Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning (Technical Appendix)

Ralf Jung
MPI-SWS & Saarland University
jung@mpi-sws.org

David Swasey
MPI-SWS
swasey@mpi-sws.org

Filip Sieczkowski
Aarhus University
filips@cs.au.dk

Kasper Svendsen
Aarhus University
ksvendsen@cs.au.dk

Aaron Turon
Mozilla Research
aturon@mozilla.com

Lars Birkedal
Aarhus University
birkedal@cs.au.dk

Derek Dreyer
MPI-SWS
dreyer@mpi-sws.org

February 2, 2015

Contents

Part I
Iris: The framework

1 Parameters to the framework

- A set *Exp* of *expressions* (metavariable e) with a subset *Val* of values (v). We assume that if e is an expression then so is $\mathbf{fork}\ e$. We moreover assume a value fRet (giving the intended return value of a fork), and we assume that

$$\mathbf{fork}\ e \notin \mathit{Val}$$
$$\mathbf{fork}\ e_1 = \mathbf{fork}\ e_2 \implies e_1 = e_2$$

- A set *Ectx* of *evaluation contexts* (K) that includes the empty context $[\,]$, a plugging operation $K[e]$ that produces an expression, and context composition $\circ$ satisfying the following axioms:

$$[\,][e] = e$$
$$K_1[K_2[e]] = (K_1 \circ K_2)[e]$$
$$K_1[e] = K_2[e] \implies K_1 = K_2$$
$$K[e_1] = K[e_2] \implies e_1 = e_2$$
$$K_1 \circ K_2 = [\,] \implies K_1 = K_2 = [\,]$$
$$K[e] \in \mathit{Val} \implies K = [\,]$$
$$K[e] = \mathbf{fork}\ e' \implies K = [\,]$$

- A set *State* of shared machine states (*e.g.,* heaps), metavariable ς.
- An *atomic stepping relation*

$$(- \to -) \subseteq (\mathit{State} \times \mathit{Exp}) \times (\mathit{State} \times \mathit{Exp})$$

  and notions of an expression to be *reducible* or *stuck*, such that

$$\mathsf{reducible}(e) \iff \exists \varsigma, e_2, \varsigma_2.\ \varsigma; e \to \varsigma_2; e_2$$
$$\mathsf{stuck}(e) \iff \forall K, e'.\ e = K[e'] \implies \neg\mathsf{reducible}(e')$$

  and the following hold

$$\mathsf{stuck}(\mathbf{fork}\ e)$$
$$\mathsf{stuck}(v)$$
$$K[e] = K'[e'] \implies \mathsf{reducible}(e') \implies e \notin \mathit{Val} \implies \exists K''.\ K' = K \circ K'' \quad \text{(step-by-value)}$$
$$K[e] = K'[\mathbf{fork}\ e'] \implies e \notin \mathit{Val} \implies \exists K''.\ K' = K \circ K'' \quad \text{(fork-by-value)}$$

- A predicate atomic on expressions satisfying

$$\mathsf{atomic}(e) \implies \mathsf{reducible}(e)$$
$$\mathsf{atomic}(e) \implies \varsigma; e \to \varsigma_2; e_2 \implies e_2 \in \mathit{Val} \quad \text{(atomic-step)}$$

- A commutative monoid with zero, M. That is, a set $|M|$ with two distinguished elements $\bot$ (zero, undefined) and ϵ (one, unit) and an operation $\cdot$ (times, combine) such that

$$a \cdot b = b \cdot a$$
$$\epsilon \cdot a = a$$
$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$
$$\bot \cdot a = \bot$$
$$\bot \neq \epsilon$$

Let $|M|^+ \triangleq |M| \setminus \{\bot\}$.

- Arbitrary additional types and terms.

2 The derived language

Machine syntax

$$T \in \mathit{ThreadPool} \triangleq \mathbb{N} \xrightarrow{\text{fin}} \mathit{Exp}$$

Machine reduction $\boxed{\varsigma; T \to \varsigma'; T'}$

$$\frac{\varsigma; e \to \varsigma'; e'}{\varsigma; T[i \mapsto K[e]] \to \varsigma'; T[i \mapsto K[e']]} \qquad \varsigma; T[i \mapsto K[\mathbf{fork}\ e]] \to \varsigma; T[i \mapsto K[\mathsf{fRet}]][j \mapsto e]$$

3 Syntax

3.1 Grammar

Iris syntax is built up from a countably infinite set *Var* of variables (ranged over by metavariables x, y, z, and p):

$$\begin{aligned}
t, P, \varphi ::= {} & x \mid () \mid (t, t) \mid \pi_i\ t \mid \lambda x.\ t \mid t\ t \mid \bot \mid \epsilon \mid t \cdot t \mid \cdots \mid \\
& \mathsf{False} \mid \mathsf{True} \mid t =_\Sigma t \mid P \Rightarrow P \mid P \wedge P \mid P \vee P \mid P * P \mid P \mathrel{-\!\!*} P \mid \\
& \mu p.\ \varphi \mid \exists x : \Sigma.\ P \mid \forall x : \Sigma.\ P \mid \\
& \boxed{P}^t \mid \lceil t \rceil \mid \lfloor t \rfloor \mid \Box P \mid \triangleright P \mid \mathsf{vs}_t^t(P) \mid \mathsf{wp}_t(t, \varphi) \\
\Sigma ::= {} & \mathsf{Val} \mid \mathsf{Exp} \mid \mathsf{Ectx} \mid \mathsf{State} \mid \mathsf{Monoid} \mid \mathsf{Name} \mid \mathsf{InvMask} \mid \mathsf{Prop} \mid \\
& 1 \mid \Sigma \times \Sigma \mid \Sigma \to \Sigma
\end{aligned}$$

Recursive predicates must be *guarded*: in $\mu p.\ \varphi$, the variable p can only appear under the later $\triangleright$ modality.

Metavariable conventions. We introduce additional metavariables ranging over terms and generally let the choice of metavariable indicate the term's sort:

metavariable	sort
t, u	arbitrary
v, w	Val
e	Exp
K	Ectx
ς	State

metavariable	sort
ι	Name
$\mathcal{E}$	$\mathsf{InvMask}$
a, b	Monoid
P, Q, R	Prop
φ, ψ, ζ	$\Sigma \to \mathsf{Prop}$ (when Σ is clear from context)

Variable conventions. We often abuse notation, using the preceding *term* metavariables to range over (bound) *variables*. We omit type annotations in binders, when the type is clear from context.

3.2 Types

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

A variable context, $\Gamma = x_1 : \Sigma_1, \ldots, x_n : \Sigma_n$, declares a list of variables and their sorts. In writing $\Gamma, x : \Sigma$, we presuppose that x is not already declared in Γ.

