry" that keeps track of all invariants and witnesses the fact that P has been registered as invariant.

In §3.3, we already saw the authoritative resource algebra. This RA allowed us to have an "authoritative" registry with fragments shared by various parties. However, for the case of invariants, we are not interested in expressing exclusive ownership of invariants, like we did for heap locations. Instead, the entire point of invariants is *sharing*, so we need that everybody *agrees* on what the invariant with a given name is. An RA for *agreement* on a set X is defined by:

$$\textsc{Ag}(X) \triangleq \mathsf{ag}(x : X) \mid \lightning \qquad \mathcal{V} \triangleq \{\mathsf{ag}(x) \mid x \in X\}$$

$$\mathsf{ag}(x) \cdot \mathsf{ag}(y) \triangleq \begin{cases} \mathsf{ag}(x) & \text{if } x = y \\ \lightning & \text{otherwise} \end{cases} \qquad |\mathsf{ag}(x)| \triangleq \mathsf{ag}(x)$$

The key property of this RA is that from $\mathsf{ag}(x) \cdot \mathsf{ag}(y) \in \mathcal{V}$, we can deduce $x = y$.

We can then compose our RAs as follows to obtain an "invariant registry":

$$\textsc{inv} \triangleq \textsc{Auth}(\mathbb{N} \xrightarrow{\text{fin}} \textsc{Ag}(\blacktriangleright \mathsf{Prop}))$$

This construction is an example of *higher-order ghost state*, which we already mentioned in §2.6. The Prop here is actually a recursive occurrence of logical assertions within resources, which has to be *guarded* by a "type-level later" $\blacktriangleright$. Furthermore, to make this really work, the agreement RA must be generalized to a proper CMRA (§2.6), so the actual definition is more involved. See the Iris 2.0 paper for details [17].

For present purposes, the only relevant outcome is the following assertions:

- $\boxed{\circ [\iota := P]}^{\gamma}$, stating that P is registered as an invariant with name ι; and
- $\boxed{\bullet I}^{\gamma}$, stating that $I \in \mathbb{N} \xrightarrow{\text{fin}} \mathsf{Prop}$ is the full map of all registered invariants.

These assertions enjoy the following three rules:

$$\boxed{\circ[\iota := P]}^{\gamma_{\text{INV}}} \vdash \Box \boxed{\circ[\iota := P]}^{\gamma_{\text{INV}}} \qquad \text{(INVREG-PERSIST)}$$

$$\boxed{\bullet I}^{\gamma_{\text{INV}}} * \boxed{\circ[\iota := P]}^{\gamma_{\text{INV}}} \vdash \triangleright I(\iota) \Leftrightarrow \triangleright P \qquad \text{(INVREG-AGREE)}$$

$$\iota \notin \mathrm{dom}(I) \land \boxed{\bullet I}^{\gamma_{\text{INV}}} \vdash \Rrightarrow \boxed{\bullet I[\iota := P]}^{\gamma_{\text{INV}}} * \boxed{\circ[\iota := P]}^{\gamma_{\text{INV}}} \qquad \text{(INVREG-ALLOC)}$$

Intuitively, INVREG-PERSIST states that the non-authoritative fragment is persistent, *i.e.,* that it can be freely moved below the $\Box$ modality and shared. INVREG-AGREE witnesses that the registry and the fragments *agree* on the proposition managed at a particular name. Note that we only get the equivalence with a $\triangleright$ because the definition of the RA (INV) contains a $\blacktriangleright$. Finally, INVREG-ALLOC lets one create a new invariant, provided the new name is not already used.

4.3 World satisfaction

To recover the invariant mechanism of Iris, we need to attach a meaning to the invariant registry from §4.2, in the sense that we must make sure that the invariants actually *hold*! We do this by defining a single global invariant called *world satisfaction*, which enforces the meaning of the invariant registry. World satisfaction itself will be enforced by threading it through the weakest preconditions.

Naively, we may think that world satisfaction always requires all invariants to hold. However, this does not work: after all, threads are allowed to *temporarily break* invariants for an atomic "instant" during program execution. To support this, world satisfaction keeps invariants in one of two states: either they are *enabled* (currently enforced), or they are *disabled* (currently broken by some thread). The definition of the weakest precondition connective will then ensure that invariants are never disabled for more than an atomic period of time. That is, no invariant is left disabled between physical computation steps.

