The Iris 4.4 dev Reference

https://iris-project.org/

April 23, 2026

Abstract

This document formally describes the Iris program logic. Every result in this document has been fully verified in Coq. The latest versions of this document and the Coq formalization can be found in the git repository at https://gitlab.mpi-sws.org/iris/iris. For further information, visit the Iris project website at https://iris-project.org.

Contents

1 Iris from the Ground Up

In "Iris from the Ground Up" [6], we describe Iris 3.1 in a bottom-up way. That paper is hence much more suited as an introduction to the model of Iris than this reference, which mostly contains definitions, not explanations or examples. For a list of changes in Iris since then, please consult our changelog at https://gitlab.mpi-sws.org/iris/iris/blob/master/CHANGELOG.md.

2 Algebraic Structures

2.1 OFE

The model of Iris lives in the category of *Ordered Families of Equivalences* (OFEs). This definition varies slightly from the original one in [2].

Definition 1. *An* ordered family of equivalences *(OFE) is a tuple* $(T, (\overset{n}{=} \subseteq T \times T)_{n\in\mathbb{N}})$ *satisfying*

$$\forall n.\ (\overset{n}{=})\ \textit{is an equivalence relation} \tag{OFE-EQUIV}$$
$$\forall n, m.\ n \geq m \Rightarrow (\overset{n}{=}) \subseteq (\overset{m}{=}) \tag{OFE-MONO}$$
$$\forall x, y.\ x = y \Leftrightarrow (\forall n.\ x \overset{n}{=} y) \tag{OFE-LIMIT}$$

The key intuition behind OFEs is that elements x and y are n-equivalent, notation $x \overset{n}{=} y$, if they are *equivalent for n steps of computation*, *i.e.,* if they cannot be distinguished by a program running for no more than n steps. In other words, as n increases, $\overset{n}{=}$ becomes more and more refined (OFE-MONO)—and in the limit, it agrees with plain equality (OFE-LIMIT).

Notice that OFEs are just a different presentation of bisected 1-bounded ultrametric spaces, where the family of equivalence relations gives rise to the distance function (two elements that are equal for n steps are no more than 2^{-n} apart).

Definition 2. *An element $x \in T$ of an OFE is called* discrete *if*

$$\forall y \in T.\ x \overset{0}{=} y \Rightarrow x = y$$

An OFE A is called discrete *if all its elements are discrete. For a set X, we write ΔX for the discrete OFE with $x \overset{n}{=} x' \triangleq x = x'$*

Definition 3. *A function $f : T \to U$ between two OFEs is* non-expansive *(written $f : T \overset{\mathsf{ne}}{\to} U$) if*

$$\forall n, x \in T, y \in T.\ x \overset{n}{=} y \Rightarrow f(x) \overset{n}{=} f(y)$$

It is contractive *if*

$$\forall n, x \in T, y \in T.\ (\forall m < n.\ x \overset{m}{=} y) \Rightarrow f(x) \overset{n}{=} f(y)$$

Intuitively, applying a non-expansive function to some data will not suddenly introduce differences between seemingly equal data. Elements that cannot be distinguished by programs within n steps remain indistinguishable after applying f.

Definition 4. *The category* **OFE** *consists of OFEs as objects, and non-expansive functions as arrows.*

Note that **OFE** is bicartesian closed, *i.e.,* it has all sums, products and exponentials as well as an initial and a terminal object. In particular:

Definition 5. *Given two OFEs T and U, the set of non-expansive functions $\{f : T \overset{\mathsf{ne}}{\to} U\}$ is itself an OFE with*

$$f \overset{n}{=} g \triangleq \forall x \in T.\ f(x) \overset{n}{=} g(x)$$

Definition 6. *A (bi)functor $F : \mathbf{OFE} \to \mathbf{OFE}$ is called* locally non-expansive *if its action F_1 on arrows is itself a non-expansive map. Similarly, F is called* locally contractive *if F_1 is a contractive map.*

The function space $(-) \xrightarrow{\mathsf{ne}} (-)$ is a locally non-expansive bifunctor. Note that the composition of non-expansive (bi)functors is non-expansive, and the composition of a non-expansive and a contractive (bi)functor is contractive.

One very important OFE is the OFE of *step-indexed propositions*: For every step-index, such a proposition either holds or does not hold. Moreover, if a propositions holds for some n, it also has to hold for all smaller step-indices.

$$\begin{aligned} \mathit{SProp} &\triangleq \wp^{\downarrow}(\mathbb{N}) \\ &\triangleq \{X \in \wp(\mathbb{N}) \mid \forall n, m.\, n \geq m \Rightarrow n \in X \Rightarrow m \in X\} \\ X \overset{n}{=} Y &\triangleq \forall m \leq n.\, m \in X \Leftrightarrow m \in Y \\ X \overset{n}{\subseteq} Y &\triangleq \forall m \leq n.\, m \in X \Rightarrow m \in Y \end{aligned}$$

2.2 COFE

COFEs are *complete OFEs*, which means that we can take limits of arbitrary chains.

Definition 7 (Chain)**.** *Given some set T and an indexed family $(\overset{n}{=} \subseteq T \times T)_{n \in \mathbb{N}}$ of equivalence relations, a* chain *$c \in \mathit{Chains}(T)$ is a function $c : \mathbb{N} \to T$ such that $\forall n, m.\, n \leq m \Rightarrow c(m) \overset{n}{=} c(n)$.*

Definition 8. *A* complete ordered family of equivalences *(COFE) is a tuple $(T : \mathbf{OFE}, \mathsf{lim} : \mathit{Chains}(T) \to T)$ satisfying*

$$\forall n, c.\, \mathsf{lim}(c) \overset{n}{=} c(n) \qquad \text{(COFE-COMPL)}$$

Definition 9. *The category* **COFE** *consists of COFEs as objects, and non-expansive functions as arrows.*

The function space $T \xrightarrow{\mathsf{ne}} U$ is a COFE if U is a COFE (*i.e.,* the domain T can actually be just an OFE). *SProp* as defined above is complete, *i.e.,* it is a COFE.

Completeness is necessary to take fixed-points.

Theorem 1 (Banach's fixed-point)**.** *Given an inhabited COFE T and a contractive function $f : T \to T$, there exists a unique fixed-point $\mathit{fix}_T f$ such that $f(\mathit{fix}_T f) = \mathit{fix}_T f$. Moreover, this theorem also holds if f is just non-expansive, and f^k is contractive for an arbitrary k.*

Theorem 2 (America and Rutten [1, 3])**.** *Let 1 be the discrete COFE on the unit type: $1 \triangleq \Delta\{()\}$. Given a locally contractive bifunctor $G : \mathbf{COFE}^{op} \times \mathbf{COFE} \to \mathbf{COFE}$, and provided that $G(1, 1)$ is inhabited, then there exists a unique*[1] *COFE T such that $G(T^{op}, T) \cong T$ (*i.e., *the two are isomorphic in* **COFE***).*

[1] Uniqueness is not proven in Coq.

2.3 RA

Definition 10. *A* resource algebra *(RA) is a tuple* $(M, \overline{\mathcal{V}} : M \to \mathit{Prop}, |-| : M \to M^?, (\cdot) : M \times M \to M)$ *satisfying:*

$$\forall a, b, c.\ (a \cdot b) \cdot c = a \cdot (b \cdot c) \qquad \text{(RA-ASSOC)}$$

$$\forall a, b.\ a \cdot b = b \cdot a \qquad \text{(RA-COMM)}$$

$$\forall a.\ |a| \in M \Rightarrow |a| \cdot a = a \qquad \text{(RA-CORE-ID)}$$

$$\forall a.\ |a| \in M \Rightarrow ||a|| = |a| \qquad \text{(RA-CORE-IDEM)}$$

$$\forall a, b.\ |a| \in M \land a \preccurlyeq b \Rightarrow |b| \in M \land |a| \preccurlyeq |b| \qquad \text{(RA-CORE-MONO)}$$

$$\forall a, b.\ \overline{\mathcal{V}}(a \cdot b) \Rightarrow \overline{\mathcal{V}}(a) \qquad \text{(RA-VALID-OP)}$$

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

$$a \preccurlyeq b \triangleq \exists c \in M.\ b = a \cdot c \qquad \text{(RA-INCL)}$$

Here, *Prop* is the set of (meta-level) propositions. Think of `Prop` in Coq or $\mathbb{B}$ in classical mathematics.

RAs are closely related to *Partial Commutative Monoids* (PCMs), with two key differences:

1. The composition operation on RAs is total (as opposed to the partial composition operation of a PCM), but there is a specific subset of *valid* elements that is compatible with the composition operation (RA-VALID-OP). These valid elements are identified by the *validity predicate* $\overline{\mathcal{V}}$.

   This take on partiality is necessary when defining the structure of *higher-order* ghost state, *cameras*, in the next subsection.

2. Instead of a single unit that is an identity to every element, we allow for an arbitrary number of units, via a function $|-|$ assigning to an 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*, defined similarly to that for PCMs (RA-INCL).

   Notice that the codomain of the core is $M^?$, a set that adds a dummy element $\bot$ to M. Thus, the core can be *partial*: not all elements need to have a unit. We use the metavariable $a^?$ to indicate elements of $M^?$. We also lift the composition $(\cdot)$ to $M^?$. Partial cores help us to build interesting composite RAs from smaller primitives.

   Notice also that the core of an RA is a strict generalization of the unit that any PCM must provide, since $|-|$ can always be picked as a constant function.

Definition 11. *It is possible to do a* frame-preserving update *from $a \in M$ to $B \subseteq M$, written $a \rightsquigarrow B$, if*

$$\forall a_{\mathsf{f}}^? \in M^?.\ \overline{\mathcal{V}}(a \cdot a_{\mathsf{f}}^?) \Rightarrow \exists b \in B.\ \overline{\mathcal{V}}(b \cdot a_{\mathsf{f}}^?)$$

We further define $a \rightsquigarrow b \triangleq a \rightsquigarrow \{b\}$.

The proposition $a \rightsquigarrow B$ says that every element $a_{\mathsf{f}}^?$ compatible with a (we also call such elements *frames*), must also be compatible with some $b \in B$. Notice that $a_{\mathsf{f}}^?$ could be $\bot$, so the frame-preserving update can also be applied to elements that have *no* frame. Intuitively, this means that whatever assumptions the rest of the program is making about the state of γ, if these assumptions

are compatible with a, then updating to b will not invalidate any of these assumptions. Since Iris ensures that the global ghost state is valid, this means that we can soundly update the ghost state from a to a non-deterministically picked $b \in B$.

2.4 Cameras

Definition 12. *A* camera *is a tuple* $(M : \mathbf{OFE}, \mathcal{V} : M \xrightarrow{\mathsf{ne}} \mathit{SProp}, |-| : M \xrightarrow{\mathsf{ne}} M^?, (\cdot) : M \times M \xrightarrow{\mathsf{ne}} M)$ *satisfying:*

$$\forall a, b, c.\ (a \cdot b) \cdot c = a \cdot (b \cdot c) \qquad \text{(CAMERA-ASSOC)}$$

$$\forall a, b.\ a \cdot b = b \cdot a \qquad \text{(CAMERA-COMM)}$$

$$\forall a.\ |a| \in M \Rightarrow |a| \cdot a = a \qquad \text{(CAMERA-CORE-ID)}$$

$$\forall a.\ |a| \in M \Rightarrow ||a|| = |a| \qquad \text{(CAMERA-CORE-IDEM)}$$

$$\forall a, b.\ |a| \in M \land a \preccurlyeq b \Rightarrow |b| \in M \land |a| \preccurlyeq |b| \qquad \text{(CAMERA-CORE-MONO)}$$

$$\forall a, b.\ \mathcal{V}(a \cdot b) \subseteq \mathcal{V}(a) \qquad \text{(CAMERA-VALID-OP)}$$

$$\forall n, a, b_1, b_2.\ n \in \mathcal{V}(a) \land a \overset{n}{=} b_1 \cdot b_2 \Rightarrow \exists c_1, c_2.\ a = c_1 \cdot c_2 \land c_1 \overset{n}{=} b_1 \land c_2 \overset{n}{=} b_2 \qquad \text{(CAMERA-EXTEND)}$$

where

$$a \preccurlyeq b \triangleq \exists c.\ b = a \cdot c \qquad \text{(CAMERA-INCL)}$$

$$a \overset{n}{\preccurlyeq} b \triangleq \exists c.\ b \overset{n}{=} a \cdot c \qquad \text{(CAMERA-INCLN)}$$

This is a natural generalization of RAs over OFEs[2]. All operations have to be non-expansive, and the validity predicate $\mathcal{V}$ can now also depend on the step-index. We define the plain $\overline{\mathcal{V}}$ as the "limit" of the step-indexed approximation:

$$\overline{\mathcal{V}}(a) \triangleq \forall n.\ n \in \mathcal{V}(a)$$

The extension axiom (CAMERA-EXTEND). Notice that the existential quantification in this axiom is *constructive*, *i.e.,* it is a sigma type in Coq. The purpose of this axiom is to compute a_1, a_2 completing the following square:

$$\begin{array}{ccc} a & \overset{n}{=} & b \\ \| & & \| \\ a_1 \cdot a_2 & \overset{n}{=} & b_1 \cdot b_2 \end{array}$$

[2]The reader may wonder why on earth we call them "cameras". The reason, which may not be entirely convincing, is that "camera" was originally just used as a comfortable pronunciation of "CMRA", the name used in earlier Iris papers. CMRA was originally supposed to be an acronym for "complete metric resource algebras" (or something like that), but we were never very satisfied with it and thus ended up never spelling it out. To make matters worse, the "complete" part of CMRA is now downright misleading, for whereas previously the carrier of a CMRA was required to be a COFE (complete OFE), we have relaxed that restriction and permit it to be an (incomplete) OFE. For these reasons, we have decided to stick with the name "camera", for purposes of continuity, but to drop any pretense that it stands for something.

where the n-equivalence at the bottom is meant to apply to the pairs of elements, *i.e.,* we demand $a_1 \overset{n}{=} b_1$ and $a_2 \overset{n}{=} b_2$. In other words, extension carries the decomposition of b into b_1 and b_2 over the n-equivalence of a and b, and yields a corresponding decomposition of a into a_1 and a_2. This operation is needed to prove that $\triangleright$ commutes with separating conjunction:

$$\triangleright(P * Q) \Leftrightarrow \triangleright P * \triangleright Q$$

Definition 13. *An element ε of a camera M is called the* unit *of M if it satisfies the following conditions:*

1. *ε is valid:*
   $\forall n.\, n \in \mathcal{V}(\varepsilon)$
2. *ε is a left-identity of the operation:*
   $\forall a \in M.\, \varepsilon \cdot a = a$
3. *ε is its own core:*
   $|\varepsilon| = \varepsilon$

Lemma 1. *If M has a unit ε, then the core $|-|$ is total,* i.e., $\forall a.\, |a| \in M$.

Definition 14. *It is possible to do a* frame-preserving update *from $a \in M$ to $B \subseteq M$, written $a \rightsquigarrow B$, if*

$$\forall n, a_{\mathsf{f}}^{?}.\, n \in \mathcal{V}(a \cdot a_{\mathsf{f}}^{?}) \Rightarrow \exists b \in B.\, n \in \mathcal{V}(b \cdot a_{\mathsf{f}}^{?})$$

We further define $a \rightsquigarrow b \triangleq a \rightsquigarrow \{b\}$.

Note that for RAs, this and the RA-based definition of a frame-preserving update coincide.

Definition 15. *A camera M is* discrete *if it satisfies the following conditions:*

1. *M is a discrete COFE*
2. *$\mathcal{V}$ ignores the step-index:*
   $\forall a \in M.\, 0 \in \mathcal{V}(a) \Rightarrow \forall n.\, n \in \mathcal{V}(a)$

Note that every RA is a discrete camera, by picking the discrete COFE for the equivalence relation. Furthermore, discrete cameras can be turned into RAs by ignoring their COFE structure, as well as the step-index of $\mathcal{V}$.

Definition 16 (Camera homomorphism)**.** *A function $f : M_1 \to M_2$ between two cameras is* a camera homomorphism *if it satisfies the following conditions:*

1. *f is non-expansive*
2. *f commutes with composition:*
   $\forall a_1 \in M_1, a_2 \in M_1.\, f(a_1) \cdot f(a_2) = f(a_1 \cdot a_2)$
3. *f commutes with the core:*
   $\forall a \in M_1.\, |f(a)| = f(|a|)$
4. *f preserves validity:*
   $\forall n, a \in M_1.\, n \in \mathcal{V}(a) \Rightarrow n \in \mathcal{V}(f(a))$

Definition 17. *The category* **Camera** *consists of cameras as objects, and camera homomorphisms as arrows.*

Note that every object/arrow in **Camera** is also an object/arrow of **OFE**. The notion of a locally non-expansive (or contractive) bifunctor naturally generalizes to bifunctors between these categories.

3 OFE and COFE Constructions

3.1 Trivial Pointwise Lifting

The (C)OFE structure on many types can be easily obtained by pointwise lifting of the structure of the components. This is what we do for option $T^?$, product $(M_i)_{i\in I}$ (with I some finite index set), sum $T + T'$ and finite partial functions $K \xrightarrow{\text{fin}} M$ (with K infinite countable).

3.2 Next (Type-Level Later)

Given an OFE T, we define $\blacktriangleright T$ as follows (using a datatype-like notation to define the type):

$$\blacktriangleright T \triangleq \mathsf{next}(x : T)$$
$$\mathsf{next}(x) \overset{n}{=} \mathsf{next}(y) \triangleq n = 0 \vee x \overset{n-1}{=} y$$

Note that in the definition of the carrier $\blacktriangleright T$, next is a constructor (like the constructors in Coq), *i.e.,* this is short for $\{\mathsf{next}(x) \mid x \in T\}$.

$\blacktriangleright(-)$ is a locally *contractive* functor from **OFE** to **OFE**.

3.3 Uniform Predicates

Given a camera M, we define the COFE $\mathit{UPred}(M)$ of *uniform predicates* over M as follows:

$$M \xrightarrow{\mathsf{mon},\mathsf{ne}} \mathit{SProp} \triangleq \left\{ \Phi : M \xrightarrow{\mathsf{ne}} \mathit{SProp} \;\middle|\; \forall n, a, b.\, a \overset{n}{\preccurlyeq} b \Rightarrow \Phi(a) \overset{n}{\subseteq} \Phi(b) \right\}$$
$$\mathit{UPred}(M) \triangleq M \xrightarrow{\mathsf{mon},\mathsf{ne}} \mathit{SProp}/_{\equiv}$$
$$\Phi \equiv \Psi \triangleq \forall m, a.\, m \in \mathcal{V}(a) \Rightarrow (m \in \Phi(a) \iff m \in \Psi(a))$$
$$\Phi \overset{n}{\equiv} \Psi \triangleq \forall m \le n, a.\, m \in \mathcal{V}(a) \Rightarrow (m \in \Phi(a) \iff m \in \Psi(a))$$

You can think of uniform predicates as monotone, step-indexed predicates over a camera that "ignore" invalid elements (as defined by the quotient). Note that we will not explicitly deal with equivalence classes below but instead just pick an arbitrary element of the class. There is an implicit proof obligation for every operation acting on *UPred* to show that the concrete representative does not matter. This is all explicit in Coq (where we use a setoid to represent the quotient); we do not spell it out on paper.

$\mathit{UPred}(-)$ is a locally non-expansive functor from **Camera** to **COFE**.

It is worth noting that the above quotient admits canonical representatives. More precisely, one can show that every equivalence class contains exactly one element P_0 such that:

$$\forall n, a.\, (\mathcal{V}(a) \overset{n}{\subseteq} P_0(a)) \Rightarrow n \in P_0(a) \qquad \text{(UPred-canonical)}$$

Intuitively, this says that P_0 trivially holds whenever the resource is invalid. Starting from any element P, one can find this canonical representative by choosing $P_0(a) := \{n \mid n \in \mathcal{V}(a) \Rightarrow n \in P(a)\}$.

Hence, as an alternative definition of *UPred*, we could use the set of canonical representatives. This alternative definition would save us from using a quotient. However, the definitions of the various connectives would get more complicated, because we have to make sure they all verify UPred-canonical, which sometimes requires some adjustments. We would moreover need to prove one more property for every logical connective.

4 RA and Camera Constructions

4.1 Product

Given a family $(M_i)_{i \in I}$ of cameras (I finite), we construct a camera for the product $\prod_{i \in I} M_i$ by lifting everything pointwise.

Frame-preserving updates on the M_i lift to the product:

$$\text{PROD-UPDATE}\quad \frac{a \rightsquigarrow_{M_i} B}{f[i \leftarrow a] \rightsquigarrow \{f[i \leftarrow b] \mid b \in B\}}$$

4.2 Sum

The *sum camera* $M_1 +_\lightning M_2$ for any cameras M_1 and M_2 is defined as (again, we use a datatype-like notation):

$$\begin{aligned}
M_1 +_\lightning M_2 &\triangleq \mathsf{inl}(a_1 : M_1) \mid \mathsf{inr}(a_2 : M_2) \mid \lightning \\
\mathcal{V}(\lightning) &\triangleq \emptyset \\
\mathcal{V}(\mathsf{inl}(a)) &\triangleq \mathcal{V}_1(a) \\
\mathsf{inl}(a_1) \cdot \mathsf{inl}(b_1) &\triangleq \mathsf{inl}(a_1 \cdot b_1) \\
|\mathsf{inl}(a_1)| &\triangleq \begin{cases} \bot & \text{if } |a_1| = \bot \\ \mathsf{inl}(|a_1|) & \text{otherwise} \end{cases}
\end{aligned}$$

Above, $\mathcal{V}_1$ refers to the validity of M_1. The validity, composition and core for inr are defined symmetrically. The remaining cases of the composition and core are all $\lightning$.

Notice that we added the artificial "invalid" (or "undefined") element $\lightning$ to this camera just in order to make certain compositions of elements (in this case, inl and inr) invalid.

The step-indexed equivalence is inductively defined as follows:

$$\frac{x \overset{n}{=} y}{\mathsf{inl}(x) \overset{n}{=} \mathsf{inl}(y)} \qquad \frac{x \overset{n}{=} y}{\mathsf{inr}(x) \overset{n}{=} \mathsf{inr}(y)} \qquad \lightning \overset{n}{=} \lightning$$

We obtain the following frame-preserving updates, as well as their symmetric counterparts:

$$\text{SUM-UPDATE}\quad \frac{a \rightsquigarrow_{M_1} B}{\mathsf{inl}(a) \rightsquigarrow \{\mathsf{inl}(b) \mid b \in B\}} \qquad \text{SUM-SWAP}\quad \frac{\forall a_\mathsf{f} \in M, n.\ n \notin \mathcal{V}(a \cdot a_\mathsf{f}) \qquad \overline{\mathcal{V}}(b)}{\mathsf{inl}(a) \rightsquigarrow \mathsf{inr}(b)}$$

Crucially, the second rule allows us to *swap* the "side" of the sum that the camera is on if $\mathcal{V}$ has *no possible frame*.