Well-typed terms $\boxed{\Gamma \vdash t : \Sigma}$

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

$$\Gamma \vdash () : 1 \qquad \frac{\Gamma \vdash t : \Sigma_1 \qquad \Gamma \vdash u : \Sigma_2}{\Gamma \vdash (t, u) : \Sigma_1 \times \Sigma_2} \qquad \frac{\Gamma \vdash t : \Sigma_1 \times \Sigma_2 \qquad i \in \{1, 2\}}{\Gamma \vdash \pi_i\, t : \Sigma_i} \qquad \frac{\Gamma, x : \Sigma \vdash t : \Sigma'}{\Gamma \vdash \lambda x.\ t : \Sigma \to \Sigma'}$$

$$\frac{\Gamma \vdash t : \Sigma \to \Sigma' \qquad u : \Sigma}{\Gamma \vdash t\ u : \Sigma'} \qquad \Gamma \vdash \bot : \mathsf{Monoid} \qquad \Gamma \vdash \epsilon : \mathsf{Monoid} \qquad \frac{\Gamma \vdash a : \mathsf{Monoid} \qquad \Gamma \vdash b : \mathsf{Monoid}}{\Gamma \vdash a \cdot b : \mathsf{Monoid}}$$

$$\Gamma \vdash \mathsf{False} : \mathsf{Prop} \qquad \Gamma \vdash \mathsf{True} : \mathsf{Prop} \qquad \frac{\Gamma \vdash t : \Sigma \qquad \Gamma \vdash u : \Sigma}{\Gamma \vdash t =_\Sigma u : \mathsf{Prop}} \qquad \frac{\Gamma \vdash P : \mathsf{Prop} \qquad \Gamma \vdash Q : \mathsf{Prop}}{\Gamma \vdash P \Rightarrow Q : \mathsf{Prop}}$$

$$\frac{\Gamma \vdash P : \mathsf{Prop} \qquad \Gamma \vdash Q : \mathsf{Prop}}{\Gamma \vdash P \wedge Q : \mathsf{Prop}} \qquad \frac{\Gamma \vdash P : \mathsf{Prop} \qquad \Gamma \vdash Q : \mathsf{Prop}}{\Gamma \vdash P \vee Q : \mathsf{Prop}} \qquad \frac{\Gamma \vdash P : \mathsf{Prop} \qquad \Gamma \vdash Q : \mathsf{Prop}}{\Gamma \vdash P * Q : \mathsf{Prop}}$$

$$\frac{\Gamma \vdash P : \mathsf{Prop} \qquad \Gamma \vdash Q : \mathsf{Prop}}{\Gamma \vdash P \mathrel{-\!\!*} Q : \mathsf{Prop}} \qquad \frac{\Gamma, p : \Sigma \to \mathsf{Prop} \vdash \varphi : \Sigma \to \mathsf{Prop} \qquad p \text{ is guarded in } \varphi}{\Gamma \vdash \mu p.\ \varphi : \Sigma \to \mathsf{Prop}}$$

$$\frac{\Gamma, x : \Sigma \vdash P : \mathsf{Prop}}{\Gamma \vdash \exists x : \Sigma.\ P : \mathsf{Prop}} \qquad \frac{\Gamma, x : \Sigma \vdash P : \mathsf{Prop}}{\Gamma \vdash \forall x : \Sigma.\ P : \mathsf{Prop}} \qquad \frac{\Gamma \vdash P : \mathsf{Prop} \qquad \Gamma \vdash \iota : \mathsf{Name}}{\Gamma \vdash \boxed{P}^{\iota} : \mathsf{Prop}} \qquad \frac{\Gamma \vdash a : \mathsf{Monoid}}{\Gamma \vdash \boxed{a} : \mathsf{Prop}}$$

$$\frac{\Gamma \vdash \varsigma : \mathsf{State}}{\Gamma \vdash \lfloor \varsigma \rfloor : \mathsf{Prop}} \qquad \frac{\Gamma \vdash P : \mathsf{Prop}}{\Gamma \vdash \Box P : \mathsf{Prop}} \qquad \frac{\Gamma \vdash P : \mathsf{Prop}}{\Gamma \vdash \triangleright P : \mathsf{Prop}}$$

$$\frac{\Gamma \vdash P : \mathsf{Prop} \qquad \Gamma \vdash \mathcal{E} : \mathsf{InvMask} \qquad \Gamma \vdash \mathcal{E}' : \mathsf{InvMask}}{\Gamma \vdash \mathsf{vs}_{\mathcal{E}}^{\mathcal{E}'}(P) : \mathsf{Prop}}$$

$$\frac{\Gamma \vdash e : \mathsf{Exp} \qquad \Gamma \vdash \varphi : \mathsf{Val} \to \mathsf{Prop} \qquad \Gamma \vdash \mathcal{E} : \mathsf{InvMask}}{\Gamma \vdash \mathsf{wp}_{\mathcal{E}}(e, \varphi) : \mathsf{Prop}}$$

4 Semantics

An *ordered family of equivalence relations* (o.f.e.) is a pair $(X, (\overset{n}{=})_{n \in \mathbb{N}})$, with X a non-empty set, and each $\overset{n}{=}$ an equivalence relation over X satisfying

- $\forall x, x'.\ x \overset{0}{=} x'$,
- $\forall x, x', n.\ x \overset{n+1}{=} x' \implies x \overset{n}{=} x'$,
- $\forall x, x'.\ (\forall n.\ x \overset{n}{=} x') \implies x = x'$.

Let $(X, (\overset{n}{=}_X)_{n \in \mathbb{N}})$ and $(Y, (\overset{n}{=}_Y)_{n \in \mathbb{N}})$ be o.f.e.'s. A function $f : X \to Y$ is *non-expansive* if, for all x, x' and n,

$$x \overset{n}{=}_X x' \implies fx \overset{n}{=}_Y fx'.$$

Let $(X, (\overset{n}{=})_{n \in \mathbb{N}})$ be an o.f.e. A sequence $(x_i)_{i \in \mathbb{N}}$ of elements in X is a *chain* (aka *Cauchy sequence*) if

$$\forall k.\ \exists n.\ \forall i, j \geq n.\ x_i \overset{k}{=} x_j.$$

A *limit* of a chain $(x_i)_{i\in\mathbb{N}}$ is an element $x \in X$ such that

$$\forall n.\ \exists k.\ \forall i \geq k.\ x_i \overset{n}{=} x.$$

An o.f.e. $(X, (\overset{n}{=})_{n\in\mathbb{N}})$ is *complete* if all chains have a limit. A complete o.f.e. is called a c.o.f.e. (pronounced "coffee"). When the family of equivalence relations is clear from context we simply write X for a c.o.f.e. $(X, (\overset{n}{=})_{n\in\mathbb{N}})$.

Let $\mathcal{U}$ be the category of c.o.f.e.'s and nonexpansive maps.