The protocol for opening (*i.e.,* disabling) and closing (*i.e.,* re-enabling) an invariant employs two *exclusive tokens*: an *enabled* token, which witnesses that the invariant is currently enabled and giving the right to disable it; and dually, a *disabled* token. These tokens are controlled by the following two simple RAs:

$$\textsc{En} \triangleq \wp(\mathbb{N}) \qquad\qquad \textsc{Dis} \triangleq \wp^{\text{fin}}(\mathbb{N})$$

The composition for both RAs is disjoint union.[5]

We can now give the actual definition of world satisfaction, W. To this end, we need instances of INV, EN and DIS, which we assume to have names γ_{INV}, γ_{EN} and γ_{DIS}, respectively:

$$W \triangleq \exists I.\ \boxed{\bullet I}^{\gamma_{\text{INV}}} * \mathop{\Large *}_{\iota \in \mathrm{dom}(I)} \left((\triangleright I(\iota) * \boxed{\{\iota\}}^{\gamma_{\text{DIS}}}) \lor \boxed{\{\iota\}}^{\gamma_{\text{EN}}} \right)$$

$$\boxed{P}^{\iota} \triangleq \boxed{\circ[\iota := P]}^{\gamma_{\text{INV}}}$$

[5] Implicitly, they also have an *invalid* element $\lightning$, for composition of overlapping sets.

World satisfaction controls the authoritative registry I of *all* existing invariants. This allows it to maintain an additional assertion for every single one of them, namely: either the invariant is enabled and maintained—in which case world satisfaction actually *owns* $\triangleright I(\iota)$—or the invariant is disabled. Unsurprisingly, $\boxed{P}^{\iota}$ just means that the registry maps ι to P—but ι may or may not be enabled.

With this encoding, we can prove the following key properties modeling the allocation, opening, and closing of invariants:

$$\textsc{wsat-alloc}\ \frac{\mathcal{E}\text{ is infinite}}{W * \triangleright P \twoheadrightarrow \Rrightarrow\big(W * \exists \iota \in \mathcal{E}.\, \boxed{P}^{\iota}\big)} \qquad \textsc{wsat-openclose}\ \boxed{P}^{\iota} \vdash W * \boxed{\{\iota\}}^{\gamma_{\text{EN}}} \Leftrightarrow W * \triangleright P * \boxed{\{\iota\}}^{\gamma_{\text{DIS}}}$$

Let us look at the proof of the direction $\boxed{P}^{\iota} \vdash W * \boxed{\{\iota\}}^{\gamma_{\text{EN}}} \Rightarrow W * \triangleright P * \boxed{\{\iota\}}^{\gamma_{\text{DIS}}}$ of WSAT-OPENCLOSE in slightly more detail. We start by using INVREG-AGREE to learn that the authoritative registry I maintained by world satisfaction contains our invariant P at index ι. We thus obtain from the big separating conjunction that $\triangleright P * \boxed{\{\iota\}}^{\gamma_{\text{DIS}}} \vee \boxed{\{\iota\}}^{\gamma_{\text{EN}}}$. Since we moreover own the enabled token $\boxed{\{\iota\}}^{\gamma_{\text{EN}}}$, we can exclude the right disjunct and deduce that the invariant is currently enabled. So we take out the $\triangleright P$ and the disabled token, and instead put the enabled token into W, disabling the invariant. This concludes the proof.

The proof of WSAT-ALLOC is slightly more subtle. In particular, we have to be careful in picking the new invariant name such that: (a) it is in $\mathcal{E}$, (b) it is not used in I yet, and (c) we can create a disabled token for that name and put it into W alongside $\triangleright P$. Since disabled tokens are modeled by *finite* sets, only finitely many of them can ever be allocated, so it is always possible to pick an appropriate fresh name.

4.4 Fancy update modality

Before we will prove the rules for invariants, there is actually one other piece of the original Iris logic we should cover: *view shifts*. View shifts serve three roles:

1. They permit frame-preserving updates (like the basic update modality does).
2. They allow one to access invariants. The *mask* $\mathcal{E}$ defines which invariants are available.
3. They allow one to strip away the $\triangleright$ modality from timeless assertions (like the $\diamond$ modality does, see §2.7).