4.3 Option

The definition of the camera/RA axioms already lifted the composition operation on M to one on $M^?$. We can easily extend this to a full camera by defining a suitable core, namely

$$\begin{aligned}
|\bot| &\triangleq \bot \\
|a^?| &\triangleq |a| \qquad \text{If } a^? \neq \bot
\end{aligned}$$

Notice that this core is total, as the result always lies in $M^?$ (rather than in $M^{?^?}$).

4.4 Finite Partial Functions

Given some infinite countable K and some camera M, the set of finite partial functions $K \xrightarrow{\mathsf{fin}} M$ is equipped with a camera structure by lifting everything pointwise.

We obtain the following frame-preserving updates:

FPFN-ALLOC-STRONG
$$\frac{G \subseteq K \text{ infinite} \qquad \overline{\mathcal{V}}(a)}{\emptyset \rightsquigarrow \{[i \leftarrow a] \mid i \in G\}}$$

FPFN-ALLOC
$$\frac{\overline{\mathcal{V}}(a)}{\emptyset \rightsquigarrow \{[i \leftarrow a] \mid i \in K\}}$$

FPFN-UPDATE
$$\frac{a \rightsquigarrow_M B}{f[i \leftarrow a]] \rightsquigarrow \{f[i \leftarrow b] \mid b \in B\}}$$

Above, $\overline{\mathcal{V}}$ refers to the (full) validity of M.

$K \xrightarrow{\mathsf{fin}} (-)$ is a locally non-expansive functor from **Camera** to **Camera**.

4.5 Agreement

Given some OFE T, we define the camera $Ag(T)$ as follows:

$$\begin{aligned} Ag(T) &\triangleq \left\{a \in \wp^{\mathsf{fin}}(T) \;\middle|\; a \neq \emptyset\right\} / \sim \\ a \overset{n}{=} b &\triangleq (\forall x \in a.\, \exists y \in b.\, x \overset{n}{=} y) \wedge (\forall y \in b.\, \exists x \in a.\, x \overset{n}{=} y) \\ &\text{where } a \sim b \triangleq \forall n.\, a \overset{n}{=} b \end{aligned}$$

$$\begin{aligned} \mathcal{V}(a) &\triangleq \left\{n \;\middle|\; \forall x, y \in a.\, x \overset{n}{=} y\right\} \\ |a| &\triangleq a \\ a \cdot b &\triangleq a \cup b \end{aligned}$$

$Ag(-)$ is a locally non-expansive functor from **OFE** to **Camera**.

We define a non-expansive injection ag into $Ag(T)$ as follows:

$$\mathsf{ag}(x) \triangleq \{x\}$$

There are no interesting frame-preserving updates for $Ag(T)$, but we can show the following:

AG-VAL
$$\overline{\mathcal{V}}(\mathsf{ag}(x))$$

AG-DUP
$$\mathsf{ag}(x) = \mathsf{ag}(x) \cdot \mathsf{ag}(x)$$

AG-AGREE
$$n \in \mathcal{V}(\mathsf{ag}(x) \cdot \mathsf{ag}(y)) \Rightarrow x \overset{n}{=} y$$

4.6 Exclusive Camera

Given an OFE T, we define a camera $Ex(T)$ such that at most one $x \in T$ can be owned:

$$\begin{aligned} Ex(T) &\triangleq \mathsf{ex}(T) \mid \lightning \\ \mathcal{V}(a) &\triangleq \left\{n \;\middle|\; a \overset{n}{\neq} \lightning\right\} \end{aligned}$$

All cases of composition go to $\lightning$.

$$|\mathsf{ex}(x)| \triangleq \bot \qquad\qquad |\lightning| \triangleq \lightning$$

Remember that $\bot$ is the "dummy" element in $M^?$ indicating (in this case) that $\mathsf{ex}(x)$ has no core.

The step-indexed equivalence is inductively defined as follows:

$$\frac{x \overset{n}{=} y}{\mathsf{ex}(x) \overset{n}{=} \mathsf{ex}(y)} \qquad\qquad \lightning \overset{n}{=} \lightning$$

$Ex(-)$ is a locally non-expansive functor from **OFE** to **Camera**.

We obtain the following frame-preserving update:

$$\textsc{ex-update}$$
$$\mathsf{ex}(x) \rightsquigarrow \mathsf{ex}(y)$$

4.7 Fractions

We define an RA structure on the rational numbers in $(0, 1]$ as follows:

$$\begin{aligned}
\mathit{Frac} &\triangleq \mathsf{frac}(\mathbb{Q} \cap (0,1]) \mid \lightning \\
\overline{\mathcal{V}}(a) &\triangleq a \neq \lightning \\
\mathsf{frac}(q_1) \cdot \mathsf{frac}(q_2) &\triangleq \mathsf{frac}(q_1 + q_2) \quad \text{if } q_1 + q_2 \leq 1 \\
|\mathsf{frac}(x)| &\triangleq \bot \\
|\lightning| &\triangleq \lightning
\end{aligned}$$

All remaining cases of composition go to $\lightning$. Frequently, we will write just x instead of $\mathsf{frac}(x)$.

The most important property of this RA is that 1 has no frame. This is useful in combination with SUM-SWAP, and also when used with pairs:

$$\textsc{pair-frac-change}$$
$$(1, a) \rightsquigarrow (1, b)$$

4.8 Authoritative

Given a camera M, we construct $\mathit{Auth}(M)$ modeling someone owning an *authoritative* element a of M, and others potentially owning fragments $b \preccurlyeq a$ of a. We assume that M has a unit ε, and hence its core is total. (If M is an exclusive monoid, the construction is very similar to a half-ownership monoid with two asymmetric halves.)

$$\begin{aligned}
\mathit{Auth}(M) &\triangleq \mathit{Ex}(M)^? \times M \\
\mathcal{V}((x,b)) &\triangleq \{n \mid (x = \bot \wedge n \in \mathcal{V}(b)) \vee (\exists a.\, x = \mathsf{ex}(a) \wedge b \preccurlyeq_n a \wedge n \in \mathcal{V}(a))\} \\
(x_1, b_1) \cdot (x_2, b_2) &\triangleq (x_1 \cdot x_2, b_2 \cdot b_2) \\
|(x,b)| &\triangleq (\bot, |b|) \\
(x_1, b_1) \overset{n}{=} (x_2, b_2) &\triangleq x_1 \overset{n}{=} x_2 \wedge b_1 \overset{n}{=} b_2
\end{aligned}$$

Note that $(\bot, \varepsilon)$ is the unit and asserts no ownership whatsoever, but $(\mathsf{ex}(\varepsilon), \varepsilon)$ asserts that the authoritative element is ε.

Let $a, b \in M$. We write $\bullet a$ for full ownership $(\mathsf{ex}(a), \varepsilon)$ and $\circ b$ for fragmental ownership $(\bot, b)$ and $\bullet a, \circ b$ for combined ownership $(\mathsf{ex}(a), b)$.

The frame-preserving update involves the notion of a *local update*:

Definition 18. *It is possible to do a* local update *from a_1 and b_1 to a_2 and b_2, written $(a_1, b_1) \rightsquigarrow_{\mathsf{L}} (a_2, b_2)$, if*

$$\forall n, a_{\mathsf{f}}^?.\ n \in \mathcal{V}(a_1) \land a_1 \overset{n}{=} b_1 \cdot a_{\mathsf{f}}^? \Rightarrow n \in \mathcal{V}(a_2) \land a_2 \overset{n}{=} b_2 \cdot a_{\mathsf{f}}^?$$

In other words, the idea is that for every possible frame $a_{\mathsf{f}}^?$ completing b_1 to a_1, the same frame also completes b_2 to a_2.

We then obtain

$$\frac{(a_1, b_1) \rightsquigarrow_{\mathsf{L}} (a_2, b_2)}{\bullet a_1, \circ b_1 \rightsquigarrow \bullet a_2, \circ b_2}\ \textsc{auth-update}$$

4.9 STS with Tokens

Given a state-transition system (STS, *i.e.,* a directed graph) $(\mathcal{S}, \to \subseteq \mathcal{S} \times \mathcal{S})$, a set of tokens $\mathcal{T}$, and a labeling $\mathcal{L} : \mathcal{S} \to \wp(\mathcal{T})$ of *protocol-owned* tokens for each state, we construct an RA modeling an authoritative current state and permitting transitions given a *bound* on the current state and a set of *locally-owned* tokens.

The construction follows the idea of STSs as described in CaReSL [9]. We first lift the transition relation to $\mathcal{S} \times \wp(\mathcal{T})$ (implementing a *law of token conservation*) and define a stepping relation for the *frame* of a given token set:

$$(s, T) \to (s', T') \triangleq s \to s' \land \mathcal{L}(s) \uplus T = \mathcal{L}(s') \uplus T'$$
$$s \xrightarrow{T} s' \triangleq \exists T_1, T_2.\ T_1 \# \mathcal{L}(s) \cup T \land (s, T_1) \to (s', T_2)$$

We further define *closed* sets of states (given a particular set of tokens) as well as the *closure* of a set:

$$\mathsf{closed}(S, T) \triangleq \forall s \in S.\ \mathcal{L}(s) \# T \land \left(\forall s'.\ s \xrightarrow{T} s' \Rightarrow s' \in S\right)$$
$$\uparrow(S, T) \triangleq \left\{ s' \in \mathcal{S} \,\middle|\, \exists s \in S.\ s \xrightarrow{T}{}^{*}\ s' \right\}$$

The STS RA is defined as follows

$$M \triangleq \mathsf{auth}(s : \mathcal{S}, T : \wp(\mathcal{T}) \mid \mathcal{L}(s) \# T) \mid \mathsf{frag}(S : \wp(\mathcal{S}), T : \wp(\mathcal{T}) \mid \mathsf{closed}(S, T) \land S \neq \emptyset) \mid \lightning$$
$$\overline{\mathcal{V}}(a) \triangleq a \neq \lightning$$
$$\mathsf{frag}(S_1, T_1) \cdot \mathsf{frag}(S_2, T_2) \triangleq \mathsf{frag}(S_1 \cap S_2, T_1 \cup T_2) \qquad \text{if } T_1 \# T_2 \text{ and } S_1 \cap S_2 \neq \emptyset$$
$$\mathsf{frag}(S, T) \cdot \mathsf{auth}(s, T') \triangleq \mathsf{auth}(s, T') \cdot \mathsf{frag}(S, T) \triangleq \mathsf{auth}(s, T \cup T') \qquad \text{if } T \# T' \text{ and } s \in S$$
$$|\mathsf{frag}(S, T)| \triangleq \mathsf{frag}(\uparrow(S, \emptyset), \emptyset)$$
$$|\mathsf{auth}(s, T)| \triangleq \mathsf{frag}(\uparrow(\{s\}, \emptyset), \emptyset)$$

The remaining cases are all $\lightning$.

We will need the following frame-preserving update:

$$\text{STS-STEP}\ \frac{(s,T) \to^* (s',T')}{\mathsf{auth}(s,T) \rightsquigarrow \mathsf{auth}(s',T')} \qquad \text{STS-WEAKEN}\ \frac{\mathsf{closed}(S_2,T_2) \qquad S_1 \subseteq S_2 \qquad T_2 \subseteq T_1}{\mathsf{frag}(S_1,T_1) \rightsquigarrow \mathsf{frag}(S_2,T_2)}$$

The core is not a homomorphism. The core of the STS construction is only satisfying the RA axioms because we are *not* demanding the core to be a homomorphism—all we demand is for the core to be monotone with respect the RA-INCL.

In other words, the following does *not* hold for the STS core as defined above:

$$|a| \cdot |b| = |a \cdot b|$$

To see why, consider the following STS:

Now consider the following two elements of the STS RA:

$$a \triangleq \mathsf{frag}(\{\mathsf{s}_1, \mathsf{s}_2\}, \{\mathsf{t}_1\}) \qquad b \triangleq \mathsf{frag}(\{\mathsf{s}_1, \mathsf{s}_3\}, \{\mathsf{t}_2\})$$

We have:

$$a \cdot b = \mathsf{frag}(\{\mathsf{s}_1\}, \{\mathsf{t}_1, \mathsf{t}_2\}) \qquad |a| = \mathsf{frag}(\{\mathsf{s}_1, \mathsf{s}_2, \mathsf{s}_4\}, \emptyset) \qquad |b| = \mathsf{frag}(\{\mathsf{s}_1, \mathsf{s}_3, \mathsf{s}_4\}, \emptyset)$$

$$|a| \cdot |b| = \mathsf{frag}(\{\mathsf{s}_1, \mathsf{s}_4\}, \emptyset) \neq |a \cdot b| = \mathsf{frag}(\{\mathsf{s}_1\}, \emptyset)$$

5 Base Logic

The base logic is parameterized by an arbitrary camera M having a unit ε. By Lemma 1, this means that the core of M is a total function, so we will treat it as such in the following. This defines the structure of resources that can be owned.

As usual for higher-order logics, you can furthermore pick a *signature* $\mathcal{S} = (\mathcal{T}, \mathcal{F}, \mathcal{A})$ to add more types, symbols and axioms to the language. You have to make sure that $\mathcal{T}$ includes the base types:

$$\mathcal{T} \supseteq \{\mathsf{M}, \mathit{iProp}\}$$

Elements of $\mathcal{T}$ are ranged over by T.

Each function symbol in $\mathcal{F}$ has an associated *arity* comprising a natural number n and an ordered list of $n+1$ types τ (the grammar of τ is defined below, and depends only on $\mathcal{T}$). We write

$$F : \tau_1, \ldots, \tau_n \to \tau_{n+1} \in \mathcal{F}$$

to express that F is a function symbol with the indicated arity.

Furthermore, $\mathcal{A}$ is a set of *axioms*, that is, terms t of type *iProp*. Again, the grammar of terms and their typing rules are defined below, and depends only on $\mathcal{T}$ and $\mathcal{F}$, not on $\mathcal{A}$. Elements of $\mathcal{A}$ are ranged over by A.

5.1 Grammar

Syntax. Iris syntax is built up from a signature $\mathcal{S}$ and a countably infinite set *Var* of variables (ranged over by metavariables x, y, z). Below, a ranges over M and i ranges over $\{1, 2\}$.

$$\begin{aligned}
\tau &::= T \mid 0 \mid 1 \mid \tau + \tau \mid \tau \times \tau \mid \tau \to \tau \\
t, P, \Phi &::= x \mid F(t_1, \ldots, t_n) \mid \mathsf{abort}\ t \mid () \mid (t, t) \mid \pi_i\ t \mid \lambda x : \tau.\, t \mid t(t) \mid \\
&\quad \mathsf{inj}_i\ t \mid \mathsf{match}\ t\ \mathsf{with}\ \mathsf{inj}_1\ x.\, t \mid \mathsf{inj}_2\ x.\, t\ \mathsf{end} \mid a \mid |t| \mid t \cdot t \mid \\
&\quad \mathsf{False} \mid \mathsf{True} \mid t =_\tau t \mid P \Rightarrow P \mid P \wedge P \mid P \vee P \mid P * P \mid P \mathrel{-\!\!*} P \mid \\
&\quad \mu x : \tau.\, t \mid \exists x : \tau.\, P \mid \forall x : \tau.\, P \mid \mathsf{Own}\,(t) \mid \mathcal{V}(t) \mid \Box P \mid \blacksquare P \mid \triangleright P \mid \dot{\Rrightarrow} P
\end{aligned}$$

Well-typedness forces recursive definitions to be *guarded*: In $\mu x.\, t$, the variable x can only appear under the later $\triangleright$ modality. Furthermore, the type of the definition must be *complete*. The type *iProp* is complete, and if τ is complete, then so is $\tau' \to \tau$.

Note that the modalities $\dot{\Rrightarrow}$, $\Box$, $\blacksquare$ and $\triangleright$ bind more tightly than $*$, $\mathrel{-\!\!*}$, $\wedge$, $\vee$, and $\Rightarrow$.

Variable conventions. We assume that, if a term occurs multiple times in a rule, its free variables are exactly those binders which are available at every occurrence.

5.2 Types

Iris terms are simply-typed. The judgment $\Gamma \vdash t : \tau$ expresses that, in variable context Γ, the term t has type τ.

A variable context, $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$, declares a list of variables and their types. In writing $\Gamma, x : \tau$, we presuppose that x is not already declared in Γ.

Well-typed terms $\boxed{\Gamma \vdash_{\mathcal{S}} t : \tau}$

$$x : \tau \vdash x : \tau \qquad \frac{\Gamma \vdash t : \tau}{\Gamma, x : \tau' \vdash t : \tau} \qquad \frac{\Gamma, x : \tau', y : \tau' \vdash t : \tau}{\Gamma, x : \tau' \vdash t[x/y] : \tau} \qquad \frac{\Gamma_1, x : \tau', y : \tau'', \Gamma_2 \vdash t : \tau}{\Gamma_1, x : \tau'', y : \tau', \Gamma_2 \vdash t[y/x, x/y] : \tau}$$

$$\frac{\Gamma \vdash t_1 : \tau_1 \qquad \cdots \qquad \Gamma \vdash t_n : \tau_n \qquad F : \tau_1, \ldots, \tau_n \to \tau_{n+1} \in \mathcal{F}}{\Gamma \vdash F(t_1, \ldots, t_n) : \tau_{n+1}} \qquad \frac{\Gamma \vdash t : 0}{\Gamma \vdash \mathsf{abort}\ t : \tau} \qquad \Gamma \vdash () : 1$$

$$\frac{\Gamma \vdash t : \tau_1 \qquad \Gamma \vdash u : \tau_2}{\Gamma \vdash (t, u) : \tau_1 \times \tau_2} \qquad \frac{\Gamma \vdash t : \tau_1 \times \tau_2 \qquad i \in \{1, 2\}}{\Gamma \vdash \pi_i\, t : \tau_i} \qquad \frac{\Gamma \vdash t : \tau_i \qquad i \in \{1, 2\}}{\Gamma \vdash \mathsf{inj}_i\ t : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash t : \tau_1 + \tau_2 \qquad \Gamma, x : \tau_1 \vdash t_1 : \tau \qquad \Gamma, y : \tau_2 \vdash t_2 : \tau}{\Gamma \vdash \mathsf{match}\ t\ \mathsf{with}\ \mathsf{inj}_1\ x.\, t_1 \mid \mathsf{inj}_2\ y.\, t_2\ \mathsf{end} : \tau} \qquad \frac{\Gamma, x : \tau \vdash t : \tau'}{\Gamma \vdash \lambda x.\, t : \tau \to \tau'}$$

$$\frac{\Gamma \vdash t : \tau \to \tau' \qquad u : \tau}{\Gamma \vdash t(u) : \tau'} \qquad \Gamma \vdash a : \mathsf{M} \qquad \frac{\Gamma \vdash a : \mathsf{M}}{\Gamma \vdash |a| : \mathsf{M}} \qquad \frac{\Gamma \vdash a : \mathsf{M} \qquad \Gamma \vdash b : \mathsf{M}}{\Gamma \vdash a \cdot b : \mathsf{M}}$$

$$\Gamma \vdash \mathsf{False} : \mathit{iProp} \qquad \Gamma \vdash \mathsf{True} : \mathit{iProp} \qquad \frac{\Gamma \vdash t : \tau \qquad \Gamma \vdash u : \tau}{\Gamma \vdash t =_\tau u : \mathit{iProp}} \qquad \frac{\Gamma \vdash P : \mathit{iProp} \qquad \Gamma \vdash Q : \mathit{iProp}}{\Gamma \vdash P \Rightarrow Q : \mathit{iProp}}$$

$$\frac{\Gamma \vdash P : \mathit{iProp} \qquad \Gamma \vdash Q : \mathit{iProp}}{\Gamma \vdash P \wedge Q : \mathit{iProp}} \qquad \frac{\Gamma \vdash P : \mathit{iProp} \qquad \Gamma \vdash Q : \mathit{iProp}}{\Gamma \vdash P \vee Q : \mathit{iProp}}$$

$$\frac{\Gamma \vdash P : \mathit{iProp} \qquad \Gamma \vdash Q : \mathit{iProp}}{\Gamma \vdash P * Q : \mathit{iProp}} \qquad \frac{\Gamma \vdash P : \mathit{iProp} \qquad \Gamma \vdash Q : \mathit{iProp}}{\Gamma \vdash P \mathrel{-\!\!*} Q : \mathit{iProp}}$$

$$\frac{\Gamma, x : \tau \vdash t : \tau \qquad x \text{ is guarded in } t \qquad \tau \text{ is complete and inhabited}}{\Gamma \vdash \mu x : \tau.\, t : \tau} \qquad \frac{\Gamma, x : \tau \vdash P : \mathit{iProp}}{\Gamma \vdash \exists x : \tau.\, P : \mathit{iProp}}$$

$$\frac{\Gamma, x : \tau \vdash P : \mathit{iProp}}{\Gamma \vdash \forall x : \tau.\, P : \mathit{iProp}} \qquad \frac{\Gamma \vdash a : \mathsf{M}}{\Gamma \vdash \mathsf{Own}\,(a) : \mathit{iProp}} \qquad \frac{\Gamma \vdash a : \tau \qquad \tau \text{ is a camera}}{\Gamma \vdash \mathcal{V}(a) : \mathit{iProp}} \qquad \frac{\Gamma \vdash P : \mathit{iProp}}{\Gamma \vdash \Box\, P : \mathit{iProp}}$$

$$\frac{\Gamma \vdash P : \mathit{iProp}}{\Gamma \vdash \blacksquare\, P : \mathit{iProp}} \qquad \frac{\Gamma \vdash P : \mathit{iProp}}{\Gamma \vdash \triangleright\, P : \mathit{iProp}} \qquad \frac{\Gamma \vdash P : \mathit{iProp}}{\Gamma \vdash \dot{\Rightarrow} P : \mathit{iProp}}$$

5.3 Proof Rules

The judgment $\Gamma \mid P \vdash Q$ says that with free variables Γ, proposition Q holds whenever assumption P holds. Most of the rules will entirely omit the variable contexts Γ. In this case, we assume the same arbitrary context is used for every constituent of the rules. Axioms $\Gamma \mid P \dashv\vdash Q$ indicate that both $\Gamma \mid P \vdash Q$ and $\Gamma \mid Q \vdash P$ are proof rules of the logic.