Products and function spaces are defined as follows. For c.o.f.e.'s $(X, (\overset{n}{=}_X)_{n\in\mathbb{N}})$ and $(Y, (\overset{n}{=}_Y)_{n\in\mathbb{N}})$, their product is $(X \times Y, (\overset{n}{=})_{n\in\mathbb{N}})$, where

$$(x, y) \overset{n}{=} (x', y') \iff x \overset{n}{=} x' \wedge y \overset{n}{=} y'.$$

The function space is

$$(\{\, f : X \to Y \mid f \text{ is non-expansive} \,\}, (\overset{n}{=})_{n\in\mathbb{N}}),$$

where

$$f \overset{n}{=} g \iff \forall x.\ f(x) \overset{n}{=} g(x).$$

For a c.o.f.e. $(X, (\overset{n}{=}_{n\in\mathbb{N}}))$, $\blacktriangleright(X, (\overset{n}{=}_{n\in\mathbb{N}}))$ is the c.o.f.e. $(X, (\overset{n}{\equiv}_{n\in\mathbb{N}}))$, where

$$x \overset{n}{\equiv} x' \iff \begin{cases} \top & \text{if } n = 0 \\ x \overset{n-1}{=} x' & \text{if } n > 0 \end{cases}$$

(Sidenote: $\blacktriangleright$ extends to a functor on $\mathcal{U}$ by the identity action on morphisms).

4.1 Semantic structures: propositions

$$\begin{aligned}
\mathit{Res} &\triangleq \{\, r = (\pi, g) \mid \pi \in \mathit{State} \uplus \{\epsilon\} \wedge g \in |M|^+ \,\} \\
(\pi, g) \bullet (\pi', g') &\triangleq \begin{cases} (\pi, g \cdot g') & \text{if } \pi' = \epsilon \text{ and } g \cdot g' \neq \bot \\ (\pi', g \cdot g') & \text{if } \pi = \epsilon \text{ and } g \cdot g' \neq \bot \end{cases} \\
r \leq r' &\triangleq \exists r''.\ r' = r \bullet r'' \\
\\
\mathit{UPred}(\mathit{Res}) &\triangleq \{\, p \subseteq \mathbb{N} \times \mathit{Res} \mid \forall (k, r) \in p.\ \forall j \leq k.\ \forall r' \geq r.\ (j, r') \in p \,\} \\
\lfloor p \rfloor_k &\triangleq \{\, (j, r) \in p \mid j < k \,\} \\
p \overset{n}{=} q &\triangleq \lfloor p \rfloor_n = \lfloor q \rfloor_n \\
\\
\mathit{PreProp} &\cong \blacktriangleright\big(\mathit{World} \overset{\text{mon}}{\to} \mathit{UPred}(\mathit{Res})\big) \\
\mathit{World} &\triangleq \mathbb{N} \overset{\text{fin}}{\rightharpoonup} \mathit{PreProp} \\
w \overset{n}{=} w' &\triangleq n = 0 \vee \big(\mathrm{dom}(w) = \mathrm{dom}(w') \wedge \forall i \in \mathrm{dom}(w).\ w(i) \overset{n}{=} w'(i)\big) \\
w \leq w' &\triangleq \mathrm{dom}(w) \subseteq \mathrm{dom}(w') \wedge \forall i \in \mathrm{dom}(w).\ w(i) = w'(i) \\
\mathit{Prop} &\triangleq \mathit{World} \overset{\text{mon}}{\to} \mathit{UPred}(\mathit{Res})
\end{aligned}$$

For $p, q \in \mathit{UPred}(\mathit{Res})$ with $p \overset{n}{=} q$ defined as above, $\mathit{UPred}(\mathit{Res})$ is a c.o.f.e.

Prop is a c.o.f.e., which exists by America and Rutten's theorem [1]. We do not need to consider how the object is constructed. We only need the isomorphism, given by maps

$$\xi : \blacktriangleright\big(\mathit{World} \overset{\text{mon}}{\to} \mathit{UPred}(\mathit{Res})\big) \to \mathit{PreProp}$$
$$\xi^{-1} : \mathit{PreProp} \to \blacktriangleright\big(\mathit{World} \overset{\text{mon}}{\to} \mathit{UPred}(\mathit{Res})\big)$$

which are inverses to each other. Note: this is an isomorphism in $\mathcal{U}$, i.e., ξ and ξ^{-1} are both non-expansive.

World is a c.o.f.e. with the family of equivalence relations defined as shown above.

4.2 Semantic structures: types and environments

For a set X, write ΔX for the discrete c.o.f.e. with $x \stackrel{n}{=} x'$ iff $n = 0$ or $x = x'$

$$\begin{array}{lll} [\![\mathbf{1}]\!] \triangleq \Delta\{\star\} & [\![\mathsf{Val}]\!] \triangleq \Delta\, Val & [\![\Sigma \times \Sigma']\!] \triangleq [\![\Sigma]\!] \times [\![\Sigma]\!] \\ [\![\mathsf{Name}]\!] \triangleq \Delta\mathbb{N} & [\![\mathsf{Exp}]\!] \triangleq \Delta\, Exp & [\![\Sigma \to \Sigma']\!] \triangleq [\![\Sigma]\!] \to [\![\Sigma]\!] \\ [\![\mathsf{InvMask}]\!] \triangleq \Delta\wp(\mathbb{N}) & [\![\mathsf{Ectx}]\!] \triangleq \Delta\, Ectx & [\![\mathsf{Prop}]\!] \triangleq Prop \\ [\![\mathsf{Monoid}]\!] \triangleq \Delta|M| & [\![\mathsf{State}]\!] \triangleq \Delta\, State & \end{array}$$

An environment Γ is interpreted as the set of maps ρ, with $\mathrm{dom}(\rho) = \mathrm{dom}(\Gamma)$ and $\rho(x) \in [\![\Gamma(x)]\!]$, and $\rho \stackrel{n}{=} \rho' \iff n = 0 \vee \big(\mathrm{dom}(\rho) = \mathrm{dom}(\rho') \wedge \forall x \in \mathrm{dom}(\rho).\ \rho(x) \stackrel{n}{=} \rho'(x)\big)$.