The view shifts of the original Iris were of the form $P \;{}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}\; Q$ where P is the precondition, Q the postcondition, and $\mathcal{E}_1$ and $\mathcal{E}_2$ are invariant masks. For the same reason that we prefer weakest preconditions over Hoare triples (§3), we will present view shifts as a modality instead of a binary connective. The modality, called the *fancy update modality* ${}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}$, is defined as follows:

$${}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2} P \triangleq W * \boxed{\mathcal{E}_1}^{\gamma_{\text{EN}}} \twoheadrightarrow \Rrightarrow \diamond (W * \boxed{\mathcal{E}_2}^{\gamma_{\text{EN}}} * P) \qquad \Rrightarrow_{\mathcal{E}} P \triangleq {}^{\mathcal{E}}\!\Rrightarrow^{\mathcal{E}} P$$

In the same way that Hoare triples are defined in terms of weakest preconditions, the binary view shift can be defined in terms of the modality.

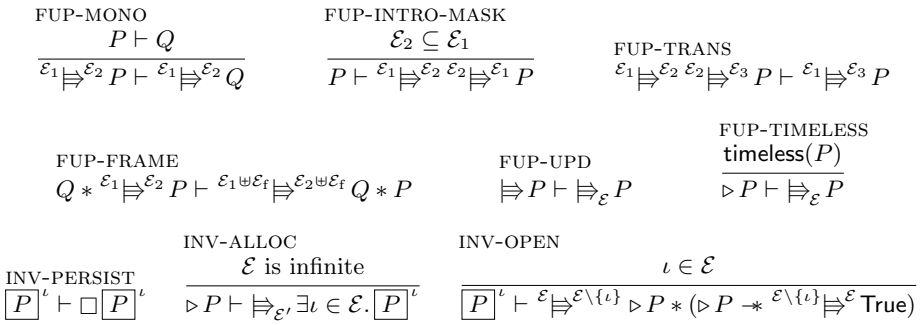

Fig. 5. Rules for the fancy update modality and invariants.

The intuition behind ${}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2} P$ is to express ownership of resources such that, if we further assume that the invariants in $\mathcal{E}_1$ are enabled, we can perform a *frame-preserving update* to the resources and the invariants, and we end up owning P and the invariants in $\mathcal{E}_2$ are enabled. By looking at the definition, we can see how it supports all the fancy features formerly handled by view shifts:

1. At the heart of the fancy update modality is a basic update modality, which permits doing frame-preserving updates (see the rule FUP-UPD in Figure 5).
2. The modality "threads through" world satisfaction, in the sense that a proof of ${}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2} P$ can use W, but also has to prove it again. Furthermore, controlled by the two masks $\mathcal{E}_1$ and $\mathcal{E}_2$, the modality provides and takes away enabled tokens. The first mask controls which invariants are available to the modality, while the second mask controls which invariants remain available after (see INV-OPEN). Furthermore, it is possible to allocate new invariants (INV-ALLOC).
3. Finally, the modality is able to remove laters from timeless assertions by incorporating the "except 0" modality $\diamond$ (see §2.7 and FUP-TIMELESS).

Ignoring the style of presentation as a modality, there are some differences here from view shifts in previous versions of Iris. Firstly, in previous versions, the rule FUP-TRANS had a side condition restricting the masks it could be instantiated with, whereas now it does not. Secondly, in previous versions, instead of FUP-INTRO-MASK, only mask-invariant view shifts could be introduced ($P \vdash \Rightarrow_{\mathcal{E}} P$). The reason we can now support FUP-INTRO-MASK is that masks are actually just sugar for owning or providing particular *resources* (namely, the enabled tokens). This is in contrast to previous versions of Iris, where masks were entirely separate from resources and treated in a rather ad-hoc manner. Our more principled treatment of masks significantly simplifies building abstractions involving invariants; however, for lack of space, we cannot further discuss these abstractions.

The rules FUP-MONO, FUP-TRANS, and FUP-FRAME correspond to the related rules of the basic update modality in Figure 1. The rule INV-OPEN may look fairly cryptic; we will see in the next section how it can be used to derive WP-INV.