$\boxed{\Gamma \mid P \vdash Q}$

Laws of intuitionistic higher-order logic with equality. This is entirely standard.

$$\textsc{Asm}\ \frac{}{P \vdash P} \qquad \textsc{Cut}\ \frac{P \vdash Q \qquad Q \vdash R}{P \vdash R} \qquad \textsc{Eq}\ \frac{\Gamma, x : \tau \vdash Q : iProp \qquad \Gamma \mid P \vdash Q[t/x] \qquad \Gamma \mid P \vdash t =_\tau t'}{\Gamma \mid P \vdash Q[t'/x]}$$

$$\textsc{Refl}\ \frac{}{\mathsf{True} \vdash t =_\tau t} \qquad \bot\text{E}\ \frac{}{\mathsf{False} \vdash P} \qquad \top\text{I}\ \frac{}{P \vdash \mathsf{True}} \qquad \wedge\text{I}\ \frac{P \vdash Q \qquad P \vdash R}{P \vdash Q \wedge R} \qquad \wedge\text{EL}\ \frac{P \vdash Q \wedge R}{P \vdash Q} \qquad \wedge\text{ER}\ \frac{P \vdash Q \wedge R}{P \vdash R}$$

$$\vee\text{IL}\ \frac{P \vdash Q}{P \vdash Q \vee R} \qquad \vee\text{IR}\ \frac{P \vdash R}{P \vdash Q \vee R} \qquad \vee\text{E}\ \frac{P \vdash R \qquad Q \vdash R}{P \vee Q \vdash R} \qquad \Rightarrow\text{I}\ \frac{P \wedge Q \vdash R}{P \vdash Q \Rightarrow R} \qquad \Rightarrow\text{E}\ \frac{P \vdash Q \Rightarrow R \qquad P \vdash Q}{P \vdash R}$$

$$\forall\text{I}\ \frac{\Gamma, x : \tau \mid P \vdash Q}{\Gamma \mid P \vdash \forall x : \tau.\, Q} \qquad \forall\text{E}\ \frac{\Gamma \mid P \vdash \forall x : \tau.\, Q \qquad \Gamma \vdash t : \tau}{\Gamma \mid P \vdash Q[t/x]}$$

$$\exists\text{I}\ \frac{\Gamma \mid P \vdash Q[t/x] \qquad \Gamma \vdash t : \tau}{\Gamma \mid P \vdash \exists x : \tau. Q} \qquad \exists\text{E}\ \frac{\Gamma, x : \tau \mid P \vdash Q}{\Gamma \mid \exists x : \tau.\, P \vdash Q}$$

Furthermore, we have the usual η and β laws for projections, abort, sum elimination, λ and μ.

Laws of (affine) bunched implications.

$$\begin{array}{c} \mathsf{True} * P \dashv\vdash P \\ P * Q \ \vdash\ Q * P \\ (P * Q) * R \ \vdash\ P * (Q * R) \end{array} \qquad *\text{-MONO}\ \frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * P_2 \vdash Q_1 * Q_2} \qquad -\!\!*\text{I-E}\ \frac{P * Q \vdash R}{P \vdash Q \mathrel{-\!\!*} R}$$

Laws for the plainness modality.

$$\blacksquare\text{-MONO}\ \frac{P \vdash Q}{\blacksquare P \vdash \blacksquare Q} \qquad \blacksquare\text{-E}\ \blacksquare P \vdash \square P \qquad \begin{array}{c} (\blacksquare P \Rightarrow \blacksquare Q) \vdash \blacksquare(\blacksquare P \Rightarrow Q) \\ \blacksquare((P \Rightarrow Q) \wedge (Q \Rightarrow P)) \vdash P =_{iProp} Q \end{array} \qquad \begin{array}{c} \blacksquare P \vdash \blacksquare\blacksquare P \\ \forall x.\ \blacksquare P \vdash \blacksquare \forall x.\, P \\ \blacksquare \exists x.\, P \vdash \exists x.\ \blacksquare P \end{array}$$

Laws for the persistence modality.

$$\square\text{-MONO}\ \frac{P \vdash Q}{\square P \vdash \square Q} \qquad \square\text{-E}\ \square P \vdash P \qquad \begin{array}{c} (\blacksquare P \Rightarrow \square Q) \vdash \square(\blacksquare P \Rightarrow Q) \\ \square P \wedge Q \vdash \square P * Q \end{array} \qquad \begin{array}{c} \square P \vdash \square\square P \\ \forall x.\ \square P \vdash \square \forall x.\, P \\ \square \exists x.\, P \vdash \exists x.\ \square P \end{array}$$

Laws for the later modality.

$$\triangleright\text{-MONO}\ \frac{P \vdash Q}{\triangleright P \vdash \triangleright Q} \qquad \triangleright\text{-I}\ P \vdash \triangleright P \qquad \begin{array}{l} \forall x.\ \triangleright P \vdash \triangleright \forall x.\, P \\ \triangleright \exists x.\, P \vdash \triangleright \mathsf{False} \vee \exists x.\ \triangleright P \\ \qquad \triangleright P \vdash \triangleright \mathsf{False} \vee (\triangleright \mathsf{False} \Rightarrow P) \end{array} \qquad \begin{array}{c} \triangleright (P * Q) \dashv\vdash \triangleright P * \triangleright Q \\ \square \triangleright P \dashv\vdash \triangleright \square P \\ \blacksquare \triangleright P \dashv\vdash \triangleright \blacksquare P \end{array}$$

Laws for resources and validity.

$$\begin{aligned}
\mathsf{Own}\,(a) * \mathsf{Own}\,(b) &\dashv\vdash \mathsf{Own}\,(a \cdot b) \\
\forall x.\, \mathsf{Own}\,(f(x)) &\vdash \exists a.\, \mathsf{Own}\,(a) \wedge (\forall x.\, \exists b.\, a = f(x) \cdot b) \\
\mathsf{Own}\,(a) &\vdash \Box\, \mathsf{Own}\,(|a|) \\
\mathsf{True} &\vdash \mathsf{Own}\,(\varepsilon) \\
\triangleright\, \mathsf{Own}\,(a) &\vdash \exists b.\, \mathsf{Own}\,(b) \wedge \triangleright(a = b)
\end{aligned}
\qquad
\begin{aligned}
\mathsf{Own}\,(a) &\vdash \mathcal{V}(a) \\
\mathcal{V}(a \cdot b) &\vdash \mathcal{V}(a) \\
\mathcal{V}(a) &\vdash \blacksquare\, \mathcal{V}(a)
\end{aligned}$$

Laws for the basic update modality.

UPD-MONO
$$\frac{P \vdash Q}{\dot{\Rightarrow} P \vdash \dot{\Rightarrow} Q}$$

UPD-I
$$P \vdash \dot{\Rightarrow} P$$

UPD-TRANS
$$\dot{\Rightarrow} \dot{\Rightarrow} P \vdash \dot{\Rightarrow} P$$

UPD-FRAME
$$Q * \dot{\Rightarrow} P \vdash \dot{\Rightarrow}(Q * P)$$

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

UPD-PLAINLY
$$\dot{\Rightarrow} \blacksquare P \vdash P$$

The premise in UPD-UPDATE is a *meta-level* side-condition that has to be proven about a and B.

5.4 Consistency

The consistency statement of the logic reads as follows: For any n, we have

$$\neg(\mathsf{True} \vdash (\triangleright)^n\, \mathsf{False})$$

where $(\triangleright)^n$ is short for $\triangleright$ being nested n times.

The reason we want a stronger consistency than the usual $\neg(\mathsf{True} \vdash \mathsf{False})$ is our modalities: it should be impossible to derive a contradiction below the modalities. For $\Box$ and $\blacksquare$, this follows from the elimination rules. For updates, we use the fact that $\dot{\Rightarrow} \mathsf{False} \vdash \dot{\Rightarrow} \blacksquare\, \mathsf{False} \vdash \mathsf{False}$. However, there is no elimination rule for $\triangleright$, so we declare that it is impossible to derive a contradiction below any number of laters.

6 Model and Semantics

The semantics closely follows the ideas laid out in [2].

Semantic domains. The semantic domains are interpreted as follows:

$$\begin{array}{lll} [\![\mathit{iProp}]\!] \triangleq \mathit{UPred}(M) & & [\![\tau + \tau']\!] \triangleq [\![\tau]\!] + [\![\tau']\!] \\ [\![\mathsf{M}]\!] \triangleq M & & [\![\tau \times \tau']\!] \triangleq [\![\tau]\!] \times [\![\tau']\!] \\ [\![0]\!] \triangleq \Delta\emptyset & & [\![\tau \to \tau']\!] \triangleq [\![\tau]\!] \xrightarrow{\mathsf{ne}} [\![\tau']\!] \\ [\![1]\!] \triangleq \Delta\{()\} & & \end{array}$$

For the remaining base types τ defined by the signature $\mathcal{S}$, we pick an object X_τ in **OFE** and define

$$[\![\tau]\!] \triangleq X_\tau$$

For each function symbol $F : \tau_1, \ldots, \tau_n \to \tau_{n+1} \in \mathcal{F}$, we pick a function $[\![F]\!] : [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!] \xrightarrow{\mathsf{ne}} [\![\tau_{n+1}]\!]$.

Interpretation of propositions. $\boxed{[\![\Gamma \vdash t : \mathit{iProp}]\!] : [\![\Gamma]\!] \xrightarrow{\mathsf{ne}} \mathit{UPred}(M)}$

Remember that $\mathit{UPred}(M)$ is defined as a quotient on $M \xrightarrow{\mathsf{mon},\mathsf{ne}} \mathit{SProp}$, but we do not make the treatment of equivalence classes explicit. We are thus going to define the propositions as mapping camera elements to sets of step-indices and omit the implicit proof obligations that arise.

$$\begin{aligned}
[\![\Gamma \vdash t =_\tau u : \mathit{iProp}]\!]_\rho &\triangleq \lambda_.\ \left\{ n \;\middle|\; [\![\Gamma \vdash t : \tau]\!]_\rho \overset{n}{=} [\![\Gamma \vdash u : \tau]\!]_\rho \right\} \\
[\![\Gamma \vdash \mathsf{False} : \mathit{iProp}]\!]_\rho &\triangleq \lambda_.\ \emptyset \\
[\![\Gamma \vdash \mathsf{True} : \mathit{iProp}]\!]_\rho &\triangleq \lambda_.\ \mathbb{N} \\
[\![\Gamma \vdash P \wedge Q : \mathit{iProp}]\!]_\rho &\triangleq \lambda a.\ [\![\Gamma \vdash P : \mathit{iProp}]\!]_\rho(a) \cap [\![\Gamma \vdash Q : \mathit{iProp}]\!]_\rho(a) \\
[\![\Gamma \vdash P \vee Q : \mathit{iProp}]\!]_\rho &\triangleq \lambda a.\ [\![\Gamma \vdash P : \mathit{iProp}]\!]_\rho(a) \cup [\![\Gamma \vdash Q : \mathit{iProp}]\!]_\rho(a) \\
[\![\Gamma \vdash P \Rightarrow Q : \mathit{iProp}]\!]_\rho &\triangleq \lambda a.\ \left\{ n \;\middle|\; \begin{array}{c} \forall m, b.\ m \le n \wedge a \preccurlyeq b \wedge m \in \mathcal{V}(b) \Rightarrow \\ m \in [\![\Gamma \vdash P : \mathit{iProp}]\!]_\rho(b) \Rightarrow \\ m \in [\![\Gamma \vdash Q : \mathit{iProp}]\!]_\rho(b) \end{array} \right\} \\
[\![\Gamma \vdash \forall x : \tau.\ P : \mathit{iProp}]\!]_\rho &\triangleq \lambda a.\ \left\{ n \;\middle|\; \forall v \in [\![\tau]\!].\ n \in [\![\Gamma, x : \tau \vdash P : \mathit{iProp}]\!]_{\rho[x \leftarrow v]}(a) \right\} \\
[\![\Gamma \vdash \exists x : \tau.\ P : \mathit{iProp}]\!]_\rho &\triangleq \lambda a.\ \left\{ n \;\middle|\; \exists v \in [\![\tau]\!].\ n \in [\![\Gamma, x : \tau \vdash P : \mathit{iProp}]\!]_{\rho[x \leftarrow v]}(a) \right\}
\end{aligned}$$

$$\llbracket \Gamma \vdash P * Q : iProp \rrbracket_\rho \triangleq \lambda a.\ \left\{ n \;\middle|\; \begin{array}{l} \exists b_1, b_2.\ a \overset{n}{=} b_1 \cdot b_2 \wedge \\ \qquad n \in \llbracket \Gamma \vdash P : iProp \rrbracket_\rho(b_1) \wedge n \in \llbracket \Gamma \vdash Q : iProp \rrbracket_\rho(b_2) \end{array} \right\}$$

$$\llbracket \Gamma \vdash P \mathrel{-\!\!*} Q : iProp \rrbracket_\rho \triangleq \lambda a.\ \left\{ n \;\middle|\; \begin{array}{l} \forall m, b.\ m \leq n \wedge m \in \mathcal{V}(a \cdot b) \Rightarrow \\ \qquad m \in \llbracket \Gamma \vdash P : iProp \rrbracket_\rho(b) \Rightarrow \\ \qquad m \in \llbracket \Gamma \vdash Q : iProp \rrbracket_\rho(a \cdot b) \end{array} \right\}$$

$$\llbracket \Gamma \vdash \mathsf{Own}\,(t) : iProp \rrbracket_\rho \triangleq \lambda b.\ \left\{ n \mid \llbracket \Gamma \vdash t : \mathsf{M} \rrbracket_\rho \overset{n}{\preccurlyeq} b \right\}$$

$$\llbracket \Gamma \vdash \mathcal{V}(t) : iProp \rrbracket_\rho \triangleq \lambda\_.\ \mathcal{V}(\llbracket \Gamma \vdash t : \mathsf{M} \rrbracket_\rho)$$

$$\llbracket \Gamma \vdash \Box P : iProp \rrbracket_\rho \triangleq \lambda a.\ \llbracket \Gamma \vdash P : iProp \rrbracket_\rho(|a|)$$

$$\llbracket \Gamma \vdash \blacksquare P : iProp \rrbracket_\rho \triangleq \lambda a.\ \llbracket \Gamma \vdash P : iProp \rrbracket_\rho(\varepsilon)$$

$$\llbracket \Gamma \vdash \triangleright P : iProp \rrbracket_\rho \triangleq \lambda a.\ \{ n \mid n = 0 \vee n - 1 \in \llbracket \Gamma \vdash P : iProp \rrbracket_\rho(a) \}$$

$$\llbracket \Gamma \vdash \dot{\Rrightarrow} P : iProp \rrbracket_\rho \triangleq \lambda a.\ \left\{ n \;\middle|\; \begin{array}{l} \forall m, a'.\ m \leq n \wedge m \in \mathcal{V}(a \cdot a') \Rightarrow \\ \qquad \exists b.\ m \in \mathcal{V}(b \cdot a') \wedge m \in \llbracket \Gamma \vdash P : iProp \rrbracket_\rho(b) \end{array} \right\}$$

For every definition, we have to show all the side-conditions: The maps have to be non-expansive and monotone.

Interpretation of non-propositional terms $\boxed{\llbracket \Gamma \vdash t : \tau \rrbracket : \llbracket \Gamma \rrbracket \xrightarrow{\mathsf{ne}} \llbracket \tau \rrbracket}$

$$\begin{aligned}
\llbracket \Gamma \vdash x : \tau \rrbracket_\rho &\triangleq \rho(x) \\
\llbracket \Gamma \vdash F(t_1, \ldots, t_n) : \tau_{n+1} \rrbracket_\rho &\triangleq \llbracket F \rrbracket(\llbracket \Gamma \vdash t_1 : \tau_1 \rrbracket_\rho, \ldots, \llbracket \Gamma \vdash t_n : \tau_n \rrbracket_\rho) \\
\llbracket \Gamma \vdash \lambda x : \tau.\, t : \tau \to \tau' \rrbracket_\rho &\triangleq \lambda u : \llbracket \tau \rrbracket.\, \llbracket \Gamma, x : \tau \vdash t : \tau \rrbracket_{\rho[x \leftarrow u]} \\
\llbracket \Gamma \vdash t(u) : \tau' \rrbracket_\rho &\triangleq \llbracket \Gamma \vdash t : \tau \to \tau' \rrbracket_\rho(\llbracket \Gamma \vdash u : \tau \rrbracket_\rho) \\
\llbracket \Gamma \vdash \mu x : \tau.\, t : \tau \rrbracket_\rho &\triangleq \mathit{fix}_{\llbracket \tau \rrbracket}(\lambda u : \llbracket \tau \rrbracket.\, \llbracket \Gamma, x : \tau \vdash t : \tau \rrbracket_{\rho[x \leftarrow u]})
\end{aligned}$$

$$\begin{aligned}
\llbracket \Gamma \vdash \mathsf{abort}\ t : \tau \rrbracket_\rho &\triangleq \mathit{abort}_{\llbracket \tau \rrbracket}(\llbracket \Gamma \vdash t : 0 \rrbracket_\rho) \\
\llbracket \Gamma \vdash () : 1 \rrbracket_\rho &\triangleq () \\
\llbracket \Gamma \vdash (t_1, t_2) : \tau_1 \times \tau_2 \rrbracket_\rho &\triangleq (\llbracket \Gamma \vdash t_1 : \tau_1 \rrbracket_\rho, \llbracket \Gamma \vdash t_2 : \tau_2 \rrbracket_\rho) \\
\llbracket \Gamma \vdash \pi_i\ t : \tau_i \rrbracket_\rho &\triangleq \pi_i(\llbracket \Gamma \vdash t : \tau_1 \times \tau_2 \rrbracket_\rho) \\
\llbracket \Gamma \vdash \mathsf{inj}_i\ t : \tau_1 + \tau_2 \rrbracket_\rho &\triangleq \mathit{inj}_i(\llbracket \Gamma \vdash t : \tau_i \rrbracket_\rho) \\
\llbracket \Gamma \vdash \mathsf{match}\ t\ \mathsf{with}\ \mathsf{inj}_1\ x_1.\, t_1 \mid \mathsf{inj}_2\ x_2.\, t_2\ \mathsf{end} : \tau \rrbracket_\rho &\triangleq \llbracket \Gamma, x_i : \tau_i \vdash t_i : \tau \rrbracket_{\rho[x_i \leftarrow u]} \\
&\qquad \text{where } \llbracket \Gamma \vdash t : \tau_1 + \tau_2 \rrbracket_\rho = \mathit{inj}_i(u)
\end{aligned}$$

$$\begin{aligned}
\llbracket a : \mathsf{M} \rrbracket_\rho &\triangleq a \\
\llbracket \Gamma \vdash |t| : \mathsf{M} \rrbracket_\rho &\triangleq |\llbracket \Gamma \vdash t : \mathsf{M} \rrbracket_\rho| \\
\llbracket \Gamma \vdash t \cdot u : \mathsf{M} \rrbracket_\rho &\triangleq \llbracket \Gamma \vdash t : \mathsf{M} \rrbracket_\rho \cdot \llbracket \Gamma \vdash u : \mathsf{M} \rrbracket_\rho
\end{aligned}$$

An environment Γ is interpreted as the set of finite partial functions ρ, with $\mathsf{dom}(\rho) = \mathsf{dom}(\Gamma)$

and $\rho(x) \in [\![\Gamma(x)]\!]$. Above, *fix* is Banach's fixed-point (see Theorem 1), and $abort_T$ is the unique function $\emptyset \to T$.

Logical entailment. We can now define *semantic* logical entailment.

Interpretation of entailment $\boxed{[\![\Gamma \mid \Theta \vdash P]\!] : Prop}$

$$[\![\Gamma \mid P \vdash Q]\!] \triangleq \forall n \in \mathbb{N}.\, \forall r \in M.\, \forall \rho \in [\![\Gamma]\!],\\ n \in \mathcal{V}(r) \wedge n \in [\![\Gamma \vdash P : iProp]\!]_\rho(r) \Rightarrow n \in [\![\Gamma \vdash Q : iProp]\!]_\rho(r)$$

The following soundness theorem connects syntactic and semantic entailment. It is proven by showing that all the syntactic proof rules of §5 can be validated in the model.

$$\Gamma \mid P \vdash Q \Rightarrow [\![\Gamma \mid P \vdash Q]\!]$$

It now becomes straight-forward to show consistency of the logic.

7 Extensions of the Base Logic

In this section we discuss some additional constructions that we define within and on top of the base logic. These are not "extensions" in the sense that they change the proof power of the logic, they just form useful derived principles.

7.1 Derived Rules about Base Connectives

We collect here some important and frequently used derived proof rules.

$$P \Rrightarrow Q \vdash P \mathrel{-\!\!*} Q \qquad P * \exists x.\, Q \dashv\vdash \exists x.\, P * Q \qquad P * \forall x.\, Q \vdash \forall x.\, P * Q$$

Verifying that existential quantifiers commute with separating conjunction requires an intermediate step using a magic wand: From $P * \exists x, Q \vdash \exists x.\, P * Q$ we can deduce $\exists x.\, Q \vdash P \mathrel{-\!\!*} \exists x.\, P * Q$ and then proceed via $\exists$-elimination.

7.2 Derived Rules about Modalities

Iris comes with 4 built-in modalities ($\Box$, $\blacksquare$, $\pupd$ and $\triangleright$) and, as we will see, plenty of derived modalities. However, almost all of them fall into one of two categories (except for $\triangleright$, as we will see): they are either *always-style* modalities ("something holds in all/many (future) worlds") or *eventually-style* modalities ("something holds in a possible (future) world").

Eventually-style modalities are characterized by being easy to "add"/introduce, but hard to "remove"/eliminate. Consider, for example, the basic update modality $\pupd$: we have $P \vdash \pupd P$ (UPD-I), but the inverse direction does not hold. Instead, from UPD-MONO and UPD-TRANS, we can derive the following elimination principle:

$$\text{UPD-E}\ \frac{P \vdash \pupd Q}{\pupd P \vdash \pupd Q}$$

In other words, we can remove an $\pupd$ in front of an assumption *if* the goal is itself wrapped in $\pupd$. Another way to view this rule is to think of it as a *bind rule*. Indeed, together with UPD-I, this rule shows that $\pupd$ forms a monad.

Always-style modalities, on the other hand, are easy to "remove"/eliminate, but hard to "add"/introduce. The most widely used example of that in Iris is the persistence modality $\Box$: we have $\Box P \vdash P$ ($\Box$-E), but the inverse direction does not hold. Instead, from $\Box$-MONO and $\Box P \vdash \Box\Box P$, we can derive the following introduction principle:

$$\Box\text{-I}\ \frac{\Box P \vdash Q}{\Box P \vdash \Box Q}$$

In other words, we can remove an $\Box$ from the goal *if* all our assumptions are wrapped in $\Box$. This matches the algebraic structure of a comonad.