Validity $\boxed{valid : \wp(Prop) \in Sets}$

$$valid(p) \iff \forall n \in \mathbb{N}.\ \forall r \in Res.\ \forall W \in World.\ (n, r) \in p(W)$$

Later modality $\boxed{\triangleright : Prop \to Prop \in \mathcal{U}}$

$$\triangleright p \triangleq \lambda W.\ \{\, (n+1, r) \mid (n, r) \in p(W) \,\} \cup \{\, (0, r) \mid r \in Res \,\}$$

Always modality $\boxed{\Box : Prop \to Prop \in \mathcal{U}}$

$$\Box p \triangleq \lambda W.\ \{\, (n, r) \mid (n, \epsilon) \in p(W) \,\}$$

Invariant definition $\boxed{inv : \Delta(\mathbb{N}) \times Prop \to Prop \in \mathcal{U}}$

$$inv(\iota, p) \triangleq \lambda W.\ \{\, (n, r) \mid \iota \in \mathrm{dom}(W) \wedge W(\iota) \stackrel{n+1}{=}_{PreProp} \xi(p) \,\}$$

Lemma 1. *inv is well-defined: $inv(\iota, p)$ is a valid proposition (this amounts to showing non-expansiveness), and inv itself is a non-expansive map.*

Erasure $\boxed{- \models_{-}\ -; -; - : \Delta(State) \times \Delta(\wp(\mathbb{N})) \times \Delta(Res) \times \Delta(Res) \times World \to \wp^{\downarrow}(\mathbb{N}) \in \mathcal{U}}$

$$\varsigma \models_{\mathcal{E}} r; s; W = \{\, n + 1 \in \mathbb{N} \mid (r \bullet s).\pi = \varsigma \wedge (n, s) \in \Pi_{\iota \in \mathcal{E} \cap \mathrm{dom}(W)} \xi^{-1}(W(\iota))(W) \,\} \cup \{0\}$$

Lemma 2. *$- \models_{-}\ -; -; -$ is well-defined: It maps into $\wp^{\downarrow}(\mathbb{N})$, and that map is non-expansive.*

Lemma 3.

$$\begin{array}{l} \forall \varsigma \in \Delta(State).\ \forall \mathcal{E}_1, \mathcal{E}_2 \in \Delta(\wp(\mathbb{N})).\ \forall r, s \in \Delta(Res).\ \forall W \in World. \\ \quad \mathcal{E}_1 \subseteq \mathcal{E}_2 \implies (\varsigma \models_{\mathcal{E}_2} r; s; W) \subseteq (\varsigma \models_{\mathcal{E}_1} r; s; W) \end{array}$$

Lemma 4.

$$\forall n \in \mathbb{N}.\ \forall W_1, W_1', W_2 \in World.\ W_1 \stackrel{n}{=} W_2 \wedge W_1 \leq W_1' \implies \exists W_2' \in World.\ W_1' \stackrel{n}{=} W_2' \wedge W_2 \leq W_2'$$

View-shift $\boxed{vs : \Delta(\wp(\mathbb{N})) \times \Delta(\wp(\mathbb{N})) \times \mathit{Prop} \to \mathit{Prop} \in \mathcal{U}}$

$$\begin{aligned} vs^{\mathcal{E}_2}_{\mathcal{E}_1}(q) = \lambda W.\ \{\, (n, r) \mid \forall W_F \geq W.\ \forall s, r_F, \mathcal{E}_F, \varsigma.\ \forall k \leq n. \\ k \in (\varsigma \models_{\mathcal{E}_1 \cup \mathcal{E}_F} r \bullet r_F; s; W_F) \land k > 0 \land \mathcal{E}_F \# (\mathcal{E}_1 \cup \mathcal{E}_2) \implies \\ \exists W' \geq W_F.\ \exists r', s'.\ k \in (\varsigma \models_{\mathcal{E}_2 \cup \mathcal{E}_F} r' \bullet r_F; s'; W') \land (k, r') \in q(W') \,\} \end{aligned}$$

Lemma 5. *vs is well-defined: $vs^{\mathcal{E}_2}_{\mathcal{E}_1}(q)$ is a valid proposition, and vs is a non-expansive map.*

Lemma 6.

$$\begin{aligned} &\forall \mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3 \in \Delta(\wp(\mathbb{N})).\ \forall p, q \in \mathit{Prop}.\ \forall W \in \mathit{World}.\ \forall n \in \mathbb{N}. \\ &\quad \mathcal{E}_2 \subseteq \mathcal{E}_1 \cup \mathcal{E}_3 \land \left(\forall W' \geq W.\ \forall r \in \mathit{Res}.\ \forall k \leq n.\ (k, r) \in p(W') \implies (k, r) \in vs^{\mathcal{E}_3}_{\mathcal{E}_2}(q)(W')\right) \\ &\qquad \implies \forall r \in \mathit{Res}.\ (n, r) \in vs^{\mathcal{E}_2}_{\mathcal{E}_1}(p)(W) \implies (n, r) \in vs^{\mathcal{E}_3}_{\mathcal{E}_1}(q)(W) \end{aligned}$$

Lemma 7.

$$\begin{aligned} &\forall \iota \in \mathbb{N}.\ \forall p \in \mathit{Prop}.\ \forall W \in \mathit{World}.\ \forall r \in \mathit{Res}.\ \forall n \in \mathbb{N}. \\ &\quad (n, r) \in \mathit{inv}(\iota, p)(W) \implies (n, r) \in vs^{\emptyset}_{\{\iota\}}(\triangleright p)(W) \end{aligned}$$

Timeless $\boxed{\mathit{timeless} : \mathit{Prop} \to \mathit{Prop}}$

$$\begin{aligned} \mathit{timeless}(p) \triangleq \lambda W.\ \{\, (n, r) \mid \forall W' \geq W.\ \forall k \leq n.\ \forall r' \in \mathit{Res}. \\ k > 0 \land (k - 1, r') \in p(W') \implies (k, r') \in p(W') \,\} \end{aligned}$$

Lemma 8. *timeless is well-defined: timeless(p) is a valid proposition, and timeless is a non-expansive map.*

Weakest precondition $\boxed{wp : \Delta(\wp(\mathbb{N})) \times \Delta(\mathit{Exp}) \times (\Delta(\mathit{Val}) \to \mathit{Prop}) \to \mathit{Prop} \in \mathcal{U}}$

$$\begin{aligned} wp_{\mathcal{E}}(e, q) \triangleq \lambda W.\ \{\, (n, r) \mid \forall W_F \geq W; k \leq n; s, r_F; \varsigma; \mathcal{E}_F \# \mathcal{E}.\ k > 0 \land k \in (\varsigma \models_{\mathcal{E} \cup \mathcal{E}_F} r \bullet r_F; s; W_F) \implies \\ (e \in \mathit{Val} \implies \exists W' \geq W_F.\ \exists r', s'. \\ \quad k \in (\varsigma \models_{\mathcal{E} \cup \mathcal{E}_F} r' \bullet r_F; s'; W') \land (k, r') \in q(e)(W')) \land \\ (\forall K, e_0, e_0', \varsigma'.\ e = K[e_0] \land \varsigma; e_0 \to \varsigma'; e_0' \implies \exists W' \geq W_F.\ \exists r', s'. \\ \quad k - 1 \in (\varsigma' \models_{\mathcal{E} \cup \mathcal{E}_F} r' \bullet r_F; s'; W') \land (k - 1, r') \in wp_{\mathcal{E}}(K[e_0'], q)(W')) \land \\ (\forall K, e'.\ e = K[\mathsf{fork}\ e'] \implies \exists W' \geq W_F.\ \exists r', s', r_1', r_2'. \\ \quad k - 1 \in (\varsigma \models_{\mathcal{E} \cup \mathcal{E}_F} r' \bullet r_F; s'; W') \land r' = r_1' \bullet r_2' \land \\ \quad (k - 1, r_1') \in wp_{\mathcal{E}}(K[\mathsf{fRet}], q)(W') \land (k - 1, r_2') \in wp_{\mathcal{E}}(e', \lambda\_.\ \top)(W')) \,\} \end{aligned}$$