WP-VUP
$$\Rrightarrow_{\mathcal{E}} \mathsf{wp}_{\mathcal{E}}\, e\, \{v.\, \Rrightarrow_{\mathcal{E}} \Phi(v)\} \vdash \mathsf{wp}_{\mathcal{E}}\, e\, \{\Phi\}$$

WP-ATOMIC
$$\frac{\mathsf{atomic}(e)}{{}^{\mathcal{E}_1}\Rrightarrow^{\mathcal{E}_2} \mathsf{wp}_{\mathcal{E}_2}\, e\, \left\{v.\, {}^{\mathcal{E}_2}\Rrightarrow^{\mathcal{E}_1} \Phi(v)\right\} \vdash \mathsf{wp}_{\mathcal{E}_1}\, e\, \{\Phi\}}$$

Fig. 6. New rules for weakest precondition with invariants.

4.5 Weakest preconditions

We will now define weakest preconditions that support not only the rules in Figure 4, but also the ones in Figure 6. We will also show how, from WP-ATOMIC and INV-OPEN, we can derive the rule motivating this entire section, WP-INV.

Compared to the definition developed in §3, there are two key differences: first of all, we use the fancy update modality instead of the basic update modality. Secondly, we do not want to tie the definition of weakest preconditions to a particular language, and instead operate generically over any notion of expressions and state, and any reduction relation.

As a consequence of this generality, we can no longer assume that our physical state is a heap of values with disjoint union as composition. Therefore, instead of using the authoritative heap defined in §3.3, we parameterize weakest preconditions by a predicate $I : \mathit{State} \to \mathit{iProp}$ called the *state interpretation*. In case $\mathit{State} = \mathit{Loc} \xrightarrow{\text{fin}} \mathit{Val}$, we can recover the definition and rules from §3.3 by taking:

$$I(\sigma) \triangleq \boxed{\bullet\, \sigma : \mathit{Loc} \xrightarrow{\text{fin}} \mathrm{Ex}(\mathit{Val})}^{\gamma}$$

More sophisticated forms of separation like fractional permissions [8,7] can be encoded by using an appropriate RA and defining I accordingly.

Given an $I : \mathit{State} \to \mathit{iProp}$, our definition of weakest precondition looks as follows (changes from §3 are colored red):

$$\begin{aligned}\mathsf{wp}_{\mathcal{E}}\, e\, \{\Phi\} \triangleq\ & (e \in \mathit{Val} \wedge \Rrightarrow_{\mathcal{E}} \Phi(e)) \\ & \vee \big(e \notin \mathit{Val} \wedge \forall \sigma.\, I(\sigma) \mathrel{-\!\!*} {}^{\mathcal{E}}\Rrightarrow^{\emptyset} \mathrm{red}(e, \sigma) \\ & \quad \wedge \triangleright (\forall e_2\, \sigma_2\, \vec{e}_f.\, (e, \sigma) \to (e_2, \sigma_2, \vec{e}_f) \mathrel{-\!\!*} {}^{\emptyset}\Rrightarrow^{\mathcal{E}} \\ & \qquad I(\sigma_2) * \mathsf{wp}_{\mathcal{E}}\, e_2\, \{\Phi\} * \mathop{\Large *}_{e' \in \vec{e}_f} \mathsf{wp}_{\top}\, e'\, \{v.\mathsf{True}\})\big)\end{aligned}$$

The mask $\mathcal{E}$ of $\mathsf{wp}_{\mathcal{E}}\, e\, \{\Phi\}$ is used for the "outside" of the fancy update modalities, providing them with access to these invariants. The "inner" masks are $\emptyset$, indicating that the reasoning about safety and progress can temporarily open *all* invariants (and none have to be left enabled). The forked-off threads $\vec{e}_f$ have access to the full mask $\top$ as they will only start running in the *next* instruction, so they are not constrained by whatever invariants are available right now. Note that the definition requires all invariants in $\mathcal{E}$ to be enabled again after every physical step: this corresponds to the fact that an invariant can only be opened atomically.

In addition to the rules already presented in §3, this version of the weakest precondition connective lets us prove (among others) the new rules in Figure 6.

WP-VUP witnesses that the entire connective as well as its postcondition are living below the fancy update modality, so we can freely add/remove that modality.