In particular, both eventually-style and always-style modalities are *idempotent*: we have $\pupd \pupd P \dashv\vdash \pupd P$ and $\Box\Box P \dashv\vdash \Box P$.

Beyond this, all modalities come with plenty of rules that show how they commute around other connectives and modalities. And, of course, they come with a few "defining rules" that give the modalities their individual meaning, *i.e.,* for the update modality, that would be UPD-UPDATE.

In the following, we briefly discuss each of the modalities.

Update modality. As already mentioned, the update modality is an eventually-style modality:

$$\text{UPD-E}\ \frac{P \vdash \dot{\Rightarrow} Q}{\dot{\Rightarrow} P \vdash \dot{\Rightarrow} Q} \qquad \text{UPD-IDEMP}\ \dot{\Rightarrow}\dot{\Rightarrow} P \dashv\vdash \dot{\Rightarrow} P$$

Beyond this (and the obvious variant of UPD-FRAME that exploits commutativity of separating conjunction), there are no outstandingly interesting derived rules.

Persistence modality. As already mentioned, the persistence modality is an always-style modality:

$$\Box\text{-I}\ \frac{\Box P \vdash Q}{\Box P \vdash \Box Q} \qquad \Box\text{-IDEMP}\ \Box\Box P \dashv\vdash \Box P$$

Some further interesting derived rules include:

$$\Box(P \wedge Q) \dashv\vdash \Box P \wedge \Box Q \qquad \Box(P \vee Q) \dashv\vdash \Box P \vee \Box Q \qquad \Box \mathsf{True} \dashv\vdash \mathsf{True} \qquad \Box \mathsf{False} \dashv\vdash \mathsf{False}$$

$$\Box(P * Q) \dashv\vdash \Box P * \Box Q \qquad \Box P * Q \dashv\vdash \Box P \wedge Q \qquad \Box(P \mathrel{-\!\!*} Q) \dashv\vdash \Box(P \Rightarrow Q)$$

$$\Box(P \Rightarrow Q) \vdash \Box P \Rightarrow \Box Q \qquad \Box(P \mathrel{-\!\!*} Q) \vdash \Box P \mathrel{-\!\!*} \Box Q$$

In particular, the persistence modality commutes around conjunction, disjunction, separating conjunction as well as universal and existential quantification. Commuting around conjunction can be derived from the primitive rule that says it commutes around universal quantification (as conjunction is equivalent to a universal quantification of a Boolean), and similar for disjunction. $\mathsf{True} \dashv\vdash \Box \mathsf{True}$ (which is basically persistence "commuting around" the nullary operator True) can be derived via $\Box$ commuting with universal quantification ranging over the empty type. A similar rule holds for False.

Moreover, if (at least) one conjunct is below the persistence modality, then conjunction and separating conjunction coincide.

Plainness modality. The plainness modality is very similar to the persistence modality (in fact, we have $\blacksquare P \vdash \Box P$, but the inverse does not hold). It is always-style:

$$\blacksquare\text{-I}\ \frac{\blacksquare P \vdash Q}{\blacksquare P \vdash \blacksquare Q} \qquad \blacksquare\blacksquare P \dashv\vdash \blacksquare P$$

It also commutes around separating conjunction, conjunction, disjunction, universal and existential quantification (and True and False).

The key difference to the persistence modality $\Box$ is that $\blacksquare$ provides a *propositional extensionality* principle:

$$\blacksquare((P \Rightarrow Q) \land (Q \Rightarrow P)) \vdash P =_{iProp} Q$$

In contrast, $\Box$ permits using some forms of ghost state ($\mathsf{Own}\,(a) \vdash \Box\,\mathsf{Own}\,(|a|)$).

Having both of these principles for the same modality would lead to a contradiction: imagine we have an RA with elements a, b such that $|a|$ is incompatible with b (*i.e.*, $\neg\overline{\mathcal{V}}(|a| \cdot b)$). Then we can prove:

$$\mathsf{Own}\,(|a|) \vdash \Box\,\mathsf{Own}\,(|a|) \vdash \Box((\mathsf{False} \Rightarrow \mathsf{Own}\,(b)) \land (\mathsf{Own}\,(b) \Rightarrow \mathsf{False}))$$

The first implication is trivial, the second implication follows because $\Box\,\mathsf{Own}\,(|a|) \land \mathsf{Own}\,(b) \vdash \mathsf{Own}\,(|a|) * \mathsf{Own}\,(b) \vdash \mathcal{V}(|a| \cdot b)$.

But now, if we had propositional extensionality for $\Box$ the way we do for $\blacksquare$, we could deduce $\mathsf{False} =_{iProp} \mathsf{Own}\,(b)$, and that is clearly wrong. This issue arises because $\Box$, as we have seen, still lets us use some resources from the context, while propositional equality has to hold completely disregarding current resources.

Later modality. The later modality is the "odd one out" in the sense that it is neither eventually-style nor always-style, because it is not idempotent:[3] with $\triangleright$, the number of times the modality is applied matters, and we can get rid of *exactly one* layer of $\triangleright$ in the assumptions only by doing the same in the conclusion ($\triangleright$-MONO).

Some derived rules:

LÖB
$$(\triangleright P \Rightarrow P) \vdash P \qquad \triangleright(P \Rightarrow Q) \vdash \triangleright P \Rightarrow \triangleright Q \qquad \triangleright(P \mathrel{-\!\!*} Q) \vdash \triangleright P \mathrel{-\!\!*} \triangleright Q$$

$$\triangleright(P \land Q) \dashv\vdash \triangleright P \land \triangleright Q \qquad \triangleright(P \lor Q) \dashv\vdash \triangleright P \lor \triangleright Q \qquad \frac{\tau \text{ is inhabited}}{\triangleright(\exists x : \tau.\, P) \dashv\vdash \exists x : \tau.\, \triangleright P}$$

$$\triangleright\,\mathsf{True} \dashv\vdash \mathsf{True} \qquad \triangleright(P * Q) \dashv\vdash \triangleright P * \triangleright Q \qquad \triangleright\Box P \dashv\vdash \Box\triangleright P \qquad \triangleright\blacksquare P \dashv\vdash \blacksquare\triangleright P$$

Noteworthy here is the fact that Löb induction (LÖB) can be derived from $\triangleright$-introduction and the fact that we can take fixed-points of functions where the recursive occurrences are below $\triangleright$ [8].[4] Also, $\triangleright$ commutes over separating conjunction, conjunction, disjunction, universal quantification and *non-empty* existential quantification, as well as both the persistence and the plainness modality.

7.3 Persistent Propositions

We call a proposition P *persistent* if $P \vdash \Box P$. These are propositions that "do not own anything", so we can (and will) treat them like "normal" intuitionistic propositions.

Of course, $\Box P$ is persistent for any P. Furthermore, by the proof rules given in §5.3, True, False, $t = t'$ as well as $\ulcorner\!\!\lfloor a \rfloor\!\!\urcorner^{\gamma}$ and $\mathcal{V}(a)$ are persistent. Persistence is preserved by conjunction, disjunction, separating conjunction as well as universal and existential quantification and $\triangleright$.

[3] This means $\triangleright$ is neither a monad nor a comonad—it does form an applicative functor, though.

[4] Also see https://en.wikipedia.org/wiki/L%C3%B6b%27s_theorem.

7.4 Timeless Propositions and Except-0

One of the troubles of working in a step-indexed logic is the "later" modality $\triangleright$. It turns out that we can somewhat mitigate this trouble by working below the following *except-0* modality:

$$\diamond P \triangleq \triangleright \mathsf{False} \lor P$$

Except-0 satisfies the usual laws of a "monadic" modality (similar to, *e.g.*, the update modalities):

$$\textsc{ex0-mono}\ \frac{P \vdash Q}{\diamond P \vdash \diamond Q} \qquad \textsc{ex0-intro}\ P \vdash \diamond P \qquad \textsc{ex0-idem}\ \diamond\diamond P \vdash \diamond P$$

$$\diamond(P * Q) \dashv\vdash \diamond P * \diamond Q \qquad \diamond(P \land Q) \dashv\vdash \diamond P \land \diamond Q \qquad \diamond(P \lor Q) \dashv\vdash \diamond P \lor \diamond Q$$

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

In particular, from EX0-MONO and EX0-IDEM we can derive a "bind"-like elimination rule:

$$\textsc{ex0-elim}\ \frac{P \vdash \diamond Q}{\diamond P \vdash \diamond Q}$$

This modality is useful because there is a class of propositions which we call *timeless* propositions, for which we have

$$\mathsf{timeless}(P) \triangleq \triangleright P \vdash \diamond P$$

In other words, when working below the except-0 modality, we can *strip away* the later from timeless propositions (using EX0-ELIM):

$$\textsc{ex0-timeless-strip}\ \frac{\mathsf{timeless}(P) \qquad P \vdash \diamond Q}{\triangleright P \vdash \diamond Q}$$

In fact, it turns out that we can strip away later from timeless propositions even when working under the later modality:

$$\textsc{later-timeless-strip}\ \frac{\mathsf{timeless}(P) \qquad P \vdash \triangleright Q}{\triangleright P \vdash \triangleright Q}$$

This follows from $\triangleright P \vdash \triangleright \mathsf{False} \lor P$, and then by straightforward disjunction elimination.

The following rules identify the class of timeless propositions:

$$\frac{\Gamma \vdash \mathsf{timeless}(P) \qquad \Gamma \vdash \mathsf{timeless}(Q)}{\Gamma \vdash \mathsf{timeless}(P \land Q)} \qquad \frac{\Gamma \vdash \mathsf{timeless}(P) \qquad \Gamma \vdash \mathsf{timeless}(Q)}{\Gamma \vdash \mathsf{timeless}(P \lor Q)}$$

$$\frac{\Gamma \vdash \mathsf{timeless}(P) \qquad \Gamma \vdash \mathsf{timeless}(Q)}{\Gamma \vdash \mathsf{timeless}(P * Q)} \qquad \frac{\Gamma \vdash \mathsf{timeless}(P)}{\Gamma \vdash \mathsf{timeless}(\Box P)} \qquad \frac{\Gamma \vdash \mathsf{timeless}(Q)}{\Gamma \vdash \mathsf{timeless}(P \Rightarrow Q)}$$

$$\frac{\Gamma \vdash \mathsf{timeless}(Q)}{\Gamma \vdash \mathsf{timeless}(P \mathrel{-\!\!*} Q)} \qquad \frac{\Gamma, x : \tau \vdash \mathsf{timeless}(P)}{\Gamma \vdash \mathsf{timeless}(\forall x : \tau.\, P)} \qquad \frac{\Gamma, x : \tau \vdash \mathsf{timeless}(P)}{\Gamma \vdash \mathsf{timeless}(\exists x : \tau.\, P)} \qquad \mathsf{timeless}(\mathsf{True})$$

$$\mathsf{timeless}(\mathsf{False}) \qquad \frac{t \text{ or } t' \text{ is a discrete OFE element}}{\mathsf{timeless}(t =_\tau t')} \qquad \frac{a \text{ is a discrete OFE element}}{\mathsf{timeless}(\mathsf{Own}\,(a))}$$

$$\frac{a \text{ is an element of a discrete camera}}{\mathsf{timeless}(\mathcal{V}(a))}$$

7.5 Dynamic Composable Higher-Order Resources

The base logic described in §5 works over an arbitrary camera M defining the structure of the resources. It turns out that we can generalize this further and permit picking cameras "$\Sigma(\mathit{iProp})$" that depend on the structure of propositions themselves. Of course, *iProp* is just the syntactic type of propositions; for this to make sense we have to look at the semantics.

Furthermore, there is a composability problem with the given logic: if we have one proof performed with camera M_1, and another proof carried out with a *different* camera M_2, then the two proofs are actually carried out in two *entirely separate logics* and hence cannot be combined.

Finally, in many cases just having a single "instance" of a camera available for reasoning is not enough. For example, when reasoning about a dynamically allocated data structure, every time a new instance of that data structure is created, we will want a fresh resource governing the state of this particular instance. While it would be possible to handle this problem whenever it comes up, it turns out to be useful to provide a general solution.

The purpose of this section is to describe how we solve these issues.

Picking the resources. The key ingredient that we will employ on top of the base logic is to give some more fixed structure to the resources. To instantiate the logic with dynamic higher-order ghost state, the user picks a family of locally contractive bifunctors $(\Sigma_i : \mathbf{COFE}^{\mathsf{op}} \times \mathbf{COFE} \to \mathbf{Camera})_{i \in \mathcal{I}}$. (This is in contrast to the base logic, where the user picks a single, fixed camera that has a unit.)

From this, we construct the bifunctor defining the overall resources as follows:

$$\begin{aligned} \mathit{GName} &\triangleq \mathbb{N} \\ \mathit{ResF}(T^{\mathsf{op}}, T) &\triangleq \prod_{i \in \mathcal{I}} \mathit{GName} \xrightarrow{\mathsf{fin}} \Sigma_i(T^{\mathsf{op}}, T) \end{aligned}$$

We will motivate both the use of a product and the finite partial function below. $\mathit{ResF}(T^{\mathsf{op}}, T)$ is a camera by lifting the individual cameras pointwise, and it has a unit (using the empty finite partial function). Furthermore, since the Σ_i are locally contractive, so is *ResF*.

Now we can write down the recursive domain equation:

$$\mathit{iPreProp} \cong \mathit{UPred}(\mathit{ResF}(\mathit{iPreProp}, \mathit{iPreProp}))$$

Here, *iPreProp* is a COFE defined as the fixed-point of a locally contractive bifunctor, which exists

and is unique up to isomorphism by Theorem 2, so we obtain some object *iPreProp* such that:

$$\begin{aligned} \mathit{Res} &\triangleq \mathit{ResF}(\mathit{iPreProp}, \mathit{iPreProp}) \\ \mathit{iProp} &\triangleq \mathit{UPred}(\mathit{Res}) \\ \xi &: \mathit{iProp} \xrightarrow{\mathsf{ne}} \mathit{iPreProp} \\ \xi^{-1} &: \mathit{iPreProp} \xrightarrow{\mathsf{ne}} \mathit{iProp} \\ \xi(\xi^{-1}(x)) &\triangleq x \\ \xi^{-1}(\xi(x)) &\triangleq x \end{aligned}$$

Now we can instantiate the base logic described in §5 with *Res* as the chosen camera:

$$[\![\mathit{iProp}]\!] \triangleq \mathit{UPred}(\mathit{Res})$$

We obtain that $[\![\mathit{iProp}]\!] = \mathit{iProp}$. Effectively, we just defined a way to instantiate the base logic with *Res* as the camera of resources, while providing a way for *Res* to depend on *iPreProp*, which is isomorphic to $[\![\mathit{iProp}]\!]$.

We thus obtain all the rules of §5, and furthermore, we can use the maps ξ and ξ^{-1} *in the logic* to convert between logical propositions $[\![\mathit{iProp}]\!]$ and the domain *iPreProp* which is used in the construction of *Res* – so from elements of *iPreProp*, we can construct elements of $[\![\mathsf{M}]\!]$, which are the elements that can be owned in our logic.

Proof composability. To make our proofs composable, we *generalize* our proofs over the family of functors. This is possible because we made *Res* a *product* of all the cameras picked by the user, and because we can actually work with that product "pointwise". So instead of picking a *concrete* family, proofs will assume to be given an *arbitrary* family of functors, plus a proof that this family *contains the functors they need*. Composing two proofs is then merely a matter of conjoining the assumptions they make about the functors. Since the logic is entirely parametric in the choice of functors, there is no trouble reasoning without full knowledge of the family of functors.

Only when the top-level proof is completed we will "close" the proof by picking a concrete family that contains exactly those functors the proof needs.

Dynamic resources. Finally, the use of finite partial functions lets us have as many instances of any camera as we could wish for: Because there can only ever be finitely many instances already allocated, it is always possible to create a fresh instance with any desired (valid) starting state. This is best demonstrated by giving some proof rules.

So let us first define the notion of ghost ownership that we use in this logic. Assuming that the family of functors contains the functor Σ_i at index i, and furthermore assuming that $M_i = \Sigma_i(\mathit{iPreProp}, \mathit{iPreProp})$, given some $a \in M_i$ we define:

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

This is ownership of the pair (element of the product over all the functors) that has the empty finite partial function in all components *except for* the component corresponding to index i, where we own the element a at index γ in the finite partial function.

We can show the following properties for this form of ownership:

RES-ALLOC
$$\frac{G \text{ infinite} \qquad a \in \mathcal{V}_{M_i}}{\mathsf{True} \vdash \dot{\Rightarrow}\exists \gamma \in G.\, \boxed{a : M_i}^{\gamma}}$$

RES-UPDATE
$$\frac{a \rightsquigarrow_{M_i} B}{\boxed{a : M_i}^{\gamma} \vdash \dot{\Rightarrow}\exists b \in B.\, \boxed{b : M_i}^{\gamma}}$$

RES-EMPTY
$$\frac{\varepsilon \text{ is a unit of } M_i}{\mathsf{True} \vdash \dot{\Rightarrow}\boxed{\varepsilon}^{\gamma}}$$

RES-OP
$$\boxed{a : M_i}^{\gamma} * \boxed{b : M_i}^{\gamma} \dashv\vdash \boxed{a \cdot b : M_i}^{\gamma}$$

RES-VALID
$$\boxed{a : M_i}^{\gamma} \Rightarrow \mathcal{V}_{M_i}(a)$$

RES-TIMELESS
$$\frac{a \text{ is a discrete OFE element}}{\mathsf{timeless}(\boxed{a : M_i}^{\gamma})}$$

Below, we will always work within (an instance of) the logic as described here. Whenever a camera is used in a proof, we implicitly assume it to be available in the global family of functors. We will typically leave the M_i implicit when asserting ghost ownership, as the type of a will be clear from the context.

8 Language

A *language* Λ consists of a set *Expr* of *expressions* (metavariable e), a set *Val* of *values* (metavariable v), a set *Obs* of *observations*[5] (or "observable events") and a set *State* of *states* (metavariable σ) such that

- There exist functions $\mathsf{val\_to\_expr} : \mathit{Val} \to \mathit{Expr}$ and $\mathsf{expr\_to\_val} : \mathit{Expr} \rightharpoonup \mathit{Val}$ (notice the latter is partial), such that

$$\forall e, v.\, \mathsf{expr\_to\_val}(e) = v \Rightarrow \mathsf{val\_to\_expr}(v) = e \qquad \forall v.\, \mathsf{expr\_to\_val}(\mathsf{val\_to\_expr}(v)) = v$$

- There exists a *primitive reduction relation*

$$(-, - \ \overset{-}{\to}_{\mathsf{t}} \ -, -, -) \subseteq (\mathit{Expr} \times \mathit{State}) \times \mathit{List}(\mathit{Obs}) \times (\mathit{Expr} \times \mathit{State} \times \mathit{List}(\mathit{Expr}))$$

  A reduction $e_1, \sigma_1 \overset{\vec{\kappa}}{\to}_{\mathsf{t}} e_2, \sigma_2, \vec{e}$ indicates that, when e_1 in state σ_1 reduces to e_2 with new state σ_2, the new threads in the list $\vec{e}$ is forked off and the observations $\vec{\kappa}$ are made. We will write $e_1, \sigma_1 \to_{\mathsf{t}} e_2, \sigma_2$ for $e_1, \sigma_1 \overset{()}{\to}_{\mathsf{t}} e_2, \sigma_2, ()$, *i.e.,* when no threads are forked off and no observations are made.

- All values are stuck:

$$e, \_ \to_{\mathsf{t}} \_, \_, \_ \Rightarrow \mathsf{expr\_to\_val}(e) = \bot$$

Definition 19. *An expression e and state σ are* reducible *(written $\mathsf{red}(e, \sigma)$) if*

$$\exists \vec{\kappa}, e_2, \sigma_2, \vec{e}.\, e, \sigma \overset{\vec{\kappa}}{\to}_{\mathsf{t}} e_2, \sigma_2, \vec{e}$$

Definition 20. *An expression e is* weakly atomic *if it reduces in one step to something irreducible:*

$$\mathsf{atomic}(e) \triangleq \forall \sigma_1, \vec{\kappa}, e_2, \sigma_2, \vec{e}.\, e, \sigma_1 \overset{\vec{\kappa}}{\to}_{\mathsf{t}} e_2, \sigma_2, \vec{e} \Rightarrow \neg \mathsf{red}(e_2, \sigma_2)$$

It is strongly atomic *if it reduces in one step to a value:*

$$\mathsf{strongly\_atomic}(e) \triangleq \forall \sigma_1, \vec{\kappa}, e_2, \sigma_2, \vec{e}.\, e, \sigma_1 \overset{\vec{\kappa}}{\to}_{\mathsf{t}} e_2, \sigma_2, \vec{e} \Rightarrow \mathsf{expr\_to\_val}(e_2) \neq \bot$$

We need two notions of atomicity to accommodate both kinds of weakest preconditions that we will define later: If the weakest precondition ensures that the program cannot get stuck, weak atomicity is sufficient. Otherwise, we need strong atomicity.

Definition 21 (Context)**.** *A function $K : \mathit{Expr} \to \mathit{Expr}$ is a* context *if the following conditions are satisfied:*

1. *K does not turn non-values into values:*

$$\forall e.\, \mathsf{expr\_to\_val}(e) = \bot \Rightarrow \mathsf{expr\_to\_val}(K(e)) = \bot$$

[5] See https://gitlab.mpi-sws.org/iris/iris/merge_requests/173 for how observations are useful to encode prophecy variables.

2. One can perform reductions below K:

$$\forall e_1, \sigma_1, \vec{\kappa}, e_2, \sigma_2, \vec{e}.\, e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e_2, \sigma_2, \vec{e} \Rightarrow K(e_1), \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{t}} K(e_2), \sigma_2, \vec{e}$$

3. Reductions stay below K until there is a value in the hole:

$$\begin{aligned}\forall e_1', \sigma_1, \vec{\kappa}, e_2, \sigma_2, \vec{e}.\, \mathsf{expr\_to\_val}(e_1') = \bot \wedge K(e_1'), \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e_2, \sigma_2, \vec{e} \Rightarrow \\ \exists e_2'.\, e_2 = K(e_2') \wedge e_1', \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e_2', \sigma_2, \vec{e}\end{aligned}$$

8.1 Concurrent Language

For any language Λ, we define the corresponding thread-pool semantics.