Lemma 9. *wp is well-defined: $wp_{\mathcal{E}}(e, q)$ is a valid proposition, and wp is a non-expansive map. Besides, the dependency on the recursive occurrence is contractive, so wp has a fixed-point.*

Interpretation of terms $\boxed{[\![\Gamma \vdash t : \Sigma]\!] : [\![\Gamma]\!] \to [\![\Sigma]\!] \in \mathcal{U}}$

$$\begin{aligned} [\![\Gamma \vdash x : \Sigma]\!]_\gamma &= \gamma(x) \\ [\![\Gamma \vdash \lambda x.\ t : \Sigma \to \Sigma']\!]_\gamma &= \lambda v : [\![\Sigma]\!].\ [\![\Gamma, x : \Sigma \vdash t : \Sigma']\!]_{\gamma[x \mapsto v]} \\ [\![\Gamma \vdash t\ u : \Sigma']\!]_\gamma &= [\![\Gamma \vdash t : \Sigma \to \Sigma']\!]_\gamma([\![\Gamma \vdash u : \Sigma]\!]_\gamma) \\ [\![\Gamma \vdash () : 1]\!]_\gamma &= \star \\ [\![\Gamma \vdash (t_1, t_2) : \Sigma_1 \times \Sigma_2]\!]_\gamma &= ([\![\Gamma \vdash t_1 : \Sigma_1]\!]_\gamma, [\![\Gamma \vdash t_2 : \Sigma_2]\!]_\gamma) \\ [\![\Gamma \vdash \pi_i\ t : \Sigma_1]\!]_\gamma &= \pi_i([\![\Gamma \vdash t : \Sigma_1 \times \Sigma_2]\!]_\gamma) \end{aligned}$$

$$\begin{aligned}
\llbracket \Gamma \vdash \bot : \mathsf{Monoid} \rrbracket_\gamma &= \bot \\
\llbracket \Gamma \vdash \epsilon : \mathsf{Monoid} \rrbracket_\gamma &= \epsilon \\
\llbracket \Gamma \vdash a \cdot b : \mathsf{Monoid} \rrbracket_\gamma &= \llbracket \Gamma \vdash a : \mathsf{Monoid} \rrbracket_\gamma \cdot \llbracket \Gamma \vdash b : \mathsf{Monoid} \rrbracket_\gamma
\end{aligned}$$

$$\begin{aligned}
\llbracket \Gamma \vdash t =_\Sigma u : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \{\, (n,r) \mid \llbracket \Gamma \vdash t : \Sigma \rrbracket_\gamma \overset{n+1}{=} \llbracket \Gamma \vdash u : \Sigma \rrbracket_\gamma \,\} \\
\llbracket \Gamma \vdash \mathsf{False} : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \emptyset \\
\llbracket \Gamma \vdash \mathsf{True} : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \mathbb{N} \times \mathit{Res} \\
\llbracket \Gamma \vdash P \wedge Q : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \llbracket \Gamma \vdash P : \mathsf{Prop} \rrbracket_\gamma(W) \cap \llbracket \Gamma \vdash Q : \mathsf{Prop} \rrbracket_\gamma(W) \\
\llbracket \Gamma \vdash P \vee Q : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \llbracket \Gamma \vdash P : \mathsf{Prop} \rrbracket_\gamma(W) \cup \llbracket \Gamma \vdash Q : \mathsf{Prop} \rrbracket_\gamma(W) \\
\llbracket \Gamma \vdash P \Rightarrow Q : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \{\, (n,r) \mid \forall n' \le n.\, \forall W' \ge W.\, \forall r' \ge r. \\
&\qquad (n',r') \in \llbracket \Gamma \vdash P : \mathsf{Prop} \rrbracket_\gamma(W') \\
&\qquad \implies (n',r') \in \llbracket \Gamma \vdash Q : \mathsf{Prop} \rrbracket_\gamma(W') \,\} \\
\llbracket \Gamma \vdash \forall x : \Sigma.\, P : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \{\, (n,r) \mid \forall v \in \llbracket \Sigma \rrbracket.\, (n,r) \in \llbracket \Gamma, x : \Sigma \vdash P : \mathsf{Prop} \rrbracket_{\gamma[x \mapsto v]}(W) \,\} \\
\llbracket \Gamma \vdash \exists x : \Sigma.\, P : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \{\, (n,r) \mid \exists v \in \llbracket \Sigma \rrbracket.\, (n,r) \in \llbracket \Gamma, x : \Sigma \vdash P : \mathsf{Prop} \rrbracket_{\gamma[x \mapsto v]}(W) \,\}
\end{aligned}$$

$$\begin{aligned}
\llbracket \Gamma \vdash \Box P : \mathsf{Prop} \rrbracket_\gamma &= \Box \llbracket \Gamma \vdash P : \mathsf{Prop} \rrbracket_\gamma \\
\llbracket \Gamma \vdash \triangleright P : \mathsf{Prop} \rrbracket_\gamma &= \triangleright \llbracket \Gamma \vdash P : \mathsf{Prop} \rrbracket_\gamma \\
\llbracket \Gamma \vdash \mu x.\, \varphi : \Sigma \to \mathsf{Prop} \rrbracket_\gamma &= \mathit{fix}(\lambda v : \llbracket \Sigma \to \mathsf{Prop} \rrbracket.\, \llbracket \Gamma, x : \Sigma \to \mathsf{Prop} \vdash \varphi : \Sigma \to \mathsf{Prop} \rrbracket_{\gamma[x \mapsto v]}) \\
\llbracket \Gamma \vdash P * Q : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \{\, (n,r) \mid \exists r_1, r_2.\, r = r_1 \bullet r_2 \wedge \\
&\qquad (n, r_1) \in \llbracket \Gamma \vdash P : \mathsf{Prop} \rrbracket_\gamma \wedge \\
&\qquad (n, r_2) \in \llbracket \Gamma \vdash Q : \mathsf{Prop} \rrbracket_\gamma \,\} \\
\llbracket \Gamma \vdash P \mathrel{-\!\!*} Q : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \{\, (n,r) \mid \forall n' \le n.\, \forall W' \ge W.\, \forall r'. \\
&\qquad (n',r') \in \llbracket \Gamma \vdash P : \mathsf{Prop} \rrbracket_\gamma(W') \wedge r \mathbin{\#} r' \\
&\qquad \implies (n', r \bullet r') \in \llbracket \Gamma \vdash Q : \mathsf{Prop} \rrbracket_\gamma(W') \} \\
\llbracket \Gamma \vdash \boxed{P}^{\iota} : \mathsf{Prop} \rrbracket_\gamma &= \mathit{inv}(\llbracket \Gamma \vdash \iota : \mathsf{Name} \rrbracket_\gamma, \llbracket \Gamma \vdash P : \mathsf{Prop} \rrbracket_\gamma) \\
\llbracket \Gamma \vdash \lceil a \rceil : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \{\, (n,r) \mid r.g \ge \llbracket \Gamma \vdash a : \mathsf{Monoid} \rrbracket_\gamma \,\} \\
\llbracket \Gamma \vdash \lfloor \varsigma \rfloor : \mathsf{Prop} \rrbracket_\gamma &= \lambda W.\, \{\, (n,r) \mid r.\pi = \llbracket \Gamma \vdash \varsigma : \mathsf{State} \rrbracket_\gamma \,\}
\end{aligned}$$