Finally, we come to WP-ATOMIC to open an invariant around an atomic expression. The rule is similar to WP-VUP, with the key difference being that it can *change the mask*. On the left hand side of the turnstile, we are allowed to first open some invariants, then reason about e, and then close invariants again. This is sound because e is atomic. WP-ATOMIC is the rule we need to derive WP-INV:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\triangleright P \vdash \mathsf{wp}_{\mathcal{E}\setminus\{\iota\}}\, e\, \{v.\, \triangleright P * \Phi(v)\}}{\triangleright P * (\triangleright P \mathrel{-\!\!*} {}^{\mathcal{E}\setminus\{\iota\}}\!\Rrightarrow^{\mathcal{E}} \mathsf{True}) \vdash \mathsf{wp}_{\mathcal{E}\setminus\{\iota\}}\, e \left\{v.\, \triangleright P * \Phi(v) * (\triangleright P \mathrel{-\!\!*} {}^{\mathcal{E}\setminus\{\iota\}}\!\Rrightarrow^{\mathcal{E}_1} \mathsf{True})\right\}}\,\text{WP-FRAME}}{\triangleright P * (\triangleright P \mathrel{-\!\!*} {}^{\mathcal{E}\setminus\{\iota\}}\!\Rrightarrow^{\mathcal{E}} \mathsf{True}) \vdash \mathsf{wp}_{\mathcal{E}\setminus\{\iota\}}\, e \left\{v.\, {}^{\mathcal{E}\setminus\{\iota\}}\!\Rrightarrow^{\mathcal{E}} \Phi(v)\right\}}\,\text{WP-MONO}}{\boxed{P}^{\iota} \vdash {}^{\mathcal{E}}\!\Rrightarrow^{\mathcal{E}\setminus\{\iota\}} \mathsf{wp}_{\mathcal{E}\setminus\{\iota\}}\, e \left\{x.\, {}^{\mathcal{E}\setminus\{\iota\}}\!\Rrightarrow^{\mathcal{E}} \Phi(v)\right\}}\,\text{INV-OPEN}}{\boxed{P}^{\iota} \vdash \mathsf{wp}_{\mathcal{E}}\, e\, \{\Phi\}}\,\text{WP-ATOMIC}$$

4.6 Adequacy

To demonstrate that $\mathsf{wp}\, e\, \{\phi\}$ actually makes the expected statements about program executions, we prove the following *adequacy theorem*.

Theorem 4.1 (Adequacy of weakest preconditions). *Let ϕ be a first-order predicate. If $\mathsf{True} \vdash \Rrightarrow_{\top} I(\sigma) * \mathsf{wp}_{\top}\, e\, \{\phi\}$ and $(e, \sigma) \to^*_{\mathsf{tp}} (e'_1 \ldots e'_n, \sigma')$, then:*

1. *For any e'_i we have that either e'_i is a value, or $\mathrm{red}(e'_i, \sigma')$;*
2. *If e'_1 (the main thread) is a value v, then $\phi(v)$.*

The proof of this theorem relies on Theorem 2.1 (in §2.8). We also impose the same restrictions on ϕ as we have done there: ϕ has to be a *first-order* predicate. This ensures we can use ϕ both *inside* our logic and at the meta level.

5 Paradoxes involving the "later" modality

A recurring element of concurrent separation logics with impredicative invariants [28,18,17] is the later modality $\triangleright$, which is used to guard resources when opening invariants. The use of $\triangleright$ has heretofore been forced by the models which were used to show soundness of these logics. It has been an open question, however, whether the need for the later modality is a mere artifact of the model, or whether it is in some sense required. In this section, we show that at the very least it plays an *essential* role: if we omit the later modality from the invariant opening rule, then we can derive a contradiction *in the logic*.

Theorem 5.1. *Assume we add the following proof rule to Iris:*

$$\dfrac{\iota \in \mathcal{E}}{\boxed{P}^{\iota} \vdash {}^{\mathcal{E}}\!\Rrightarrow^{\mathcal{E}\setminus\{\iota\}} P * (P \mathrel{-\!\!*} {}^{\mathcal{E}\setminus\{\iota\}}\!\Rrightarrow^{\mathcal{E}} \mathsf{True})}$$

Then, if we pick an appropriate RA, $\mathsf{True} \vdash \Rrightarrow_{\top} \mathsf{False}$.

Notice that the above rule is the same as INV-OPEN in Figure 5, *except that it does not add a* $\triangleright$ *in front of* P.