Machine syntax

$$T \in \mathit{ThreadPool} \triangleq \mathit{List}(\mathit{Expr})$$

Machine reduction $\boxed{T; \sigma \xrightarrow{\vec{\kappa}}_{\mathsf{tp}} T'; \sigma'}$

$$\frac{e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e_2, \sigma_2, \vec{e}}{T +\!\!+ [e_1] +\!\!+ T'; \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{tp}} T +\!\!+ [e_2] +\!\!+ T' +\!\!+ \vec{e}; \sigma_2}$$

We use $\xrightarrow{-}{}^{*}_{\mathsf{tp}}$ for the reflexive transitive closure of $\xrightarrow{-}_{\mathsf{tp}}$, as usual concatenating the lists of observations of the individual steps.

9 Program Logic

This section describes how to build a program logic for an arbitrary language (*c.f.* §8) on top of the base logic. So in the following, we assume that some language Λ was fixed. Furthermore, we work in the logic with higher-order ghost state as described in §7.5.

9.1 Later Credits

Introducing a later modality is easy (see ▷-I), but eliminating them can be tricky. Laters often appear in the middle of our proofs after unfolding a circular construction (*e.g.,* by opening an invariant, see Section 9.2). In these cases, we get to assume $\triangleright P$, but we really want P to continue in the proof. Iris offers us four options to do so. We have seen two of them already: timeless propositions (see Section 7.4) and the commuting rules for later. Together, they can be used to turn $\triangleright P$ into P, or to delay when we have to deal with the later modality by commuting it inward (*e.g.,* over an existential quantifer and a separating conjunction). Another option, which we will encounter in Section 9.3, is taking program steps: every program step allows us to eliminate (at least) one later (see WP-LIFT-STEP). We now introduce the fourth option: *later credits*. Later credits turn the right to eliminate a later into an ownable separation-logic resource $£\,n$, where n is the number of laters that we can eliminate.

Resources We assume that the camera

$$\mathit{LaterCredits} \triangleq \mathit{Auth}(\mathbb{N}^+)$$

is available (*i.e.,* part of Σ), where $\mathbb{N}^+$ is the RA derived from the monoid $\mathbb{N}$ with operation $+$, and that an instance of it has been created and made globally available at the beginning of verification under the name $\gamma_{\mathit{Credits}}$. We define the following notations for the fragments and authoritative part:

$$£\,n \triangleq \boxed{\circ n}^{\gamma_{\mathit{Credits}}} \qquad £_\bullet\, n \triangleq \boxed{\bullet n}^{\gamma_{\mathit{Credits}}}$$

This definition satisfies the following laws:

CREDIT-SPLIT
$£\,(n+m) \Leftrightarrow £\,n * £\,m$

CREDIT-TIMELESS
$\mathsf{timeless}(£\,n)$

CREDIT-SUPPLYBOUND
$£_\bullet\, m * £\,n \vdash m \geq n$

CREDIT-SUPPLYDECR
$£_\bullet\, (n+m) * £\,n \vdash \Rrightarrow £_\bullet\, m$

CREDIT-SUPPLYEXCL
$£_\bullet\, m_1 * £_\bullet\, m_2 \vdash \mathsf{False}$

Later Elimination Update To eliminate laters by *spending* later credits $£\,n$, we define a *later-elimination update* $\Rrightarrow^{£} P$ on top of the basic update modality. It satisfies all the properties of basic updates, except for UPD-PLAINLY, but with the additional rule

CREDIT-UPD-USE
$$\triangleright P * £\,1 \vdash \Rrightarrow^{£} P$$

CREDIT-UPD-USE allows to *spend* one credit in exchange for stripping one later off of P.

The later-elimination update is defined by guarded recursion:

$$\Rrightarrow^{£} \triangleq \mu\ \mathit{upd}.\ \lambda P.\ \forall n.\ £_\bullet\, n \mathrel{-\!\!*} \dot{\Rrightarrow} (£_\bullet\, n * P) \vee (\exists m < n.\ £_\bullet\, m * \triangleright\, \mathit{upd}(P))$$

It threads through the authoritative resource $\pounds_\bullet n$, the credit supply, to control how many credits can be spent in total. The basic update ensures that the later elimination update inherits the ability to update resources. In the first disjunct (the "base case"), no credits are spent. In the second disjunct (the "recursive case"), a later can be eliminated before going into recursion. To take the second disjunct, the credit supply $\pounds_\bullet n$ has to be decreased, which means giving up at least $\pounds 1$.

The later-elimination update satisfies the following laws:

CREDIT-UPD-MONO
$$\frac{P \vdash Q}{\Rrightarrow^{\pounds} P \vdash \Rrightarrow^{\pounds} Q}$$

CREDIT-UPD-INTRO
$$P \vdash \Rrightarrow^{\pounds} P$$

CREDIT-UPD-TRANS
$$\Rrightarrow^{\pounds} \Rrightarrow^{\pounds} P \vdash \Rrightarrow^{\pounds} P$$

CREDIT-UPD-FRAME
$$Q * \Rrightarrow^{\pounds} P \vdash \Rrightarrow^{\pounds} (Q * P)$$

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

CREDIT-UPD-LATER
$$\pounds 1 * \triangleright \Rrightarrow^{\pounds} P \vdash \Rrightarrow^{\pounds} P$$

The rule CREDIT-UPD-USE shown above can be derived from CREDIT-UPD-LATER and CREDIT-UPD-INTRO.

Note the absence of a rule corresponding to UPD-PLAINLY, which is not validated by the model. As some existing Iris developments rely on UPD-PLAINLY, we parameterize the logic by a boolean constant $\mathsf{UseLaterCredits}$ that determines whether the later-elimination update is used instead of the basic update, in particular in the definition of fancy updates below.

9.2 World Satisfaction, Invariants, Fancy Updates

To introduce invariants into our logic, we will define weakest precondition to explicitly thread through the proof that all the invariants are maintained throughout program execution. However, in order to be able to access invariants, we will also have to provide a way to *temporarily disable* (or "open") them. To this end, we use tokens that manage which invariants are currently enabled.

We assume to have the following four cameras available:

$$\begin{aligned} \mathit{InvName} &\triangleq \mathbb{N} \\ \mathit{Inv} &\triangleq \mathit{Auth}(\mathit{InvName} \xrightarrow{\mathsf{fin}} \mathit{Ag}(\blacktriangleright \mathit{iPreProp})) \\ \mathit{En} &\triangleq \wp(\mathit{InvName}) \\ \mathit{Dis} &\triangleq \wp^{\mathsf{fin}}(\mathit{InvName}) \end{aligned}$$

The last two are the tokens used for managing invariants, Inv is the monoid used to manage the invariants themselves.

We assume that at the beginning of the verification, instances named γ_{Inv}, γ_{En} and γ_{Dis} of these cameras have been created, such that these names are globally known.

World Satisfaction. We can now define the proposition W (*world satisfaction*) which ensures that the enabled invariants are actually maintained:

$$W \triangleq \exists I : \mathit{InvName} \xrightarrow{\mathsf{fin}} \mathit{iProp}.\ \boxed{\bullet \lceil \iota \leftarrow \mathsf{ag}(\mathsf{next}(\xi(I(\iota)))) \mid \iota \in \mathsf{dom}(I) \rceil}^{\gamma_{\mathit{Inv}}} * \ast_{\iota \in \mathsf{dom}(I)} \left(\triangleright I(\iota) * \boxed{\{\iota\}}^{\gamma_{\mathit{Dis}}} \vee \boxed{\{\iota\}}^{\gamma_{\mathit{En}}} \right)$$

Invariants. The following proposition states that an invariant with name ι exists and maintains proposition P:

$$\mathsf{Inv}^{\iota}\,(P) \triangleq \boxed{\circ\,\boxed{\iota \leftarrow \mathsf{ag}(\mathsf{next}(\xi(P)))}}^{\gamma_{Inv}}$$

Fancy Updates and View Shifts. Next, we define *fancy updates*, which are essentially the same as the basic updates of the base logic (§5) or later-elimination updates (§9.1), except that they also have access to world satisfaction and can enable and disable invariants.

Depending on how the logic is parameterized with the UseLaterCredits, fancy updates are defined on top the basic update or the later-elimination update. This influences which rules fancy updates satisfy: either fancy updates can be used to eliminate later credits, or they satisfy certain interaction laws with the plainly modality.

$$\mathcal{E}_1 \Mapsto_{\mathcal{E}_2} P \triangleq \begin{cases} W * \boxed{\mathcal{E}_1}^{\gamma_{En}} \mathrel{-\!\!*} \Mapsto^{£} \diamond (W * \boxed{\mathcal{E}_2}^{\gamma_{En}} * P) & \text{if } \mathsf{UseLaterCredits} = \mathsf{true} \\ W * \boxed{\mathcal{E}_1}^{\gamma_{En}} \mathrel{-\!\!*} \dot{\Mapsto} \diamond (W * \boxed{\mathcal{E}_2}^{\gamma_{En}} * P) & \text{if } \mathsf{UseLaterCredits} = \mathsf{false} \end{cases}$$

Here, $\mathcal{E}_1$ and $\mathcal{E}_2$ are the *masks* of the view update, defining which invariants have to be (at least!) available before and after the update. Masks are sets of natural numbers, *i.e.,* they are subsets of $\mathbb{N}$.[6] We use $\top$ as symbol for the largest possible mask, $\mathbb{N}$, and $\bot$ for the smallest possible mask $\emptyset$. We will write $\Mapsto_{\mathcal{E}} P$ for ${}_{\mathcal{E}}\Mapsto_{\mathcal{E}} P$. Fancy updates satisfy the following basic proof rules:

FUP-MONO
$$\frac{P \vdash Q}{\mathcal{E}_1 \Mapsto_{\mathcal{E}_2} P \vdash \mathcal{E}_1 \Mapsto_{\mathcal{E}_2} Q}$$

FUP-INTRO-MASK
$$\frac{\mathcal{E}_2 \subseteq \mathcal{E}_1}{P \vdash \mathcal{E}_1 \Mapsto_{\mathcal{E}_2} \mathcal{E}_2 \Mapsto_{\mathcal{E}_1} P}$$

FUP-TRANS
$$\mathcal{E}_1 \Mapsto_{\mathcal{E}_2} \mathcal{E}_2 \Mapsto_{\mathcal{E}_3} P \vdash \mathcal{E}_1 \Mapsto_{\mathcal{E}_3} P$$

FUP-UPD
$$\dot{\Mapsto} P \vdash \Mapsto_{\mathcal{E}} P$$

FUP-FRAME
$$Q * \mathcal{E}_1 \Mapsto_{\mathcal{E}_2} P \vdash \mathcal{E}_1 \uplus \mathcal{E}_{\mathsf{f}} \Mapsto_{\mathcal{E}_2 \uplus \mathcal{E}_{\mathsf{f}}} Q * P$$

FUP-UPDATE
$$\frac{a \rightsquigarrow B}{\mathsf{Own}\,(a) \vdash \Mapsto_{\mathcal{E}} \exists b \in B.\, \mathsf{Own}\,(b)}$$

FUP-TIMELESS
$$\frac{\mathsf{timeless}(P)}{\triangleright P \vdash \Mapsto_{\mathcal{E}} P}$$

FUP-CREDIT-USE
$$\frac{\mathsf{UseLaterCredits} = \mathsf{true}}{£\,1 * \triangleright \mathcal{E}_1 \Mapsto_{\mathcal{E}_2} P \vdash \mathcal{E}_1 \Mapsto_{\mathcal{E}_2} P}$$

(There are no rules related to invariants here. Those rules will be discussed later, in §9.4.)

We can further define the notions of *view shifts* and *linear view shifts*:

$$P \;{}_{\mathcal{E}_1}\!\Rrightarrow\!\!\!\!*_{\mathcal{E}_2}\; Q \triangleq P \mathrel{-\!\!*} \mathcal{E}_1 \Mapsto_{\mathcal{E}_2} Q$$
$$P \;{}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2}\; Q \triangleq \Box(P \mathrel{-\!\!*} \mathcal{E}_1 \Mapsto_{\mathcal{E}_2} Q)$$
$$P \Rrightarrow_{\mathcal{E}} Q \triangleq P \;{}_{\mathcal{E}}\!\Rrightarrow_{\mathcal{E}} Q$$

These two are useful when writing down specifications and for comparing with previous versions of Iris, but for reasoning, it is typically easier to just work directly with fancy updates. Still, just to give an idea of what view shifts "are", here are some proof rules for them:

[6] Actually, in the Coq development masks are restricted to a class of sets of natural numbers that contains all finite sets and is closed under union, intersection, difference and complement. The restriction is necessary for engineering reasons to still obtain representation independence: two masks should be *propositionally* equal iff they contain the same invariant names.

VS-UPDATE
$$\frac{a \rightsquigarrow B}{\boxed{a}^{\gamma} \Rrightarrow_{\emptyset} \exists b \in B.\, \boxed{b}^{\gamma}}$$

VS-TRANS
$$\frac{P \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} Q \qquad Q \;\; {}_{\mathcal{E}_2}\!\Rrightarrow_{\mathcal{E}_3} R}{P \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_3} R}$$

VS-IMP
$$\frac{\Box\,(P \Rightarrow Q)}{P \Rrightarrow_{\emptyset} Q}$$

VS-MASK-FRAME
$$\frac{P \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} Q}{P \;\; {}_{\mathcal{E}_1 \uplus \mathcal{E}'}\!\Rrightarrow_{\mathcal{E}_2 \uplus \mathcal{E}'} Q}$$

VS-FRAME
$$\frac{P \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} Q}{P * R \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} Q * R}$$

VS-TIMELESS
$$\frac{\mathsf{timeless}(P)}{\triangleright P \Rrightarrow_{\emptyset} P}$$

VS-DISJ
$$\frac{P \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} R \qquad Q \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} R}{P \vee Q \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} R}$$

VS-EXIST
$$\frac{\forall x.\, (P \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} Q)}{(\exists x.\, P) \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} Q}$$

VS-ALWAYS
$$\frac{\Box Q \vdash P \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} R}{P \wedge \Box Q \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} R}$$

VS-FALSE
$$\mathsf{False} \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} P$$

9.3 Weakest Precondition

Finally, we can define the core piece of the program logic, the proposition that reasons about program behavior: Weakest precondition, from which Hoare triples will be derived.

Defining weakest precondition. We assume that everything making up the definition of the language, *i.e.,* values, expressions, states, the conversion functions, reduction relation and all their properties, are suitably reflected into the logic (*i.e.,* they are part of the signature $\mathcal{S}$). We further assume (as a parameter) a predicate $S : \mathit{State} \times \mathbb{N} \times \mathit{List}(\mathit{Obs}) \times \mathbb{N} \to \mathit{iProp}$ that interprets the machine state as an Iris proposition, a predicate $\Phi_F : \mathit{Val} \to \mathit{iProp}$ that serves as postcondition for forked-off threads, and a function $n_{\triangleright} : \mathbb{N} \to \mathbb{N}$ specifying the number of additional laters and later credits used for each physical step. The state interpretation can depend on the current physical state, the number of steps since the beginning of the execution, the list of *future* observations as well as the total number of *forked* threads (that is one less that the total number of threads). It should be monotone with respect to the step counter: $S(\sigma, n_s, \vec{\kappa}, n_t) \Rrightarrow_{\emptyset} S(\sigma, n_s + 1, \vec{\kappa}, n_t)$. This can be instantiated, for example, with ownership of an authoritative RA to tie the physical state to fragments that are used for user-level proofs. Finally, weakest precondition takes a parameter $s \in \{\mathsf{NotStuck}, \mathsf{Stuck}\}$ indicating whether program execution is allowed to get stuck.

$$\begin{aligned}
\mathit{wp}(S, \Phi_F, s) \triangleq\; & \mu \mathit{wp_rec}.\, \lambda \mathcal{E}, e, \Phi. \\
& (\exists v.\, \mathsf{expr_to_val}(e) = v \wedge \Rrightarrow_{\mathcal{E}} \Phi(v)) \vee \\
& \Big(\mathsf{expr_to_val}(e) = \bot \wedge \forall \sigma, n_s, \vec{\kappa}, \vec{\kappa}', n_t.\, S(\sigma, n_s, \vec{\kappa} +\!\!+ \vec{\kappa}', n_t) \;\; {}_{\mathcal{E}}\!\Rrightarrow\!\!\!\divideontimes_{\emptyset} \\
& \quad (s = \mathsf{NotStuck} \Rightarrow \mathsf{red}(e, \sigma)) * \forall e', \sigma', \vec{e}.\, (e, \sigma \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e', \sigma', \vec{e}) \mathrel{-\!\!*} \pounds\,(n_{\triangleright}(n_s) + 1) \mathrel{-\!\!*} \\
& \quad (\Rrightarrow_{\emptyset} \triangleright \Rrightarrow_{\emptyset})^{n_{\triangleright}(n_s)+1} {}_{\emptyset}\!\Rrightarrow_{\mathcal{E}} S(\sigma', n_s + 1, \vec{\kappa}', n + |\vec{e}|) * \mathit{wp_rec}(\mathcal{E}, e', \Phi) * \\
& \quad \divideontimes_{e'' \in \vec{e}} \mathit{wp_rec}(\top, e'', \Phi_F)\Big)
\end{aligned}$$

$$\mathsf{wp}^{S;\Phi_F}_{s;\mathcal{E}}\; e\; \{v.\, P\} \triangleq \mathit{wp}(S, \Phi_F, s)(\mathcal{E}, e, \lambda v.\, P)$$

The S and Φ_F will always be set by the context; typically, when instantiating Iris with a language, we also pick the corresponding state interpretation S and fork-postcondition Φ_F. All proof rules

leave S and Φ_F unchanged. If we leave away the mask $\mathcal{E}$, we assume it to default to $\top$. If we leave away the stuckness s, it defaults to $\mathsf{NotStuck}$.

Laws of weakest precondition. The following rules can all be derived:

WP-VALUE
$$P[v/x] \vdash \mathsf{wp}_{s;\mathcal{E}}\, v\, \{x.\, P\}$$

WP-MONO
$$\frac{\mathcal{E}_1 \subseteq \mathcal{E}_2 \qquad \Gamma, x : \mathsf{val} \mid P \vdash Q \qquad (s_2 = \mathsf{Stuck} \lor s_1 = s_2)}{\Gamma \mid \mathsf{wp}_{s_1;\mathcal{E}_1}\, e\, \{x.\, P\} \vdash \mathsf{wp}_{s_2;\mathcal{E}_2}\, e\, \{x.\, Q\}}$$

FUP-WP
$$\Rightarrow\!\!\!\!\!\!\Rightarrow_{\mathcal{E}} \mathsf{wp}_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{wp}_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

WP-FUP
$$\mathsf{wp}_{s;\mathcal{E}}\, e \left\{x.\, \Rightarrow\!\!\!\!\!\!\Rightarrow_{s;\mathcal{E}} P\right\} \vdash \mathsf{wp}_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

WP-ATOMIC
$$\frac{s = \mathsf{NotStuck} \Rightarrow \mathsf{atomic}(e) \qquad s = \mathsf{Stuck} \Rightarrow \mathsf{strongly\_atomic}(e)}{{}_{\mathcal{E}_1}\!\Rightarrow\!\!\!\!\!\!\Rightarrow_{\mathcal{E}_2} \mathsf{wp}_{s;\mathcal{E}_2}\, e \left\{x.\, {}_{\mathcal{E}_2}\!\Rightarrow\!\!\!\!\!\!\Rightarrow_{\mathcal{E}_1} P\right\} \vdash \mathsf{wp}_{s;\mathcal{E}_1}\, e\, \{x.\, P\}}$$

WP-FRAME
$$Q * \mathsf{wp}_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{wp}_{s;\mathcal{E}}\, e\, \{x.\, Q * P\}$$

WP-FRAME-STEP
$$\frac{\mathsf{expr\_to\_val}(e) = \bot \qquad \mathcal{E}_2 \subseteq \mathcal{E}_1}{\mathsf{wp}_{s;\mathcal{E}_2}\, e\, \{x.\, P\} * {}_{\mathcal{E}_1}\!\Rightarrow\!\!\!\!\!\!\Rightarrow_{\mathcal{E}_2} \triangleright\, {}_{\mathcal{E}_2}\!\Rightarrow\!\!\!\!\!\!\Rightarrow_{\mathcal{E}_1} Q \vdash \mathsf{wp}_{s;\mathcal{E}_1}\, e\, \{x.\, Q * P\}}$$

WP-FRAME-N-STEPS
$$\frac{\mathsf{expr\_to\_val}(e) = \bot \qquad \mathcal{E}_2 \subseteq \mathcal{E}_1}{\begin{array}{l}(\forall \sigma, n_s, \vec{\kappa}, n_t.\, S(\sigma, n_s, \vec{\kappa}, n_t) \Rrightarrow\!\!\!\!\!\!\ast_{\mathcal{E}_1,\emptyset}\, n \le n_\triangleright(n_s) + 1) \land \\ \mathsf{wp}_{s;\mathcal{E}_2}\, e\, \{x.\, P\} * {}_{\mathcal{E}_1}\!\Rightarrow\!\!\!\!\!\!\Rightarrow_{\mathcal{E}_2} (\triangleright \Rightarrow\!\!\!\!\!\!\Rightarrow_{\emptyset})^n\, {}_{\mathcal{E}_2}\!\Rightarrow\!\!\!\!\!\!\Rightarrow_{\mathcal{E}_1} Q \\ \vdash \mathsf{wp}_{s;\mathcal{E}_1}\, e\, \{x.\, Q * P\}\end{array}}$$

WP-BIND
$$\frac{K \text{ is a context}}{\mathsf{wp}_{s;\mathcal{E}}\, e \left\{x.\, \mathsf{wp}_{s;\mathcal{E}}\, K(\mathsf{val\_to\_expr}(x))\, \{y.\, P\}\right\} \vdash \mathsf{wp}_{s;\mathcal{E}}\, K(e)\, \{y.\, P\}}$$

We will also want a rule that connect weakest preconditions to the operational semantics of the language. This basically just copies the second branch (the non-value case) of the definition of weakest preconditions.