$$\begin{aligned}
\llbracket \Gamma \vdash \mathsf{vs}^{\mathcal{E}_2}_{\mathcal{E}_1}(P) : \mathsf{Prop} \rrbracket_\gamma &= vs^{\llbracket \Gamma \vdash \mathcal{E}_2 : \mathsf{InvMask} \rrbracket_\gamma}_{\llbracket \Gamma \vdash \mathcal{E}_1 : \mathsf{InvMask} \rrbracket_\gamma}(\llbracket \Gamma \vdash P : \mathsf{Prop} \rrbracket_\gamma) \\
\llbracket \Gamma \vdash \mathsf{wp}_{\mathcal{E}}(e, \varphi) : \mathsf{Prop} \rrbracket_\gamma &= wp_{\llbracket \Gamma \vdash \mathcal{E} : \mathsf{InvMask} \rrbracket_\gamma}(\llbracket \Gamma \vdash e : \mathsf{Exp} \rrbracket_\gamma, \llbracket \Gamma \vdash \varphi : \mathsf{Val} \to \mathsf{Prop} \rrbracket_\gamma)
\end{aligned}$$

Interpretation of entailment $\boxed{\llbracket \Gamma \mid \Theta \vdash P \rrbracket : 2 \in \mathit{Sets}}$

$$\begin{aligned}
\llbracket \Gamma \mid \Theta \vdash Q \rrbracket \triangleq\ & \forall n \in \mathbb{N}.\, \forall W \in \mathit{World}.\, \forall r \in \mathit{Res}.\, \forall \gamma \in \llbracket \Gamma \rrbracket, \\
& \big(\forall Q \in \Theta.\, (n,r) \in \llbracket \Gamma \vdash Q : \mathsf{Prop} \rrbracket_\gamma(W)\big) \implies (n,r) \in \llbracket \Gamma \vdash P : \mathsf{Prop} \rrbracket_\gamma(W)
\end{aligned}$$

5 Proof theory

The judgment $\Gamma \mid \Theta \vdash P$ says that with free variables Γ, proposition P holds whenever all assumptions Θ hold. We implicitly assume that an arbitrary variable context, Γ, is added to every constituent of the rules. Axioms $P \Rightarrow Q$ stand for judgments $\Gamma \mid \cdot \vdash P \Rightarrow Q$ with no assumptions. (Bi-implications are analogous.)

5.1 Laws of intuitionistic higher-order logic with guarded recursion over a simply-typed lambda calculus

Standard.

Soundness follows from the theorem that $\mathcal{U}(\_, \mathit{Prop}) : \mathcal{U}^{\mathrm{op}} \to \mathrm{Poset}$ is a hyperdoctrine.

5.2 Axioms from the logic of (affine) bunched implications

$$P * Q \Leftrightarrow Q * P \qquad (P \lor Q) * R \Leftrightarrow (P * R) \lor (Q * R)$$
$$(P * Q) * R \Leftrightarrow P * (Q * R) \qquad (P \land Q) * R \Rightarrow (P * R) \land (Q * R)$$
$$P * Q \Rightarrow P \qquad (\exists x.\ P) * Q \Leftrightarrow \exists x.\ (P * Q)$$
$$(\forall x.\ P) * Q \Rightarrow \forall x.\ (P * Q)$$

$$\frac{\Theta, P_1 \vdash Q_1 \qquad \Theta, P_2 \vdash Q_2}{\Theta, P_1 * P_2 \vdash Q_1 * Q_2} \qquad \frac{\Theta, P * Q \vdash R}{\Theta, P \vdash Q \mathrel{-\!\!*} R} \qquad \frac{\Theta, P \vdash Q \mathrel{-\!\!*} R}{\Theta, P * Q \vdash R}$$

5.3 Laws for ghosts and physical resources

$$\boxed{a} * \boxed{b} \Leftrightarrow \boxed{a \cdot b} \qquad \lfloor \varsigma \rfloor * \lfloor \varsigma' \rfloor \Rightarrow \mathsf{False}$$
$$\mathsf{True} \Rightarrow \boxed{\epsilon}$$
$$\boxed{\bot} \Rightarrow \mathsf{False}$$

5.4 Laws for the later modality

$$\textsc{Mono}\ \frac{\Theta \vdash P}{\Theta \vdash \triangleright P} \qquad \textsc{Löb}\ \frac{\Theta, \triangleright P \vdash P}{\Theta \vdash P}$$

$$\triangleright \Box P \Leftrightarrow \Box \triangleright P \qquad \triangleright \forall x.\ P \Leftrightarrow \forall x.\ \triangleright P$$
$$\triangleright (P \land Q) \Leftrightarrow \triangleright P \land \triangleright Q \qquad \triangleright \exists x.\ P \Leftrightarrow \exists x.\ \triangleright P$$
$$\triangleright (P \lor Q) \Leftrightarrow \triangleright P \lor \triangleright Q \qquad \triangleright (P * Q) \Leftrightarrow \triangleright P * \triangleright Q$$

5.5 Laws for the always modality

$$\textsc{Necessity}\ \Box P \Rightarrow P \qquad \Box\mathrm{I}\ \frac{\Box\Theta \vdash P}{\Box\Theta \vdash \Box P}$$

$$\Box(t =_\Sigma u) \Leftrightarrow t =_\Sigma u \qquad \Box(P \land Q) \Leftrightarrow \Box P \land \Box Q$$
$$\Box P * Q \Leftrightarrow \Box P \land Q \qquad \Box(P \lor Q) \Leftrightarrow \Box P \lor \Box Q$$
$$\Box(P \Rightarrow Q) \Rightarrow \Box P \Rightarrow \Box Q \qquad \Box \forall x.\ P \Leftrightarrow \forall x.\ \Box P$$
$$\Box \exists x.\ P \Leftrightarrow \exists x.\ \Box P$$

Note that $\Box$ binds more tightly than $*$, $\land$, $\lor$, and $\Rightarrow$.