Of course, this does not prove that we absolutely must have a $\triangleright$ modality, but it *does* show that the stronger rule one would prefer to have for invariants is unsound. Step-indexing is but one way to navigate around this unsoundness. However, we are not aware of another technique that would yield a logic with comparably powerful impredicative invariants.

The proof of this theorem does not use the fact that fancy updates are defined in a particular way in terms of basic updates, but just uses the proof rules for this modality. The proof also makes no use of higher-order ghost state. In fact, the result holds for all versions of Iris [18,17], as is shown by the following theorem:

Theorem 5.2. *Assume a higher-order separation logic with* $\Box$ *and an update modality with a binary mask* $\Rightarrow\!\!\!\!\Rightarrow_{\{0,1\}}$ *(think: empty mask and full mask) satisfying strong monad rules with respect to separating conjunction and such that:*

$$\text{WEAKEN-MASK}\quad \Rightarrow\!\!\!\!\Rightarrow_0 P \vdash \Rightarrow\!\!\!\!\Rightarrow_1 P$$

Assume a type $\mathcal{I}$ *and an assertion* $\boxed{\cdot}^{\cdot} : \mathcal{I} \to \mathit{Prop} \to \mathit{Prop}$ *satisfying:*

$$\text{INV-ALLOC}\quad P \vdash \Rightarrow\!\!\!\!\Rightarrow_1 \exists \iota.\, \boxed{P}^{\iota} \qquad \text{INV-PERSIST}\quad \boxed{P}^{\iota} \vdash \Box \boxed{P}^{\iota} \qquad \text{INV-OPEN-NOLATER}\quad \frac{P * Q \vdash \Rightarrow\!\!\!\!\Rightarrow_0 (P * R)}{\boxed{P}^{\iota} * Q \vdash \Rightarrow\!\!\!\!\Rightarrow_1 R}$$

Finally, assume the existence of a type $\mathcal{G}$ *and two tokens* $\boxed{\mathrm{S}}^{\cdot} : \mathcal{G} \to \mathit{Prop}$ *and* $\boxed{\mathrm{F}}^{\cdot} : \mathcal{G} \to \mathit{Prop}$ *parameterized by* $\mathcal{G}$ *and satisfying the following properties:*

$$\text{START-ALLOC}\quad \vdash \Rightarrow\!\!\!\!\Rightarrow_0 \exists \gamma.\, \boxed{\mathrm{S}}^{\gamma} \qquad \text{START-FINISH}\quad \boxed{\mathrm{S}}^{\gamma} \vdash \Rightarrow\!\!\!\!\Rightarrow_0 \boxed{\mathrm{F}}^{\gamma} \qquad \text{START-NOT-FINISHED}\quad \boxed{\mathrm{S}}^{\gamma} * \boxed{\mathrm{F}}^{\gamma} \vdash \mathit{False} \qquad \text{FINISHED-DUP}\quad \boxed{\mathrm{F}}^{\gamma} \vdash \boxed{\mathrm{F}}^{\gamma} * \boxed{\mathrm{F}}^{\gamma}$$

Then $\mathit{True} \vdash \Rightarrow\!\!\!\!\Rightarrow_1 \mathit{False}$.

In other words, the theorem requires three ingredients to be present in the logic in order to derive a contradiction:

- An update modality that satisfies the laws of Iris's basic update modality (Figure 1). The modality needs a mask for the same reason that Iris's fancy update modality has a mask: to prevent opening the same invariant twice.
- Invariants that can be opened around the update modality, and that can be opened without a later.
- A two-state protocol whose only transition is from the first to the last state. This is what $\boxed{\mathrm{S}}^{\cdot}$ and $\boxed{\mathrm{F}}^{\cdot}$ encode. The proof does not actually depend on how that protocol is made available to the logic. For example, to apply this proof to iCAP [28], one could use iCAP's built-in support for state-transition systems to achieve the same result. However, for the purpose of the theorem, we had to pick *some* way of expressing protocols. We picked the token-based approach common in Iris.

All versions of Iris easily satisfy the first and third of these requirements, by using fancy updates (Iris 3) or primitive view shifts (Iris 1 and 2) for the update modality, and by constructing an appropriate RA (Iris 2 and 3) or PCM (Iris 1) for the two-state protocol. Of course, INV-OPEN-NOLATER is the one assumption of the theorem that no version of Iris satisfies, which is the entire point.