WP-LIFT-STEP
$$\frac{\mathsf{expr\_to\_val}(e_1) = \bot}{\begin{array}{l}\forall \sigma_1, \vec{\kappa}, \vec{\kappa}', n.\, S(\sigma_1, n_s, \vec{\kappa} +\!\!+ \vec{\kappa}', n_t) \;{}_{\mathcal{E}}\!\Rrightarrow\!\!\!\!\!\!\ast_{\emptyset}\, (s = \mathsf{NotStuck} \Rightarrow \mathsf{red}(e_1, \sigma_1)) * \\ \quad \forall e_2, \sigma_2, \vec{e}.\, (e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e_2, \sigma_2, \vec{e}) \mathrel{-\!\!*} £\, (n_\triangleright(n_s) + 1) \mathrel{-\!\!*} (\Rightarrow\!\!\!\!\!\!\Rightarrow_{\emptyset} \triangleright \Rightarrow\!\!\!\!\!\!\Rightarrow_{\emptyset})^{n_\triangleright(n_s)}\, {}_{\emptyset}\!\Rightarrow\!\!\!\!\!\!\Rightarrow_{\mathcal{E}} \\ \qquad \left(S(\sigma_2, n_s + 1, \vec{\kappa}', n_t + |\vec{e}|) * \mathsf{wp}^{S;\Phi_F}_{s;\mathcal{E}}\, e_2\, \{x.\, P\} * \bigstar_{e_{\mathsf{f}} \in \vec{e}} \mathsf{wp}^{S\Phi_F}_{s;\top}\, e_{\mathsf{f}}\, \{\Phi_F\}\right) \\ \vdash \mathsf{wp}^{S\Phi_F}_{s;\mathcal{E}}\, e_1\, \{x.\, P\}\end{array}}$$

Adequacy of weakest precondition. The purpose of the adequacy statement is to show that our notion of weakest preconditions is *realistic* in the sense that it actually has anything to do with

the actual behavior of the program. The most general form of the adequacy statement is about proving properties of an arbitrary program execution.

Theorem 3 (Adequacy)**.** *Assume we are given some* $\vec{e}_1$*,* σ_1*,* $\vec{\kappa}$*,* T_2*,* σ_2 *such that* $(\vec{e}_1, \sigma_1) \xrightarrow{\vec{\kappa}}{}^{*}_{\mathsf{tp}} (T_2, \sigma_2)$*. Moreover, assume we are given a stuckness parameter* s *and* meta-level *property* p *that we want to show. To verify that* p *holds, it is sufficient to show the following Iris entailment:*

$$\mathsf{True} \vdash \Rrightarrow_{\top} \exists S, \vec{\Phi}, \Phi_F.\, S(\sigma_1, 0, \vec{\kappa}, 0) * \left(\mathop{\Large *}_{e, \Phi \in \vec{e}_1, \vec{\Phi}} \mathsf{wp}^{S;\Phi_F}_{s;\top}\, e\, \{x.\, \Phi(x)\} \right) * \left(C^{S;\vec{\Phi};\Phi_F}_s(T_2, \sigma_2) \;\; {}_{\top}\Rrightarrow_{\emptyset} \hat{p} \right)$$

where C *describes states that are consistent with the state interpretation and postconditions:*

$$\begin{aligned} C^{S;\vec{\Phi};\Phi_F}_s(T_2, \sigma_2) \triangleq \exists \vec{e}_2, T_2'.\, & T_2 = \vec{e}_2 +\!\!+ T_2' * \\ & |\vec{e}_1| = |\vec{e}_2| * \\ & (s = \mathsf{NotStuck} \Rightarrow \forall e \in T_2.\, \mathsf{expr\_to\_val}(e) \neq \bot \vee \mathsf{red}(e, \sigma_2)) * \\ & S(\sigma_2, (), |T_2'|) * \\ & \left(\mathop{\Large *}_{e, \Phi \in \vec{e}_2, \vec{\Phi}} \mathsf{expr\_to\_val}(e) \neq \bot \mathrel{-\!\!*} \Phi(\mathsf{expr\_to\_val}(e)) \right) * \\ & \left(\mathop{\Large *}_{e \in T_2'} \mathsf{expr\_to\_val}(e) \neq \bot \mathrel{-\!\!*} \Phi_F(\mathsf{expr\_to\_val}(e)) \right) \end{aligned}$$

The $\hat{p}$ *here arises because we need a way to talk about* p *inside Iris. To this end, we assume that the signature* $\mathcal{S}$ *contains some assertion* $\hat{p}$*:*

$$\hat{p} : \mathit{iProp} \in \mathcal{F}$$

Furthermore, we assume that the interpretation $[\![\hat{p}]\!]$ *of* $\hat{p}$ *reflects* p *(also see §6):*

$$\begin{aligned} [\![\hat{p}]\!] \;&:\; [\![\mathit{iProp}]\!] \\ [\![\hat{p}]\!] &\triangleq \lambda\_.\, \{n \mid p\} \end{aligned}$$

The signature can of course state arbitrary additional properties of $\hat{p}$*, as long as they are proven sound.*

In other words, to show that p holds, we have to prove an entailment in Iris that, starting from the empty context, chooses some state interpretation, postconditions for the initial threads, forked-thread postcondition and stuckness and then proves:

- the initial state interpretation,
- a weakest-precondition,
- and a view shift showing the desired $\hat{p}$ under the extra assumption $C(T_2, \sigma_2)$.

Notice that the state interpretation and the postconditions are chosen *after* doing a fancy update, which allows them to depend on the names of ghost variables that are picked in that initial fancy update. This gives us a chance to allocate some "global" ghost state that state interpretation and postcondition can refer to (*e.g.,* the name $\gamma_{Credits}$).

$C_s^{S;\vec{\Phi};\Phi_F}(T_2, \sigma_2)$ says that:

- The final thread-pool T_2 contains the final state of the initial threads $\vec{e}_2$, and any number of additional forked threads in T_2'.
- If this is a stuck-free weakest precondition, then all threads in the final thread-pool are either values or are reducible in the final state σ_2.
- The state interpretation S holds for the final state.
- If one of the initial threads reduced to a value, the corresponding post-condition $\Phi \in \vec{\Phi}$ holds for that value.
- If any other thread reduced to a value, the forked-thread post-condition Φ_F holds for that value.

As an example for how to use this adequacy theorem, let us say we wanted to prove that a program e_1 for which we derived a $\mathsf{NotStuck}$ weakest-precondition cannot get stuck:

Corollary 1 (Stuck-freedom)**.** *Assume we are given some e_1 such that the following holds:*

$$\mathsf{True} \vdash \forall \sigma_1, \vec{\kappa}.\ \Rrightarrow_\top \exists S, \Phi, \Phi_F.\ S(\sigma_1, 0, \vec{\kappa}, 0) * \mathsf{wp}^{S;\Phi_F}_{\mathsf{NotStuck};\top}\ e_1\ \{x.\ \Phi(x)\}$$

Then it is the case that:

$$\forall \sigma_1, \vec{\kappa}, T_2, \sigma_2.\ ([e_1], \sigma_1) \xrightarrow{\vec{\kappa}}{}^*_{\mathsf{tp}}\ (T_2, \sigma_2) \Rightarrow \forall e \in T_2.\ \mathsf{expr\_to\_val}(e) \neq \bot \vee \mathsf{red}(e, \sigma_2)$$

To prove the conclusion of this corollary, we assume some $\sigma_1, \vec{\kappa}, T_2, \sigma_2$ and $([e_1], \sigma_1) \xrightarrow{\vec{\kappa}}{}^*_{\mathsf{tp}}\ (T_2, \sigma_2)$, and we instantiate the main theorem with this execution and $p \triangleq \forall e \in T_2.\ \mathsf{expr\_to\_val}(e) \neq \bot \vee \mathsf{red}(e, \sigma_2)$. We can then show the premise of adequacy using the Iris entailment that we assumed in the corollary and:

$$\mathsf{True} \vdash C^{S;[\Phi];\Phi_F}_{\mathsf{NotStuck}}(T_2, \sigma_2)\ \ {}_\top\Rrightarrow_\emptyset\ p$$

This proof, just like the following, also exploits that we can freely swap between meta-level universal quantification ($\forall x.\ \mathsf{True} \vdash P$) and quantification in Iris ($\mathsf{True} \vdash \forall x.\ P$).

Similarly we could show that the postcondition makes adequate statements about the possible final value of the main thread:

Corollary 2 (Adequate postcondition)**.** *Assume we are given some e_1 and a set $V \subseteq Val$ such that the following holds (assuming we can talk about sets like V inside the logic):*

$$\mathsf{True} \vdash \forall \sigma_1, \vec{\kappa}.\ \Rrightarrow_\top \exists S, \Phi_F.\ S(\sigma_1, 0, \vec{\kappa}, 0) * \mathsf{wp}^{S;\Phi_F}_{s;\top}\ e_1\ \{x.\ x \in V\}$$

Then it is the case that:

$$\forall \sigma_1, \vec{\kappa}, v_2, T_2, \sigma_2.\ ([e_1], \sigma_1) \xrightarrow{\vec{\kappa}}{}^*_{\mathsf{tp}}\ ([\mathsf{val\_to\_expr}(v_2)] +\!\!+ T_2, \sigma_2) \Rightarrow v_2 \in V$$

To show this, we assume some $\sigma_1, \vec{\kappa}, v_2, T_2, \sigma_2$ such that $([e_1], \sigma_1) \xrightarrow{\vec{\kappa}}{}^*_{\mathsf{tp}} ([\mathsf{val\_to\_expr}(v_2)] +\!\!+ T_2, \sigma_2)$, and we instantiate adequacy with this execution and $p \triangleq v_2 \in \mathit{Val}$. Then we only have to show:

$$\mathsf{True} \vdash C_s^{S;[(\lambda v.\ v \in \mathit{Val})];\Phi_F}([\mathsf{val\_to\_expr}(v_2)] +\!\!+ T_2, \sigma_2) \ \ {}_\top\Rrightarrow_\emptyset\ v_2 \in \mathit{Val}$$

As a final example, we could use adequacy to show that the state σ of the program is always in some set $\Sigma \subseteq \mathit{State}$:

Corollary 3 (Adequate state interpretation)**.** *Assume we are given some e_1 and a set $\Sigma \subseteq \mathit{State}$ such that the following holds (assuming we can talk about sets like Σ inside the logic):*

$$\mathsf{True} \vdash \forall \sigma_1, \vec{\kappa}.\ \Rrightarrow_\top \exists S, \Phi, \Phi_F.\ S(\sigma_1, 0, \vec{\kappa}, 0) * \mathsf{wp}^{S;\Phi_F}_{s;\top}\ e_1\ \{\Phi\} * (\forall \sigma_2, n_s, n_t.\ S(\sigma_2, n_s, (), n_t) \ \ {}_\top\Rrightarrow_\emptyset\ \sigma_2 \in \Sigma)$$

Then it is the case that:

$$\forall \sigma_1, \vec{\kappa}, T_2, \sigma_2.\ ([e_1], \sigma_1) \xrightarrow{\vec{\kappa}}{}^*_{\mathsf{tp}} (T_2, \sigma_2) \Rightarrow \sigma_2 \in \Sigma$$

To show this, we assume some $\sigma_1, \vec{\kappa}, T_2, \sigma_2$ such that $([e_1], \sigma_1) \xrightarrow{\vec{\kappa}}{}^*_{\mathsf{tp}} (T_2, \sigma_2)$, and we instantiate adequacy with this execution and $p \triangleq \sigma_2 \in \Sigma$. Then we have to show:

$$(\forall \sigma_2, n_s, n_t.\ S(\sigma_2, n_s, (), n_t) \ \ {}_\top\Rrightarrow_\emptyset\ \sigma_2 \in \Sigma) \vdash C_s^{S;[\Phi];\Phi_F}(T_2, \sigma_2) \ \ {}_\top\Rrightarrow_\emptyset\ \sigma_2 \in \Sigma$$

Hoare triples. It turns out that weakest precondition is actually quite convenient to work with, in particular when performing these proofs in Coq. Still, for a more traditional presentation, we can easily derive the notion of a Hoare triple:

$$\{P\}\, e\, \{v.\, Q\}_{\mathcal{E}} \triangleq \Box\,(P \mathrel{-\!\!*} \mathsf{wp}_{\mathcal{E}}\, e\, \{v.\, Q\})$$

We assume the state interpretation S to be fixed by the context.

We only give some of the proof rules for Hoare triples here, since we usually do all our reasoning directly with weakest preconditions and use Hoare triples only to write specifications.

HT-RET
$$\{\mathsf{True}\}\, w\, \{v.\, v = w\}_{\mathcal{E}}$$

HT-BIND
$$\frac{K \text{ is a context} \qquad \{P\}\, e\, \{v.\, Q\}_{\mathcal{E}} \qquad \forall v.\, \{Q\}\, K(v)\, \{w.\, R\}_{\mathcal{E}}}{\{P\}\, K(e)\, \{w.\, R\}_{\mathcal{E}}}$$

HT-CSQ
$$\frac{P \Rrightarrow P' \qquad \{P'\}\, e\, \{v.\, Q'\}_{\mathcal{E}} \qquad \forall v.\, Q' \Rrightarrow Q}{\{P\}\, e\, \{v.\, Q\}_{\mathcal{E}}}$$

HT-FRAME
$$\frac{\{P\}\, e\, \{v.\, Q\}_{\mathcal{E}}}{\{P * R\}\, e\, \{v.\, Q * R\}_{\mathcal{E}}}$$

HT-ATOMIC
$$\frac{P \ \ {}_{\mathcal{E}\uplus\mathcal{E}'}\Rrightarrow_{\mathcal{E}} P' \qquad \{P'\}\, e\, \{v.\, Q'\}_{\mathcal{E}} \qquad \forall v.\, Q' \ \ {}_{\mathcal{E}}\Rrightarrow_{\mathcal{E}\uplus\mathcal{E}'} Q \qquad \mathsf{atomic}(e)}{\{P\}\, e\, \{v.\, Q\}_{\mathcal{E}\uplus\mathcal{E}'}}$$

HT-FALSE
$$\{\mathsf{False}\}\, e\, \{v.\, P\}_{\mathcal{E}}$$

HT-DISJ
$$\frac{\{P\}\, e\, \{v.\, R\}_{\mathcal{E}} \qquad \{Q\}\, e\, \{v.\, R\}_{\mathcal{E}}}{\{P \vee Q\}\, e\, \{v.\, R\}_{\mathcal{E}}}$$

HT-EXIST
$$\frac{\forall x.\, \{P\}\, e\, \{v.\, Q\}_{\mathcal{E}}}{\{\exists x.\, P\}\, e\, \{v.\, Q\}_{\mathcal{E}}}$$

HT-BOX
$$\frac{\Box\, Q \vdash \{P\}\, e\, \{v.\, R\}_{\mathcal{E}}}{\{P \wedge \Box\, Q\}\, e\, \{v.\, R\}_{\mathcal{E}}}$$

9.4 Invariant Namespaces

In §9.2, we defined a proposition $\mathsf{Inv}^{\iota}(P)$ expressing knowledge (*i.e.,* the proposition is persistent) that P is maintained as invariant with name ι. The concrete name ι is picked when the invariant is allocated, so it cannot possibly be statically known – it will always be a variable that's threaded through everything. However, we hardly care about the actual, concrete name. All we need to know is that this name is *different* from the names of other invariants that we want to open at the same time. Keeping track of the n^2 mutual inequalities that arise with n invariants quickly gets in the way of the actual proof.

To solve this issue, instead of remembering the exact name picked for an invariant, we will keep track of the *namespace* the invariant was allocated in. Namespaces are sets of invariants, following a tree-like structure: Think of the name of an invariant as a sequence of identifiers, much like a fully qualified Java class name. A *namespace* $\mathcal{N}$ then is like a Java package: it is a sequence of identifiers that we think of as *containing* all invariant names that begin with this sequence. For example, `org.mpi-sws.iris` is a namespace containing the invariant name `org.mpi-sws.iris.heap`.

The crux is that all namespaces contain infinitely many invariants, and hence we can *freely pick* the namespace an invariant is allocated in – no further, unpredictable choice has to be made. Furthermore, we will often know that namespaces are *disjoint* just by looking at them. The namespaces $\mathcal{N}$.`iris` and $\mathcal{N}$.`gps` are disjoint no matter the choice of $\mathcal{N}$. As a result, there is often no need to track disjointness of namespaces, we just have to pick the namespaces that we allocate our invariants in accordingly.

Formally speaking, let $\mathcal{N} \in \mathsf{InvNamesp} \triangleq \mathit{List}(\mathbb{N})$ be the type of *invariant namespaces*. We use the notation $\mathcal{N}.\iota$ for the namespace $[\iota] +\!\!+ \mathcal{N}$. (In other words, the list is "backwards". This is because cons-ing to the list, like the dot does above, is easier to deal with in Coq than appending at the end.)

The elements of a namespaces are *structured invariant names* (think: Java fully qualified class name). They, too, are lists of $\mathbb{N}$, the same type as namespaces. In order to connect this up to the definitions of §9.2, we need a way to map structured invariant names to *InvName*, the type of "plain" invariant names. Any injective mapping $\mathsf{namesp\_inj}$ will do; and such a mapping has to exist because $\mathit{List}(\mathbb{N})$ is countable and *InvName* is infinite. Whenever needed, we (usually implicitly) coerce $\mathcal{N}$ to its encoded suffix-closure, *i.e.,* to the set of encoded structured invariant names contained in the namespace:

$$\mathcal{N}^{\uparrow} \triangleq \{\iota \mid \exists \mathcal{N}'.\, \iota = \mathsf{namesp\_inj}(\mathcal{N}' +\!\!+ \mathcal{N})\}$$

We will overload the notation for invariant propositions for using namespaces instead of names:

$$\mathsf{Inv}^{\mathcal{N}}(P) \triangleq \exists \iota \in \mathcal{N}^{\uparrow}.\, \mathsf{Inv}^{\iota}(P)$$

We can now derive the following rules (this involves unfolding the definition of fancy updates):

INV-PERSIST
$$\mathsf{Inv}^{\mathcal{N}}(P) \vdash \Box\, \mathsf{Inv}^{\mathcal{N}}(P)$$

INV-ALLOC
$$\triangleright P \vdash \Rrightarrow_{\emptyset} \mathsf{Inv}^{\mathcal{N}}(P)$$

INV-OPEN
$$\frac{\mathcal{N} \subseteq \mathcal{E}}{\mathsf{Inv}^{\mathcal{N}}(P) \;\; {}_{\mathcal{E}}\!\Rrightarrow_{\mathcal{E}\setminus\mathcal{N}} \triangleright P * (\triangleright P \;{}_{\mathcal{E}\setminus\mathcal{N}}\!\Rrightarrow\!\!\divideontimes_{\mathcal{E}}\; \mathsf{True})}$$

INV-OPEN-TIMELESS
$$\frac{\mathcal{N} \subseteq \mathcal{E} \qquad \mathsf{timeless}(P)}{\mathsf{Inv}^{\mathcal{N}}(P) \;\; {}_{\mathcal{E}}\!\Rrightarrow_{\mathcal{E}\setminus\mathcal{N}} P * (P \;{}_{\mathcal{E}\setminus\mathcal{N}}\!\Rrightarrow\!\!\divideontimes_{\mathcal{E}}\; \mathsf{True})}$$

9.5 Accessors

The two rules INV-OPEN and INV-OPEN-TIMELESS above may look a little surprising, in the sense that it is not clear on first sight how they would be applied. The rules are the first *accessors* that show up in this document. Accessors are propositions of the form

$$P \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} \exists x.\, Q * (\forall y.\, Q' \;\; {}_{\mathcal{E}_2}\!\Rrightarrow\!\!\!\!\!\!\ast_{\mathcal{E}_1} R)$$

One way to think about such propositions is as follows: Given some accessor, if during our verification we have the proposition P and the mask $\mathcal{E}_1$ available, we can use the accessor to *access* Q and obtain the witness x. We call this *opening* the accessor, and it changes the mask to $\mathcal{E}_2$. Additionally, opening the accessor provides us with $\forall y.\, Q' \;\; {}_{\mathcal{E}_2}\!\Rrightarrow\!\!\!\!\!\!\ast_{\mathcal{E}_1} R$, a *linear view shift* (*i.e.,* a view shift that can only be used once). This linear view shift tells us that in order to *close* the accessor again and go back to mask $\mathcal{E}_1$, we have to pick some y and establish the corresponding Q'. After closing, we will obtain R.

Using VS-TRANS and HT-ATOMIC (or the corresponding proof rules for fancy updates and weakest preconditions), we can show that it is possible to open an accessor around any view shift and any *atomic* expression:

$$\textsc{Acc-vs}\quad \frac{P \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} \exists x.\, Q * (\forall y.\, Q' \;\; {}_{\mathcal{E}_2}\!\Rrightarrow\!\!\!\!\!\!\ast_{\mathcal{E}_1} R) \qquad \forall x.\, Q * P_F \Rrightarrow_{\mathcal{E}_2} \exists y.\, Q' * P_F}{P * P_F \Rrightarrow_{\mathcal{E}_1} R * P_F}$$

$$\textsc{Acc-Ht}\quad \frac{P \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} \exists x.\, Q * (\forall y.\, Q' \;\; {}_{\mathcal{E}_2}\!\Rrightarrow\!\!\!\!\!\!\ast_{\mathcal{E}_1} R) \qquad \forall x.\, \{Q * P_F\}\, e\, \{\exists y.\, Q' * P_F\}_{\mathcal{E}_2} \qquad \mathsf{atomic}(e)}{\{P * P_F\}\, e\, \{R * P_F\}_{\mathcal{E}_1}}$$

Furthermore, in the special case that $\mathcal{E}_1 = \mathcal{E}_2$, the accessor can be opened around *any* expression. For this reason, we also call such accessors *non-atomic*.

The reasons accessors are useful is that they let us talk about "opening X" (*e.g.,* "opening invariants") without having to care what X is opened around. Furthermore, as we construct more sophisticated and more interesting things that can be opened (*e.g.,* invariants that can be "cancelled", or STSs), accessors become a useful interface that allows us to mix and match different abstractions in arbitrary ways.

For the symmetric case where $P = R$ and $Q = Q'$, we use the following notation that avoids repetition:

$$P \;\; {}_{\mathcal{E}_1}\!\propto_{\mathcal{E}_2} \;\; x.\, Q \triangleq P \;\; {}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} \exists x.\, Q * (Q \;\; {}_{\mathcal{E}_2}\!\Rrightarrow\!\!\!\!\!\!\ast_{\mathcal{E}_1} P)$$

This accessor is "idempotent" in the sense that it does not actually change the state. After applying it, we get our P back so we end up where we started.