6 Program logic

Hoare triples and view shifts are syntactic sugar for weakest (liberal) preconditions and primitive view shifts, respectively:

$$\{P\}\ e\ \{v.\ Q\}_{\mathcal{E}} \triangleq \Box(P \Rightarrow \mathsf{wp}_{\mathcal{E}}(e, \lambda v.\ Q)) \qquad P\ {}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2} Q \triangleq \Box(P \Rightarrow \mathsf{vs}^{\mathcal{E}_2}_{\mathcal{E}_1}(Q))$$
$$P\ {}^{\mathcal{E}_1}\!\Leftrightarrow^{\mathcal{E}_2} Q \triangleq P\ {}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2} Q \land Q\ {}^{\mathcal{E}2}\!\Rightarrow^{\mathcal{E}_1} P$$

We write just one mask for a view shift when $\mathcal{E}_1 = \mathcal{E}_2$. The convention for omitted masks is generous: An omitted $\mathcal{E}$ is $\top$ for Hoare triples and $\emptyset$ for view shifts.

Henceforward, we implicitly assume a proof context, Θ, is added to every constituent of the rules. Generally, this is an arbitrary proof context. We write $\vdash_{\Box}$ to denote judgments that can only be extended with a boxed proof context.

6.1 Hoare triples

$$\textsc{Ret}\quad \{\mathsf{True}\}\ w\ \{v.\ v = w\}_{\mathcal{E}}$$

$$\textsc{Bind}\quad \frac{\{P\}\ e\ \{v.\ Q\}_{\mathcal{E}} \qquad \forall v.\ \{Q\}\ K[v]\ \{w.\ R\}_{\mathcal{E}}}{\{P\}\ K[e]\ \{w.\ R\}_{\mathcal{E}}}$$

$$\textsc{Csq}\quad \frac{P \Rrightarrow P' \qquad \{P'\}\ e\ \{v.\ Q'\}_{\mathcal{E}} \qquad \forall v.\ Q' \Rrightarrow Q}{\{P\}\ e\ \{v.\ Q\}_{\mathcal{E}}}$$

$$\textsc{Frame}\quad \frac{\{P\}\ e\ \{v.\ Q\}_{\mathcal{E}}}{\{P * R\}\ e\ \{v.\ Q * R\}_{\mathcal{E}\uplus\mathcal{E}'}}$$

$$\textsc{AFrame}\quad \frac{\{P\}\ e\ \{v.\ Q\}_{\mathcal{E}} \qquad e \text{ phys. atomic}}{\{P * \triangleright R\}\ e\ \{v.\ Q * R\}_{\mathcal{E}\uplus\mathcal{E}'}}$$

$$\textsc{Fork}\quad \frac{\{P\}\ e\ \{\_.\ \mathsf{True}\}_{\mathcal{E}}}{\{\triangleright P * \triangleright Q\}\ \mathbf{fork}\ e\ \{v.\ v = \mathsf{fRet} \wedge Q\}_{\mathcal{E}}}$$

$$\textsc{ACsq}\quad \frac{P\ {}^{\mathcal{E}\uplus\mathcal{E}'}\!\Rrightarrow^{\mathcal{E}}\ P' \qquad \{P'\}\ e\ \{v.\ Q'\}_{\mathcal{E}} \qquad \forall v.\ Q'\ {}^{\mathcal{E}}\!\Rrightarrow^{\mathcal{E}\uplus\mathcal{E}'}\ Q \qquad e \text{ phys. atomic}}{\{P\}\ e\ \{v.\ Q\}_{\mathcal{E}\uplus\mathcal{E}'}}$$

6.2 View shifts

$$\textsc{NewInv}\quad \frac{\mathsf{infinite}(\mathcal{E})}{\triangleright P \Rrightarrow_{\mathcal{E}} \exists \iota \in \mathcal{E}.\ \boxed{P}^{\iota}}$$

$$\textsc{FpUpd}\quad \frac{a \rightsquigarrow B}{\boxed{a} \Rrightarrow \exists b \in B.\ \boxed{b}}$$

$$\textsc{VSTrans}\quad \frac{P\ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}\ Q \qquad Q\ {}^{\mathcal{E}_2}\!\Rrightarrow^{\mathcal{E}_3}\ R \qquad \mathcal{E}_2 \subseteq \mathcal{E}_1 \cup \mathcal{E}_3}{P\ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_3}\ R}$$

$$\textsc{VSImp}\quad \frac{\Box(P \Rightarrow Q)}{P \Rrightarrow_{\emptyset} Q}$$

$$\textsc{VSFrame}\quad \frac{P\ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}\ Q}{P * R\ {}^{\mathcal{E}_1\uplus\mathcal{E}'}\!\Rrightarrow^{\mathcal{E}_2\uplus\mathcal{E}'}\ Q * R}$$

$$\textsc{VSTimeless}\quad \frac{\mathsf{timeless}(P)}{\triangleright P \Rrightarrow P}$$

$$\textsc{InvOpen}\quad \boxed{P}^{\iota} \vdash \mathsf{True}\ {}^{\{\iota\}}\!\Rrightarrow^{\emptyset}\ \triangleright P$$

$$\textsc{InvClose}\quad \boxed{P}^{\iota} \vdash \triangleright P\ {}^{\emptyset}\!\Rrightarrow^{\{\iota\}}\ \mathsf{True}$$

Note that $\mathsf{timeless}(P)$ means that P does not depend on the step index. Furthermore,

$$a \rightsquigarrow B \triangleq \Box\forall a_f.\ a \# a_f \Rightarrow \exists b \in B.\ b \# a_f$$

6.3 Derived rules

Derived structural rules. The following are easily derived by unfolding the sugar for Hoare triples and view shifts.

$$\textsc{Disj}\quad \frac{\{P\}\ e\ \{v.\ R\}_{\mathcal{E}} \qquad \{Q\}\ e\ \{v.\ R\}_{\mathcal{E}}}{\{P \vee Q\}\ e\ \{v.\ R\}_{\mathcal{E}}}$$

$$\textsc{VSDisj}\quad \frac{P\ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}\ R \qquad Q\ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}\ R}{P \vee Q\ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}\ R}$$

$$\textsc{Exist}\quad \frac{\forall x.\ \{P\}\ e\ \{v.\ Q\}_{\mathcal{E}}}{\{\exists x.\ P\}\ e\ \{v.\ Q\}_{\mathcal{E}}}$$

$$\textsc{VSExist}\quad \frac{\forall x.\ (P\ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}\ Q)}{(\exists x.\ P)\ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}\ Q}$$

$$\textsc{BoxOut}\quad \frac{\Box Q \vdash_{\Box} \{P\}\ e\ \{v.\ R\}_{\mathcal{E}}}{\{P \wedge \Box Q\}\ e\ \{v.\ R\}_{\mathcal{E}}}$$

$$\textsc{VSBoxOut}\quad \frac{\Box Q \vdash_{\Box} P\ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}\ R}{P \wedge \Box Q\ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}\ R}$$

$$\textsc{False}\quad \{\mathsf{False}\}\ e\ \{v.\ P\}_{\mathcal{E}}$$

$$\textsc{VSFalse}\quad \mathsf{False}\ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2}\ P$$

The proofs all follow the same pattern, so we only show two of them in detail.