Unsurprisingly, the proof works by constructing an assertion that is equivalent (in some rather loose sense) to its own negation. The full details of this construction are spelled out in the appendix [1].

6 Related work

Since O'Hearn introduced the original concurrent separation logic (CSL) [24], many more CSLs have been developed [12,11,16,14,29,28,30,9,27,18,17]. Though these logics have explored different techniques for reasoning about concurrency, they have one thing in common: their proof rules and models are complicated.

There have been attempts at mitigating the difficulty of the models of these logics. Most notably, Svendsen and Birkedal [28] defined the model of the iCAP logic in the internal logic of the topos of trees, which includes a *later* connective to reason about step-indexing abstractly. However, their model of Hoare triples still involves explicit resource management, which ours does not.

On the other end of the spectrum, there has been work on encoding binary logical relations in a concurrent separation logic [13,30,22,21]. These encodings are relying on a base logic that already includes a plethora of high-level concepts, such as weakest preconditions and view shifts. Our goal, in contrast, is precisely to *define* these concepts in simpler terms.

FCSL [27] takes an opposite approach to our work. To ease reasoning about programs in a proof assistant, they avoid reasoning in separation logic as much as possible, and reason mostly in the model of their logic. This requires the model to stay as simple as possible; in particular, FCSL does not make use of step-indexing. As a consequence, they do not support impredicative invariants, which we believe are an important feature of Iris. For example, they are needed to model impredicative type systems [21] or to model a reentrant event loop library [28]. Furthermore, as we have shown in recent work [21], one *can* actually reason conveniently in a separation logic in Coq, so the additional complexity of our model is hardly visible to users of our logic.

Additionally, there is a difference in expressiveness w.r.t. "hiding" of invariants. FCSL supports a certain kind of hiding, namely the ability to transfer some local state into an invariant (actually a "concurroid"), which is enforced during the execution of a single expression e, but after which the state governed by the invariant is returned to local control. Iris can support such hiding as well, via an encoding of what we call "cancelable invariants" [1]. Additionally, we allow a different kind of hiding, namely the ability to hide invariants used by (nested) Hoare-triple specifications. For example, a higher-order function f may return another function g, whose Hoare-triple specification is only correct under some invariant I (which was established during execution of f). Since invariants in Iris

are persistent assertions, I can be hidden, *i.e.,* it need not infect the specification of f or g. To our knowledge, FCSL does not support hiding of this form.

The Verified Software Toolchain (VST) [4] is a framework that provides machinery for constructing sophisticated higher-order separation logics with support for impredicative invariants in Coq. However, VST is not a logic and, as such, does not abstract over step-indices and resources the way working in a logic like Iris 3.0 does. Defining a program logic in VST thus still requires significant manual management of such details, which are abstracted away when defining a program logic in the Iris base logic. Furthermore, VST has so far only been demonstrated in the context of sequential reasoning and coarse-grained (lock-based) concurrency [6], whereas the focus of Iris is on fine-grained concurrency.

7 Conclusion

We have presented a minimal *base logic* in which we can define concurrent separation logics in a concise and abstract way. This has the benefit of making higher-level concepts (like weakest preconditions) easier to define, easier to understand, and easier to reason about.

Definitions become simpler as they can be performed at a much higher level of abstraction. In particular, the definitions of logical connectives such as the fancy update modality and weakest preconditions do not have to deal with any details about disjointness of resources or step-indexing—this is all abstractly handled by the base logic. Proofs become simpler since only the rules of the primitive connectives of the base logic have to be verified w.r.t. the model. The proofs about fancier connectives are carried out *inside* the logic, again abstracting over details that have to be managed manually when working in the model.

Thanks to these simplifications, we are able now, for the first time, to explain what the program logic connectives in Iris actually *mean.* Furthermore, we have ported the Coq formalization of Iris [1], including a rich body of examples, over to the new connectives defined in the base logic. The interactive proof mode (IPM) [21] provided crucial tactic support for reasoning with interesting combinations of separation-logic assertions and our modalities (as they arise, *e.g.,* in weakest preconditions). In performing the port, the definitions and proofs related to weakest preconditions, view shifts, and invariants shrank in size significantly, indicating that proofs and definitions can now be carried out with considerably greater ease.

Acknowledgments. 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); and by the ModuRes Sapere Aude Advanced Grant from The Danish Council for Independent Research for the Natural Sciences (FNU).