Accessor-style invariants. In fact, the user-visible notion of invariants $\boxed{P}^{\mathcal{N}}$ is defined via INV-OPEN:

$$\boxed{P}^{\mathcal{N}} \triangleq \Box \forall \mathcal{E}.\; {}_{\mathcal{E}}\!\models\!\!\!\Rrightarrow_{\mathcal{E}\setminus\mathcal{N}} \triangleright P * (\triangleright P \;\; {}_{\mathcal{E}\setminus\mathcal{N}}\!\Rrightarrow\!\!\!\!\!\!\ast_{\mathcal{E}} \mathsf{True})$$

All the invariant laws shown above for $\mathsf{Inv}^{\mathcal{N}}(P)$ also hold for $\boxed{P}^{\mathcal{N}}$, but we can also show some additional laws that would otherwise not hold:

INV-COMBINE
$$\frac{\mathcal{N}_1 \# \mathcal{N}_2 \qquad \mathcal{N}_1 \cup \mathcal{N}_2 \subseteq \mathcal{N}}{\boxed{P_1}^{\mathcal{N}_1} * \boxed{P_2}^{\mathcal{N}_2} \vdash \boxed{P_1 * P_2}^{\mathcal{N}}}$$

INV-SPLIT
$$\boxed{P_1 * P_2}^{\mathcal{N}} \vdash \boxed{P_1}^{\mathcal{N}} * \boxed{P_2}^{\mathcal{N}}$$

INV-ALTER
$$\triangleright \Box(P \mathrel{-\!\!*} Q * (Q \mathrel{-\!\!*} P)) \vdash \boxed{P}^{\mathcal{N}} \mathrel{-\!\!*} \boxed{Q}^{\mathcal{N}}$$

10 Derived Constructions

10.1 Cancellable Invariants

Iris invariants as described in §9.2 are persistent—once established, they hold forever. However, based on them, it is possible to *encode* a form of invariants that can be "cancelled" again.

First, we need some ghost state:

$$\mathit{CInvTok} \triangleq \mathit{Frac}$$

Now we define:

$$[\mathsf{CInv} : \gamma]_q \triangleq \boxed{q}^{\gamma}$$
$$\mathsf{CInv}^{\gamma,\mathcal{N}}(P) \triangleq \boxed{P \vee \boxed{1}^{\gamma}}^{\mathcal{N}}$$

It is then straightforward to prove:

CInv-new
$$\triangleright P \Rrightarrow_{\bot} \exists \gamma.\, [\mathsf{CInv} : \gamma]_1 * \square\, \mathsf{CInv}^{\gamma,\mathcal{N}}(P)$$

CInv-acc
$$\mathsf{CInv}^{\gamma,\mathcal{N}}(P) \vdash [\mathsf{CInv} : \gamma]_q \quad {}_{\mathcal{N}}\!\propto_{\emptyset} \quad \triangleright P$$

CInv-cancel
$$\mathsf{CInv}^{\gamma,\mathcal{N}}(P) \vdash [\mathsf{CInv} : \gamma]_1 \Rrightarrow_{\mathcal{N}} \triangleright P$$

Cancellable invariants are useful, for example, when reasoning about data structures that will be deallocated: Every reference to the data structure comes with a fraction of the token, and when all fractions have been gathered, CInv-cancel is used to cancel the invariant, after which the data structure can be deallocated.

10.2 Non-atomic ("Thread-Local") Invariants

Sometimes it is necessary to maintain invariants that we need to open non-atomically. Clearly, for this mechanism to be sound we need something that prevents us from opening the same invariant twice, something like the masks that avoid reentrancy on the "normal", atomic invariants. The idea is to use tokens[7] that guard access to non-atomic invariants. Having the token $[\mathsf{NaInv} : p.\mathcal{E}]$ indicates that we can open all invariants in $\mathcal{E}$. The p here is the name of the *invariant pool*. This mechanism allows us to have multiple, independent pools of invariants that all have their own namespaces.

One way to think about non-atomic invariants is as "thread-local invariants", where every pool is a thread. Every thread thus has its own, independent set of invariants. Every thread threads through all the tokens for its own pool, so that each invariant can only be opened in the thread it belongs to. As a consequence, they can be kept open around any sequence of expressions (*i.e.,* there is no restriction to atomic expressions) – after all, there cannot be any races with other threads.

Concretely, this is the monoid structure we need:

$$\mathit{PId} \triangleq \mathit{GName}$$
$$\mathit{NaTok} \triangleq \wp^{\mathsf{fin}}(\mathit{InvName}) \times \wp(\mathit{InvName})$$

[7] Very much like the tokens that are used to encode "normal", atomic invariants

For every pool, there is a set of tokens designating which invariants are *enabled* (closed). This corresponds to the mask of "normal" invariants. We re-use the structure given by namespaces for non-atomic invariants. Furthermore, there is a *finite* set of invariants that is *disabled* (open).

Owning tokens is defined as follows:

$$[\mathsf{NaInv} : p.\mathcal{E}] \triangleq \boxed{(\emptyset, \mathcal{E})}^{p}$$
$$[\mathsf{NaInv} : p] \triangleq [\mathsf{NaInv} : p.\top]$$

Next, we define non-atomic invariants. To simplify this construction,we piggy-back into "normal" invariants.

$$\mathsf{NaInv}^{p.\mathcal{N}}(P) \triangleq \exists \iota \in \mathcal{N}. \boxed{P * \boxed{(\{\iota\}, \emptyset)}^{p} \lor [\mathsf{NaInv} : p.\{\iota\}]}^{\mathcal{N}}$$

We easily obtain:

NAInv-new-pool
$\mathsf{True} \Rrightarrow_{\bot} \exists p.\, [\mathsf{NaInv} : p]$

NAInv-tok-split
$[\mathsf{NaInv} : p.\mathcal{E}_1 \uplus \mathcal{E}_2] \Leftrightarrow [\mathsf{NaInv} : p.\mathcal{E}_1] * [\mathsf{NaInv} : p.\mathcal{E}_2]$

NAInv-new-inv
$\triangleright P \Rrightarrow_{\mathcal{N}} \Box\, \mathsf{NaInv}^{p.\mathcal{N}}(P)$

NAInv-acc
$\mathsf{NaInv}^{p.\mathcal{N}}(P) \vdash [\mathsf{NaInv} : p.\mathcal{N}] \;\; \propto_{\mathcal{N}} \;\; \triangleright P$

from which we can derive

$$\frac{\mathcal{N} \subseteq \mathcal{E}}{\mathsf{NaInv}^{p.\mathcal{N}}(P) \vdash [\mathsf{NaInv} : p.\mathcal{E}] \;\; \propto_{\mathcal{N}} \;\; \triangleright P * [\mathsf{NaInv} : p.\mathcal{E} \setminus \mathcal{N}]}$$

10.3 Boxes

The idea behind the *boxes* is to have a proposition P that is actually split into a number of pieces, each of which can be taken out and back in separately. In some sense, this is a replacement for having an "authoritative PCM of Iris propositions itself". It is similar to the pattern involving saved propositions that was used for the barrier [5], but more complicated because there are some operations that we want to perform without a later.

Roughly, the idea is that a *box* is a container for a proposition P. A box consists of a bunch of *slices* which decompose P into a separating conjunction of the propositions Q_ι governed by the individual slices. Each slice is either *full* (it right now contains Q_ι), or *empty* (it does not contain anything currently). The proposition governing the box keeps track of the state of all the slices that make up the box. The crux is that opening and closing of a slice can be done even if we only have ownership of the boxes "later" ($\triangleright$).

The interface for boxes is as follows: The two core propositions are: $\mathsf{BoxSlice}(\mathcal{N}, P, \iota)$, saying that there is a slice in namespace $\mathcal{N}$ with name ι and content P; and $\mathsf{Box}(\mathcal{N}, P, f)$, saying that f describes the slices of a box in namespace $\mathcal{N}$, such that all the slices together contain P. Here, f is

of type $\mathbb{N} \xrightarrow{\text{fin}} \mathit{BoxState}$ mapping names to states, where $\mathit{BoxState} \triangleq \{\mathsf{full}, \mathsf{empty}\}$.

$$\textsc{Box-create}\quad \mathsf{True} \Rrightarrow_{\mathcal{N}} \mathsf{Box}(\mathcal{N}, \mathsf{True}, \emptyset)$$

$$\textsc{Slice-insert-empty}\quad \triangleright^b \mathsf{Box}(\mathcal{N}, P, f) \Rrightarrow_{\mathcal{N}} \exists \iota \notin \mathsf{dom}(f).\ \square\, \mathsf{BoxSlice}(\mathcal{N}, Q, \iota) * \triangleright^b \mathsf{Box}(\mathcal{N}, P * Q, f[\iota \leftarrow \mathsf{empty}])$$

$$\textsc{Slice-delete-empty}\quad \frac{f(\iota) = \mathsf{empty}}{\mathsf{BoxSlice}(\mathcal{N}, Q, \iota) \vdash \triangleright^b \mathsf{Box}(\mathcal{N}, P, f) \Rrightarrow_{\mathcal{N}} \exists P'.\ \triangleright^b(\triangleright(P = P' * Q) * \mathsf{Box}(\mathcal{N}, P', f[\iota \leftarrow \bot]))}$$

$$\textsc{Slice-fill}\quad \frac{f(\iota) = \mathsf{empty}}{\mathsf{BoxSlice}(\mathcal{N}, Q, \iota) \vdash \triangleright^b Q * \triangleright \mathsf{Box}(\mathcal{N}, P, f) \Rrightarrow_{\mathcal{N}} \triangleright^b \mathsf{Box}(\mathcal{N}, P, f[\iota \leftarrow \mathsf{full}])}$$

$$\textsc{Slice-empty}\quad \frac{f(\iota) = \mathsf{full}}{\mathsf{BoxSlice}(\mathcal{N}, Q, \iota) \vdash \triangleright^b \mathsf{Box}(\mathcal{N}, P, f) \Rrightarrow_{\mathcal{N}} \triangleright Q * \triangleright^b \mathsf{Box}(\mathcal{N}, P, f[\iota \leftarrow \mathsf{empty}])}$$

$$\textsc{Box-fill}\quad \frac{\forall \iota \in \mathsf{dom}(f).\ f(\iota) = \mathsf{empty}}{\triangleright P * \mathsf{Box}(\mathcal{N}, P, f) \Rrightarrow_{\mathcal{N}} \mathsf{Box}(\mathcal{N}, P, f[\iota \leftarrow \mathsf{full} \mid \iota \in \mathsf{dom}(f)])}$$

$$\textsc{Box-empty}\quad \frac{\forall \iota \in \mathsf{dom}(f).\ f(\iota) = \mathsf{full}}{\mathsf{Box}(\mathcal{N}, P, f) \Rrightarrow_{\mathcal{N}} \triangleright P * \mathsf{Box}(\mathcal{N}, P, f[\iota \leftarrow \mathsf{empty} \mid \iota \in \mathsf{dom}(f)])}$$

Above, $\triangleright^b P$ is syntactic sugar for $\triangleright P$ (if b is 1) or P (if b is 0). This is essentially an *optional later*, indicating that the lemmas can be applied with Box being owned now or later, and that ownership is returned the same way.

Model. The above rules are validated by the following model. We need a camera as follows:

$$\mathit{BoxState} \triangleq \mathsf{full} + \mathsf{empty}$$
$$\mathit{Box} \triangleq \mathit{Auth}(\mathit{Ex}(\mathit{BoxState})^?) \times \mathit{Ag}(\blacktriangleright \mathit{iProp})^?$$

Now we can define the propositions:

$$\mathsf{SliceInv}(\iota, P) \triangleq \exists b.\ \boxed{(\bullet b, \varepsilon)}^{\iota} * ((b = \mathsf{full}) \Rightarrow P)$$
$$\mathsf{BoxSlice}(\mathcal{N}, P, \iota) \triangleq \boxed{(\varepsilon, P)}^{\iota} * \boxed{\mathsf{SliceInv}(\iota, P)}^{\mathcal{N}}$$
$$\mathsf{Box}(\mathcal{N}, P, f) \triangleq \exists Q : \mathbb{N} \to \mathit{iProp}.\ \triangleright \left(P = \mathop{\Large *}_{\iota \in \mathsf{dom}(f)} Q(\iota) \right) * \mathop{\Large *}_{\iota \in \mathsf{dom}(f)} \boxed{(\circ f(\iota), Q(\iota))}^{\iota} * \boxed{\mathsf{SliceInv}(\iota, Q(\iota))}^{\mathcal{N}}$$

Derived rules. Here are some derived rules:

SLICE-INSERT-FULL

$$\triangleright Q * \triangleright^b \mathsf{Box}(\mathcal{N}, P, f) \Rrightarrow_{\mathcal{N}} \exists \iota \notin \mathsf{dom}(f).\ \square\, \mathsf{BoxSlice}(\mathcal{N}, Q, \iota) * \triangleright^b \mathsf{Box}(\mathcal{N}, P * Q, f[\iota \leftarrow \mathsf{full}])$$

SLICE-DELETE-FULL

$$\frac{f(\iota) = \mathsf{full}}{\mathsf{BoxSlice}(\mathcal{N}, Q, \iota) \vdash \triangleright^b \mathsf{Box}(\mathcal{N}, P, f) \Rrightarrow_{\mathcal{N}} \triangleright Q * \exists P'.\ \triangleright^b(\triangleright(P = P' * Q) * \mathsf{Box}(\mathcal{N}, P', f[\iota \leftarrow \bot]))}$$

SLICE-SPLIT

$$\frac{f(\iota) = s}{\begin{array}{l}\mathsf{BoxSlice}(\mathcal{N}, Q_1 * Q_2, \iota) \vdash \triangleright^b \mathsf{Box}(\mathcal{N}, P, f) \Rrightarrow_{\mathcal{N}} \exists \iota_1 \notin \mathsf{dom}(f), \iota_2 \notin \mathsf{dom}(f).\ \iota_1 \neq \iota_2 \wedge \\ \quad \square\, \mathsf{BoxSlice}(\mathcal{N}, Q_1, \iota_1) * \square\, \mathsf{BoxSlice}(\mathcal{N}, Q_2, \iota_2) * \triangleright^b \mathsf{Box}(\mathcal{N}, P, f[\iota \leftarrow \bot][\iota_1 \leftarrow s][\iota_2 \leftarrow s])\end{array}}$$

SLICE-MERGE

$$\frac{\iota_1 \neq \iota_2 \qquad f(\iota_1) = f(\iota_2) = s}{\begin{array}{l}\mathsf{BoxSlice}(\mathcal{N}, Q_1, \iota_1), \mathsf{BoxSlice}(\mathcal{N}, Q_2, \iota_2) \vdash \triangleright^b \mathsf{Box}(\mathcal{N}, P, f) \Rrightarrow_{\mathcal{N}} \exists \iota \notin \mathsf{dom}(f) \setminus \{\iota_1, \iota_2\}.\ \\ \quad \square\, \mathsf{BoxSlice}(\mathcal{N}, Q_1 * Q_2, \iota) * \triangleright^b \mathsf{Box}(\mathcal{N}, P, f[\iota_1 \leftarrow \bot][\iota_2 \leftarrow \bot][\iota \leftarrow s])\end{array}}$$

11 Logical Paradoxes

In this section we provide proofs of some logical inconsistencies that arise when slight changes are made to the Iris logic.

11.1 Saved Propositions without a Later

As a preparation for the proof about invariants in §11.2, we show that omitting the later modality from a variant of *saved propositions* leads to a contradiction. Saved propositions have been introduced in prior work [4, 5] to prove correctness of synchronization primitives; we will explain all that is necessary here. The counterexample assumes a higher-order logic with separating conjunction, magic wand and the modalities $\Box$ and $\dot{\Rightarrow}$ satisfying the rules in §5.

Theorem 4. *If there exists a type GName and a proposition* $\_ \Mapsto \_ : \mathit{GName} \to \mathit{iProp} \to \mathit{iProp}$ *associating names* $\gamma : \mathit{GName}$ *to propositions and satisfying:*

$$\vdash \dot{\Rightarrow} \exists \gamma : \mathit{GName}.\, \gamma \Mapsto P(\gamma) \qquad \text{(SPROP-ALLOC)}$$

$$\gamma \Mapsto P \vdash \Box(\gamma \Mapsto P) \qquad \text{(SPROP-PERSIST)}$$

$$\gamma \Mapsto P * \gamma \Mapsto Q \vdash P \Leftrightarrow Q \qquad \text{(SPROP-AGREE)}$$

then $\vdash \dot{\Rightarrow} \mathsf{False}$.

The type *GName* should be thought of as the type of "locations" and $\gamma \Mapsto P$ should be read as stating that location γ "stores" proposition P. Notice that these are immutable locations, so the maps-to proposition is persistent. The rule SPROP-ALLOC is then thought of as allocation, and the rule SPROP-AGREE states that a given location γ can only store *one* proposition, so multiple witnesses covering the same location must agree.

The conclusion of SPROP-AGREE usually is guarded by a $\triangleright$. The point of this theorem is to show that said later is *essential*, as removing it introduces inconsistency. The key to proving Theorem 4 is the following proposition:

Definition 22. $A(\gamma) \triangleq \exists P : \mathit{iProp}.\ \Box \neg P \land \gamma \Mapsto P$.

Intuitively, $A(\gamma)$ says that the saved proposition named γ does *not* hold, *i.e.,* we can disprove it. Using SPROP-PERSIST, it is immediate that $A(\gamma)$ is persistent.

Now, by applying SPROP-ALLOC with A, we obtain a proof of $P \triangleq \gamma \Mapsto A(\gamma)$: this says that the proposition named γ is the proposition saying that it, itself, does not hold. In other words, P says that the proposition named γ expresses its own negation. Unsurprisingly, that leads to a contradiction, as is shown in the following lemma:

Lemma 2. *We have* $\gamma \Mapsto A(\gamma) \vdash \Box \neg A(\gamma)$ *and* $\gamma \Mapsto A(\gamma) \vdash A(\gamma)$.

Proof.

- First we show $\gamma \Mapsto A(\gamma) \vdash \Box \neg A(\gamma)$. Since $\gamma \Mapsto A(\gamma)$ is persistent it suffices to show $\gamma \Mapsto A(\gamma) \vdash \neg A(\gamma)$. Suppose $\gamma \Mapsto A(\gamma)$ and $A(\gamma)$. Then by definition of A there is a P such that $\Box \neg P$ and $\gamma \Mapsto P$. By SPROP-AGREE we have $P \Leftrightarrow A(\gamma)$ and so from $\neg P$ we get $\neg A(\gamma)$, which leads to a contradiction with $A(\gamma)$.

- Using the first item we can now prove $\gamma \Rightarrow\!\Mapsto A(\gamma) \vdash A(\gamma)$. We need to prove

$$\exists P : iProp.\ \Box \neg P \land \gamma \Rightarrow\!\Mapsto P.$$

  We do so by picking P to be $A(\gamma)$, which leaves us to prove $\Box \neg A(\gamma) \land \gamma \Rightarrow\!\Mapsto A(\gamma)$. The last conjunct holds by assumption, and the first conjunct follows from the previous item of this lemma.

□

With this lemma in hand, the proof of Theorem 4 is simple.

Proof of Theorem 4. Using the previous lemmas we have

$$\vdash \forall \gamma.\ \neg(\gamma \Rightarrow\!\Mapsto A(\gamma)).$$

Together with the rule SPROP-ALLOC we thus derive $\dot{\Rightarrow\!\Mapsto}\mathsf{False}$. □

11.2 Invariants without a Later

Now we come to the main paradox: if we remove the $\triangleright$ from INV-OPEN, the logic becomes inconsistent. The theorem is stated as general as possible so that it also applies to previous, less powerful versions of Iris.

Theorem 5. *Assume a higher-order separation logic with $\Box$ and an update modality with a binary mask $\Rightarrow\!\Mapsto_{\{0,1\}}$ (think: empty mask and full mask) satisfying strong monad rules with respect to separating conjunction and such that:*

$$\frac{}{\Rightarrow\!\Mapsto_0 P \vdash \Rightarrow\!\Mapsto_1 P}\ \textit{WEAKEN-MASK}$$

Assume a type InvName and a proposition $\boxed{\cdot}^{\cdot} : InvName \to iProp \to iProp$ satisfying:

$$\frac{}{P \vdash \Rightarrow\!\Mapsto_1 \exists \iota.\ \boxed{P}^{\iota}}\ \textit{INV-ALLOC} \qquad \frac{}{\boxed{P}^{\iota} \vdash \Box \boxed{P}^{\iota}}\ \textit{INV-PERSIST} \qquad \frac{P * Q \vdash \Rightarrow\!\Mapsto_0 (P * R)}{\boxed{P}^{\iota} * Q \vdash \Rightarrow\!\Mapsto_1 R}\ \textit{INV-OPEN-NOLATER}$$

Finally, assume the existence of a type GName and two tokens $\boxed{S}^{\cdot} : GName \to iProp$ and $\boxed{F}^{\cdot} : GName \to iProp$ parameterized by GName and satisfying the following properties:

$$\frac{}{\vdash \Rightarrow\!\Mapsto_0 \exists \gamma.\ \boxed{S}^{\gamma}}\ \textit{START-ALLOC} \qquad \frac{}{\boxed{S}^{\gamma} \vdash \Rightarrow\!\Mapsto_0 \boxed{F}^{\gamma}}\ \textit{START-FINISH} \qquad \frac{}{\boxed{S}^{\gamma} * \boxed{F}^{\gamma} \vdash \mathsf{False}}\ \textit{START-NOT-FINISHED} \qquad \frac{}{\boxed{F}^{\gamma} \vdash \boxed{F}^{\gamma} * \boxed{F}^{\gamma}}\ \textit{FINISHED-DUP}$$

Then $\mathsf{True} \vdash \Rightarrow\!\Mapsto_1 \mathsf{False}$.

The core of the proof is defining the $\Rightarrow\!\Mapsto$ from the previous counterexample using invariants. Then, using the standard proof rules for invariants, we show that it satisfies SPROP-ALLOC and SPROP-PERSIST. Furthermore, assuming the rule for opening invariants without a $\triangleright$, we can prove a slightly weaker version of SPROP-AGREE, which is sufficient for deriving a contradiction.

We start by defining $\Rightarrow\!\Mapsto$ satisfying (almost) the assumptions of Lemma 4.

Definition 23. *We define* $\_ \Rrightarrow \_ : \mathit{GName} \to \mathit{iProp} \to \mathit{iProp}$ *as:*

$$\gamma \Rrightarrow P \triangleq \exists \iota. \boxed{\boxed{S}^{\gamma} \vee \boxed{F}^{\gamma} * \Box P}^{\iota}.$$

Note that using INV-PERSIST, it is immediate that $\gamma \Rrightarrow P$ is persistent.

We use the tokens $\boxed{S}^{\gamma}$ and $\boxed{F}^{\gamma}$ to model invariants that can be initialized "lazily": $\boxed{S}^{\gamma}$ indicates that the invariant is still not initialized, whereas the duplicable $\boxed{F}^{\gamma}$ indicates it has been initialized with a resource satisfying P.

We can show variants of SPROP-AGREE and SPROP-ALLOC for the defined $\Rrightarrow$.

Lemma 3. *We have* $\vdash \Rrightarrow_1 \exists \gamma.\, \gamma \Rrightarrow P(\gamma)$.

Proof. We have to show the allocation rule

$$\vdash \Rrightarrow_1 \exists \gamma.\, \gamma \Rrightarrow P.$$

From START-ALLOC we have a γ such that $\Rrightarrow_0 \boxed{S}^{\gamma}$ holds and hence from WEAKEN-MASK we have $\Rrightarrow_1 \boxed{S}^{\gamma}$. Since we are proving a goal of the form $\Rrightarrow_1 R$ we may assume $\boxed{S}^{\gamma}$. Thus for any P we have $\Rrightarrow_1 \left(\boxed{S}^{\gamma} \vee \boxed{F}^{\gamma} * P\right)$. Again since our goal is still of the form $\Rrightarrow_1$ we may assume $\boxed{S}^{\gamma} \vee \boxed{F}^{\gamma} * \Box P$. The rule INV-ALLOC then gives us precisely what we need. □

Lemma 4. *We have* $\gamma \Rrightarrow P * \gamma \Rrightarrow Q * \Box P \vdash \Rrightarrow_1 \Box Q$ *and thus* $\gamma \Rrightarrow P * \gamma \Rrightarrow Q \vdash (\Rrightarrow_1 \Box P) \Leftrightarrow (\Rrightarrow_1 \Box Q)$.

Proof.

- We first show
$$\gamma \Rrightarrow P * \gamma \Rrightarrow Q * \Box P \vdash \Rrightarrow_1 \Box Q.$$
We use INV-OPEN-NOLATER to open the invariant in $\gamma \Rrightarrow P$ and consider two cases:
  1. $\boxed{S}^{\gamma}$(the invariant is "uninitialized") : In this case, we use START-FINISH to "initialize" the invariant and obtain $\boxed{F}^{\gamma}$. Then we duplicate $\boxed{F}^{\gamma}$, and use it together with $\Box P$ to close the invariant.
  2. $\boxed{F}^{\gamma} * \Box P$ (the invariant is "initialized"): In this case we duplicate $\boxed{F}^{\gamma}$, and use a copy to close the invariant.

  After closing the invariant, we have obtained $\boxed{F}^{\gamma}$. Hence, it is sufficient to prove
  $$\boxed{F}^{\gamma} * \gamma \Rrightarrow P * \gamma \Rrightarrow Q * \Box P \vdash \Rrightarrow_1 \Box Q.$$
  We proceed by using INV-OPEN-NOLATER to open the other invariant in $\gamma \Rrightarrow Q$, and we again consider two cases:
  1. $\boxed{S}^{\gamma}$ (the invariant is "uninitialized"): As witnessed by START-NOT-FINISHED, this cannot happen, so we derive a contradiction. Notice that this is a key point of the proof: because the two invariants ($\gamma \Rrightarrow P$ and $\gamma \Rrightarrow Q$) *share* the ghost name γ, initializing one of them is enough to show that the other one has been initialized. Essentially, this is an indirect way of saying that really, we have been opening the same invariant two times.

2. $\boxed{\mathrm{F}}^{\gamma} * \square Q$ (the invariant is "initialized"): Since $\square Q$ is duplicable we use one copy to close the invariant, and retain another to prove $\Rrightarrow_1 \square Q$.

- By applying the above twice, we easily obtain

$$\gamma \Mapsto P * \gamma \Mapsto Q \vdash (\Rrightarrow_1 \square P) \Leftrightarrow (\Rrightarrow_1 \square Q)$$

□

Intuitively, Lemma 4 shows that we can "convert" a proof from P to Q.

We are now in a position to replay the counterexample from §11.1. The only difference is that because Lemma 4 is slightly weaker than the rule SPROP-AGREE of Theorem 4, we need to use $\Rrightarrow_1 \mathsf{False}$ in place of False in the definition of the predicate A: we let $A(\gamma) \triangleq \exists P : \mathit{iProp}.\ \square(P \Rightarrow \Rrightarrow_1 \mathsf{False}) \wedge \gamma \Mapsto P$ and replay the proof that we have presented above.

12 HeapLang

So far, we have treated the programming language we work on entirely generically. In this section we present the default language that ships with Iris, HeapLang. HeapLang is an ML-like languages with a higher-order heap, unstructured concurrency, some common atomic operations, and prophecy variables. It is an instance of the language interface (§8), so we only define a per-thread small-step operational semantics—the thread-pool semantics are given in in §8.1.

12.1 HeapLang syntax and operational semantics

The grammar of HeapLang, and in particular its set *Expr* of *expressions* and *Val* of *values*, is defined as follows:

$$
\begin{aligned}
v, w \in \mathit{Val} ::= {} & z \mid \texttt{true} \mid \texttt{false} \mid () \mid \textcircled{\scriptsize ☠} \mid \ell \mid p \mid && (z \in \mathbb{Z}, \ell \in \mathit{Loc}, p \in \mathit{ProphId}) \\
& \texttt{rec}_{\texttt{v}}\ f(x) = e \mid (v, w)_{\texttt{v}} \mid \texttt{inl}_{\texttt{v}}(v) \mid \texttt{inr}_{\texttt{v}}(v) \\
e \in \mathit{Expr} ::= {} & v \mid x \mid \texttt{rec}_{\texttt{e}}\ f(x) = e \mid e_1(e_2) \mid \\
& \circledcirc_1\, e \mid e_1 \circledcirc_2 e_2 \mid \texttt{if}\, e\, \texttt{then}\, e_1\, \texttt{else}\, e_2 \mid \\
& (e_1, e_2)_{\texttt{e}} \mid \texttt{fst}(e) \mid \texttt{snd}(e) \mid \\
& \texttt{inl}_{\texttt{e}}(e) \mid \texttt{inr}_{\texttt{e}}(e) \mid \texttt{match}\, e\, \texttt{with}\, \texttt{inl} \Rightarrow e_1 \mid \texttt{inr} \Rightarrow e_2\, \texttt{end} \mid \\
& \texttt{AllocN}(e_1, e_2) \mid \texttt{Free}(e) \mid !\, e \mid e_1 \leftarrow e_2 \mid \texttt{CmpXchg}(e_1, e_2, e_3) \mid \texttt{Xchg}(e_1, e_2) \mid \texttt{FAA}(e_1, e_2) \mid \\
& \texttt{fork}\ \{e\} \mid \texttt{newproph} \mid \texttt{resolve}\, \texttt{with}\, e_1\, \texttt{at}\, e_2\, \texttt{to}\, e_3 \\
\circledcirc_1 ::= {} & - \mid \ldots \quad \text{(list incomplete)} \\
\circledcirc_2 ::= {} & + \mid - \mid +_{\texttt{L}} \mid = \mid \ldots \quad \text{(list incomplete)}
\end{aligned}
$$

(Note that **match** contains a literal | that is not part of the BNF but part of HeapLang syntax.)

To simplify the formalization, the only binders occur in **rec**. **match** has a closure in each arm which will be applied to the value of the left/right variant, respectively. (See the syntactic sugar defined later.)

Recursive abstractions, pairs, and the injections exist both as a value form and an expression form. The expression forms will reduce to the value form once all their arguments are values. Conceptually, one can think of that as corresponding to "boxing" that most functional language implementations do. We will leave away the disambiguating subscript when it is clear from the context or does not matter. All of this lets us define val_to_expr as simply applying the value injection (the very first syntactic form of *Expr*), which makes a lot of things in Coq much simpler. expr_to_val is defined recursively in the obvious way.

AllocN takes as first argument the number of heap cells to allocate (must be strictly positive), and as second argument the default value to use for these heap cells. This lets one allocate arrays. $+_{\texttt{L}}$ implements pointer arithmetic (the left operand must be a pointer, the right operand an integer), which is used to access array elements.

For our set of states and observations, we pick

$$\begin{aligned} \ell \ni \mathit{Loc} &\triangleq \mathbb{Z} \\ p \ni \mathit{ProphId} &\triangleq \mathbb{Z} \\ \sigma \ni \mathit{State} &\triangleq \left\{ \begin{array}{r} \text{HEAP} : \mathit{Loc} \xrightarrow{\text{fin}} \mathit{Val}, \\ \text{PROPHS} : \wp(\mathit{ProphId}) \end{array} \right\} \\ \kappa \ni \mathit{Obs} &\triangleq \mathit{ProphId} \times (\mathit{Val} \times \mathit{Val}) \end{aligned}$$

The HeapLang operational semantics is defined via the use of *evaluation contexts*:

$$\begin{aligned} K \in \mathit{Ctx} ::= {} & \bullet \mid \mathit{Ctx}_> \\ K_> \in \mathit{Ctx}_> ::= {} & e(K) \mid K(v) \mid \\ & \circledcirc_1\, K \mid e \circledcirc_2 K \mid K \circledcirc_2 v \mid \texttt{if}\, K\, \texttt{then}\, e_1\, \texttt{else}\, e_2 \mid \\ & (e, K) \mid (K, v) \mid \texttt{fst}(K) \mid \texttt{snd}(K) \mid \\ & \texttt{inl}(K) \mid \texttt{inr}(K) \mid \texttt{match}\, K\, \texttt{with}\, \texttt{inl} \Rightarrow e_1 \mid \texttt{inr} \Rightarrow e_2\, \texttt{end} \mid \\ & \texttt{AllocN}(e, K) \mid \texttt{AllocN}(K, v) \mid \texttt{Free}(K) \mid !\, K \mid e \leftarrow K \mid K \leftarrow v \mid \\ & \texttt{CmpXchg}(e_1, e_2, K) \mid \texttt{CmpXchg}(e_1, K, v_3) \mid \texttt{CmpXchg}(K, v_2, v_3) \mid \\ & \texttt{Xchg}(e, K) \mid \texttt{Xchg}(K, v) \mid \texttt{FAA}(e, K) \mid \texttt{FAA}(K, v) \mid \\ & \texttt{resolve}\, \texttt{with}\, e_1\, \texttt{at}\, e_2\, \texttt{to}\, K \mid \texttt{resolve}\, \texttt{with}\, e_1\, \texttt{at}\, K\, \texttt{to}\, v_3 \mid \\ & \texttt{resolve}\, \texttt{with}\, K_>\, \texttt{at}\, v_2\, \texttt{to}\, v_3 \end{aligned}$$

Note that we use right-to-left evaluation order. This means in a curried function call $f(x)(y)$, we know syntactically the arguments will all evaluate before f gets to do anything, which makes specifying curried calls a lot easier.

The **resolve** evaluation context for the leftmost expression (the nested expression that executes atomically together with the prophecy resolution) is special: it must not be empty; only further nested evaluation contexts are allowed. **resolve** takes care of reducing the expression once the nested contexts are taken care of, and at that point it requires the expression to reduce to a value in exactly one step. Hence we define $\mathit{Ctx}_>$ for non-empty evaluation contexts. For more details on prophecy variables, see [7].

This lets us define the primitive reduction relation in terms of a "head step" reduction; see Figure 1 and Figure 2. Comparison (both for **CmpXchg** and the binary comparison operator) is a bit tricky and uses a helper judgment (Figure 3). Basically, two values can only be compared if at least one of them is "compare-safe". Compare-safe values are basic literals (integers, Booleans, locations, unit) as well as **inl** and **inr** of those literals. The intention of this is to forbid directly comparing large values such as pairs, which could not be done in a single atomic step on a real machine.

Per-thread reduction $\boxed{e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathtt{t}} e_2, \sigma_2, \vec{e}}$

$$\frac{e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathtt{h}} e_2, \sigma_2, \vec{e}}{K[\,e_1\,], \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathtt{t}} K[\,e_2\,], \sigma_2, \vec{e}}$$

"Head" reduction (pure) $\boxed{e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathtt{h}} e_2, \sigma_2, \vec{e}}$

"Boxing" reductions

$$(\mathtt{rec}_{\mathtt{e}}\ f(x) = e, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (\mathtt{rec}_{\mathtt{v}}\ f(x) = e, \sigma, \epsilon)$$
$$((v_1, v_2)_{\mathtt{e}}, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} ((v_1, v_2)_{\mathtt{v}}, \sigma, \epsilon)$$
$$(\mathtt{inl}_{\mathtt{e}}(v), \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (\mathtt{inl}_{\mathtt{v}}(v), \sigma, \epsilon)$$
$$(\mathtt{inr}_{\mathtt{e}}(v), \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (\mathtt{inr}_{\mathtt{v}}(v), \sigma, \epsilon)$$

Pure reductions

$$((\mathtt{rec}_{\mathtt{v}}\ f(x) = e)(v), \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (e[(\mathtt{rec}\ f(x) = e)/f][v/x], \sigma, \epsilon)$$
$$(-_{\circledcirc} z, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (-z, \sigma, \epsilon)$$
$$(z_1 +_{\circledcirc} z_2, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (z_1 + z_2, \sigma, \epsilon)$$
$$(z_1 -_{\circledcirc} z_2, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (z_1 - z_2, \sigma, \epsilon)$$
$$(\ell +_{\mathtt{L}} z, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (\ell + z, \sigma, \epsilon)$$
$$(v_1 =_{\circledcirc} v_2, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (\mathtt{true}, \sigma, \epsilon) \qquad \text{if } v_1 \cong v_2$$
$$(v_1 =_{\circledcirc} v_2, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (\mathtt{false}, \sigma, \epsilon) \qquad \text{if } v_1 \ncong v_2$$
$$(\mathtt{if}\ \mathtt{true}\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (e_1, \sigma, \epsilon)$$
$$(\mathtt{if}\ \mathtt{false}\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (e_2, \sigma, \epsilon)$$
$$(\mathtt{fst}((v_1, v_2)_{\mathtt{v}}), \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (v_1, \sigma, \epsilon)$$
$$(\mathtt{snd}((v_1, v_2)_{\mathtt{v}}), \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (v_2, \sigma, \epsilon)$$
$$(\mathtt{match}\ \mathtt{inl}_{\mathtt{v}}(v)\ \mathtt{with}\ \mathtt{inl} \Rightarrow e_1 \mid \mathtt{inr} \Rightarrow e_2\ \mathtt{end}, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (e_1(v), \sigma, \epsilon)$$
$$(\mathtt{match}\ \mathtt{inr}_{\mathtt{v}}(v)\ \mathtt{with}\ \mathtt{inl} \Rightarrow e_1 \mid \mathtt{inr} \Rightarrow e_2\ \mathtt{end}, \sigma) \xrightarrow{\epsilon}_{\mathtt{h}} (e_2(v), \sigma, \epsilon)$$

Figure 1: HeapLang pure and boxed reduction rules.
The $\circledcirc$ subscript indicates that this is the HeapLang operator, not the mathematical operator.

"Head" reduction (impure) $\boxed{e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{h}} e_2, \sigma_2, \vec{e}}$

Heap reductions

$$
\begin{aligned}
(\texttt{AllocN}(z, v), \sigma) &\xrightarrow{\epsilon}_{\mathsf{h}} (\ell, \sigma : \text{HEAP}[[\ell, \ell + z) \leftarrow v], \epsilon) && \text{if } z > 0 \text{ and } \forall i < z.\, \sigma.\text{HEAP}(\ell + i) = \bot \\
(\texttt{Free}(\ell), \sigma) &\xrightarrow{\epsilon}_{\mathsf{h}} ((), \sigma : \text{HEAP}[\ell \leftarrow \bot], \epsilon) && \text{if } \sigma.\text{HEAP}(l) = v \\
(!\,\ell, \sigma) &\xrightarrow{\epsilon}_{\mathsf{h}} (v, \sigma, \epsilon) && \text{if } \sigma.\text{HEAP}(l) = v \\
(\ell \leftarrow w, \sigma) &\xrightarrow{\epsilon}_{\mathsf{h}} ((), \sigma : \text{HEAP}[\ell \leftarrow w], \epsilon) && \text{if } \sigma.\text{HEAP}(l) = v \\
(\texttt{CmpXchg}(\ell, w_1, w_2), \sigma) &\xrightarrow{\epsilon}_{\mathsf{h}} ((v, \texttt{true}), \sigma : \text{HEAP}[\ell \leftarrow w_2], \epsilon) && \text{if } \sigma.\text{HEAP}(l) = v \text{ and } v \cong w_1 \\
(\texttt{CmpXchg}(\ell, w_1, w_2), \sigma) &\xrightarrow{\epsilon}_{\mathsf{h}} ((v, \texttt{false}), \sigma, \epsilon) && \text{if } \sigma.\text{HEAP}(l) = v \text{ and } v \ncong w_1 \\
(\texttt{Xchg}(\ell, w), \sigma) &\xrightarrow{\epsilon}_{\mathsf{h}} (v, \sigma : \text{HEAP}[\ell \leftarrow w], \epsilon) && \text{if } \sigma.\text{HEAP}(l) = v \\
(\texttt{FAA}(\ell, z_2), \sigma) &\xrightarrow{\epsilon}_{\mathsf{h}} (z_1, \sigma : \text{HEAP}[\ell \leftarrow z_1 + z_2], \epsilon) && \text{if } \sigma.\text{HEAP}(l) = z_1
\end{aligned}
$$

Special reductions

$$
\begin{aligned}
(\texttt{fork}\ \{e\}, \sigma) &\xrightarrow{\epsilon}_{\mathsf{h}} ((), \sigma, e) \\
(\texttt{newproph}, \sigma) &\xrightarrow{\epsilon}_{\mathsf{h}} (p, \sigma : \text{PROPHS} \uplus \{p\}, \epsilon) && \text{if } p \notin \sigma.\text{PROPHS}
\end{aligned}
$$

$$
\frac{(e, \sigma) \xrightarrow{\vec{\kappa}}_{\mathsf{h}} (v, \sigma', \vec{e'})}{(\texttt{resolve}\ \texttt{with}\ e\ \texttt{at}\ p\ \texttt{to}\ w, \sigma) \xrightarrow{\vec{\kappa} +\!\!+ [(p,(v,w))]}_{\mathsf{h}} (v, \sigma', \vec{e'})}
$$

Figure 2: HeapLang impure reduction rules.
Here, $\sigma : \text{HEAP}$ denotes σ with the HEAP field updated as indicated. $[\ell, \ell + z)$ in the `AllocN` rule indicates that we update all locations in this (left-closed, right-open) interval.

Value (in)equality $\boxed{v \cong w, v \ncong w}$

$$
\begin{aligned}
v \cong w &\triangleq v = w \wedge (\mathsf{val\_compare\_safe}(v) \vee \mathsf{val\_compare\_safe}(w)) \\
v \ncong w &\triangleq v \neq w \wedge (\mathsf{val\_compare\_safe}(v) \vee \mathsf{val\_compare\_safe}(w))
\end{aligned}
$$

$$
\mathsf{lit\_compare\_safe}(z) \qquad \mathsf{lit\_compare\_safe}(\texttt{true}) \qquad \mathsf{lit\_compare\_safe}(\texttt{false})
$$

$$
\mathsf{lit\_compare\_safe}(\ell) \qquad \mathsf{lit\_compare\_safe}(())
$$

$$
\frac{\mathsf{lit\_compare\_safe}(v)}{\mathsf{val\_compare\_safe}(v)} \qquad \frac{\mathsf{lit\_compare\_safe}(v)}{\mathsf{val\_compare\_safe}(\texttt{inl}(v))} \qquad \frac{\mathsf{lit\_compare\_safe}(v)}{\mathsf{val\_compare\_safe}(\texttt{inr}(v))}
$$

Figure 3: HeapLang value comparison judgment.

12.2 Syntactic sugar

We recover many of the common language operations as syntactic sugar.

$$
\begin{aligned}
\lambda x.\, e &\triangleq \texttt{rec}\ \_(x) = e \\
\texttt{let}\ x = e\ \texttt{in}\ e' &\triangleq (\lambda x.\, e')(e) \\
e; e' &\triangleq \texttt{let}\ \_ = e\ \texttt{in}\ e' \\
\texttt{skip} &\triangleq (); () \\
e_1 \,\&\&\, e_2 &\triangleq \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ \texttt{false} \\
e_1 \,||\, e_2 &\triangleq \texttt{if}\ e_1\ \texttt{then}\ \texttt{true}\ \texttt{else}\ e_2 \\
\texttt{match}\ e\ \texttt{with}\ \texttt{inl}(x) \Rightarrow e_1 \mid \texttt{inr}(y) \Rightarrow e_2\ \texttt{end} &\triangleq \texttt{match}\ e\ \texttt{with}\ \texttt{inl} \Rightarrow \lambda x.\, e_1 \mid \texttt{inr} \Rightarrow \lambda y.\, e_2\ \texttt{end} \\
\texttt{ref}(e) &\triangleq \texttt{AllocN}(1, e) \\
\texttt{CAS}(e_1, e_2, e_3) &\triangleq \texttt{snd}(\texttt{CmpXchg}(e_1, e_2, e_3)) \\
\texttt{resolve}\ e_1\ \texttt{to}\ e_2 &\triangleq \texttt{resolve}\ \texttt{with}\ \texttt{skip}\ \texttt{at}\ e_1\ \texttt{to}\ e_2
\end{aligned}
$$

References

[1] Pierre America and Jan Rutten. "Solving Reflexive Domain Equations in a Category of Complete Metric Spaces". In: *JCSS* 39.3 (1989), pp. 343–375.

[2] Lars Birkedal and Aleš Bizjak. *A Taste of Categorical Logic — Tutorial Notes*. Available at http://users-cs.au.dk/birke/modures/tutorial/categorical-logic-tutorial-notes.pdf. Oct. 2014.

[3] Lars Birkedal, Kristian Støvring, and Jacob Thamsborg. "The category-theoretic solution of recursive metric-space equations". In: *TCS* 411.47 (2010), pp. 4102–4122. DOI: 10.1016/j.tcs.2010.07.010. URL: http://dx.doi.org/10.1016/j.tcs.2010.07.010.

[4] Mike Dodds et al. "Verifying Custom Synchronization Constructs Using Higher-Order Separation Logic". In: *TOPLAS* 38.2 (2016), p. 4. DOI: 10.1145/2818638. URL: http://doi.acm.org/10.1145/2818638.

[5] Ralf Jung et al. "Higher-order ghost state". In: *ICFP*. 2016, pp. 256–269.

[6] Ralf Jung et al. "Iris from the Ground Up". In: *Submitted to JFP* (2017).

[7] Ralf Jung et al. "The future is ours: prophecy variables in separation logic". In: *PACMPL* 4.POPL (2020), 45:1–45:32. DOI: 10.1145/3371113.

[8] Martin H. Löb. "Solution of a Problem of Leon Henkin". In: *The Journal of Symbolic Logic* 20.2 (1955), pp. 115–118. ISSN: 00224812. URL: http://www.jstor.org/stable/2266895.

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