References

1. The Iris 3.0 documentation and Coq development. Available on the Iris project website at: http://iris-project.org.
2. A. Appel and D. McAllester. An indexed model of recursive types for foundational proof-carrying code. *TOPLAS*, 23(5):657–683, 2001.
3. A. Appel, P.-A. Melliès, C. Richards, and J. Vouillon. A very modal model of a modern, major, general type system. In *POPL*, 2007.
4. A. W. Appel, editor. *Program Logics for Certified Compilers*. Cambridge University Press, 2014.
5. E. A. Ashcroft. Proving assertions about parallel programs. *JCSS*, 10(1):110–135, 1975.
6. L. Beringer, G. Stewart, R. Dockins, and A. W. Appel. Verified compilation for shared-memory C. In *ESOP*, 2014.
7. R. Bornat, C. Calcagno, P. W. O'Hearn, and M. J. Parkinson. Permission accounting in separation logic. In *POPL*, pages 259–270, 2005.
8. J. Boyland. Checking interference with fractional permissions. In *SAS*, volume 2694 of *LNCS*, pages 55–72, 2003.
9. P. da Rocha Pinto, T. Dinsdale-Young, and P. Gardner. TaDA: A logic for time and data abstraction. In *ECOOP*, pages 207–231, 2014.
10. E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *CACM*, 18(8):453–457, 1975.
11. T. Dinsdale-Young, M. Dodds, P. Gardner, M. Parkinson, and V. Vafeiadis. Concurrent abstract predicates. In *ECOOP*, pages 504–528, 2010.
12. M. Dodds, X. Feng, M. J. Parkinson, and V. Vafeiadis. Deny-guarantee reasoning. In *ESOP*, volume 5502 of *LNCS*, pages 363–377, 2009.
13. D. Dreyer, G. Neis, A. Rossberg, and L. Birkedal. A relational modal logic for higher-order stateful ADTs. In *POPL*, 2010.
14. A. Hobor, A. Appel, and F. Zappa Nardelli. Oracle semantics for concurrent separation logic. In *ESOP*, pages 353–367, 2008.
15. S. S. Ishtiaq and P. W. O'Hearn. BI as an assertion language for mutable data structures. In *POPL*, pages 14–26, 2001.
16. B. Jacobs and F. Piessens. Expressive modular fine-grained concurrency specification. In *POPL*, 2011.
17. R. Jung, R. Krebbers, L. Birkedal, and D. Dreyer. Higher-order ghost state. In *ICFP*, pages 256–269, 2016.
18. R. Jung, D. Swasey, F. Sieczkowski, K. Svendsen, A. Turon, L. Birkedal, and D. Dreyer. Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. In *POPL*, pages 637–650, 2015.
19. A. Kock. Monads on symmetric monoidal closed categories. *Archiv der Mathematik*, 21(1):1–10, 1970.
20. A. Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 23(1):113–120, 1972.
21. R. Krebbers, A. Timany, and L. Birkedal. Interactive proofs in higher-order concurrent separation logic. In *POPL*, pages 205–217, 2017.
22. M. Krogh-Jespersen, K. Svendsen, and L. Birkedal. A relational model of types-and-effects in higher-order concurrent separation logic. In *POPL*, 2017.
23. H. Nakano. A modality for recursion. In *LICS*, 2000.
24. P. O'Hearn. Resources, concurrency, and local reasoning. *TCS*, 375(1):271–307, 2007.

25. P. W. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 1999.
26. M. J. Parkinson. The next 700 separation logics - (Invited paper). In *VSTTE*, volume 6217 of *LNCS*, pages 169–182, 2010.
27. I. Sergey, A. Nanevski, and A. Banerjee. Mechanized verification of fine-grained concurrent programs. In *PLDI*, pages 77–87, 2015.
28. K. Svendsen and L. Birkedal. Impredicative concurrent abstract predicates. In *ESOP*, pages 149–168, 2014.
29. K. Svendsen, L. Birkedal, and M. J. Parkinson. Modular reasoning about separation of concurrent data structures. In *ESOP*, pages 169–188, 2013.
30. A. Turon, D. Dreyer, and L. Birkedal. Unifying refinement and Hoare-style reasoning in a logic for higher-order concurrency. In *ICFP*, pages 377–390, 2013.