Proof of Exist. After unfolding the syntactic sugar for Hoare triples and removing the boxes from premise and conclusion, our goal becomes

$$(\exists x.\ P(x)) \Rightarrow \mathsf{wp}_{\mathcal{E}}(e, \lambda v.\ Q)$$

(remember that x is free in P) and the premise reads

$$\forall x.\ P(x) \Rightarrow \mathsf{wp}_{\mathcal{E}}(e, \lambda v.\ Q).$$

Let x be given and assume $P(x)$. To show $\mathsf{wp}_{\mathcal{E}}(e, \lambda v.\ Q)$, apply the premise to x and $P(x)$.

For the other direction, assume

$$\{\exists x.\ P(x)\}\ e\ \{v.\ Q\}_{\mathcal{E}}$$

and let x be given. We have to show $\{P(x)\}\ e\ \{v.\ Q\}_{\mathcal{E}}$. This trivially follows from Csq with $P(x) \Rightarrow \exists x.\ P(x)$. □

Proof of BoxOut. After unfolding the syntactic sugar for Hoare triples, our goal becomes

$$\Box\Theta \vdash \Box\big(P \wedge \Box Q \Rightarrow \mathsf{wp}_{\mathcal{E}}(e, \lambda v.\ R)\big) \tag{1}$$

while our premise reads

$$\Box\Theta, \Box Q \vdash \Box(P \Rightarrow \mathsf{wp}_{\mathcal{E}}(e, \lambda v.\ R)) \tag{2}$$

By the introduction rules for $\Box$ and implication, it suffices to show

$$(\Box\Theta), P, \Box Q \vdash \mathsf{wp}_{\mathcal{E}}(e, \lambda v.\ R)$$

By modus ponens and Necessity, it suffices to show (2), which is exactly our assumption.

For the other direction, assume (1). We have to show (2). By $\Box$I and implication introduction, it suffices to show

$$(\Box\Theta), P, \Box Q \vdash \mathsf{wp}_{\mathcal{E}}(e, \lambda v.\ R)$$

which easily follows from (1). □

Derived rules for invariants. Invariants can be opened around atomic expressions and view shifts.

$$\textsc{Inv}\ \frac{\{\triangleright R * P\}\ e\ \{v.\ \triangleright R * Q\}_{\mathcal{E}} \qquad e \text{ phys. atomic}}{\boxed{R}^{\iota} \vdash \{P\}\ e\ \{v.\ Q\}_{\mathcal{E} \uplus \{\iota\}}} \qquad \textsc{VSInv}\ \frac{\triangleright P * Q \ {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2} \triangleright P * R}{\boxed{P}^{\iota} \vdash Q \ {}^{\mathcal{E}_1 \uplus \{\iota\}}\!\Rrightarrow^{\mathcal{E}_2 \uplus \{\iota\}} R}$$

Proof of Inv. Use ACsq with $\mathcal{E}_1 \triangleq \mathcal{E} \cup \{\iota\}$, $\mathcal{E}_2 \triangleq \mathcal{E}$. The view shifts are obtained by InvOpen and InvClose with framing of $\mathcal{E}$ and P or Q, respectively. □

Proof of VSInv. Analogous to the proof of Inv, using VSTrans instead of ACsq. □

6.3.1 Unsound rules

Some rule suggestions (or rather, wishes) keep coming up, which are unsound. We collect them here.

$$\frac{P \Rrightarrow \mathsf{False}}{\triangleright P \Rrightarrow \triangleright \mathsf{False}}$$

To see why this does not work, let

$$P \triangleq \boxed{a}^{\iota} * \boxed{a}$$

with $a \cdot a = \bot$. Clearly, $P \Rrightarrow \mathsf{False}$: Just open the invariant, obtain False, and close it again. But with $\triangleright P$, as invariant assertions are vacuous at step-index 0, our premise is vacuous at indices 0 and 1. In the proof, we cannot show the view shift at index 1: We cannot use the premise view shift at index 0 (because view shifts are trivial at that step-index), so knowing $\triangleright P$ is of no use. Hence we cannot proceed.

For similar reasons, the following rules are unsound.

$$\frac{P \Rrightarrow Q}{\triangleright P \Rrightarrow \triangleright Q} \qquad \frac{\triangleright(P \Rrightarrow Q)}{\triangleright P \Rrightarrow \triangleright Q}$$

6.4 Adequacy

The adequacy statement reads as follows:

$$\begin{array}{l}\forall \mathcal{E}, e, v, \varphi, i, \varsigma, \varsigma', T'.\\ (\vdash \{\lfloor \varsigma \rfloor\}\ e\ \{x.\ \varphi(x)\}_{\mathcal{E}}) \implies\\ \varsigma; [i \mapsto e] \to^* \varsigma'; [i \mapsto v] \uplus T' \implies\\ \varphi(v)\end{array}$$

where φ can mention neither resources nor invariants.

6.5 Axiom lifting

The following lemmas help in proving axioms for a particular language; for example, see §12. The first applies to expressions with side-effects, and the second to side-effect-free expressions.

$$\begin{array}{l}\forall e, \varsigma, \varphi, P, Q, \mathcal{E}.\\ \mathsf{reducible}(e) \implies\\ (\forall e_2, \varsigma_2.\ \varsigma; e \to \varsigma_2; e_2 \implies \varphi(e_2, \varsigma_2)) \implies\\ \vdash \big((\forall e', \varsigma'.\ \varphi(e', \varsigma') \Rightarrow \{P\}\ e'\ \{v.\ Q(v, \varsigma')\}_{\mathcal{E}}) \Rightarrow \{\triangleright P * \lfloor \varsigma \rfloor\}\ e\ \{v.\ \exists \varsigma'.\ \lfloor \varsigma' \rfloor * Q(v, \varsigma')\}_{\mathcal{E}}\big)\end{array}$$

$$\begin{array}{l}\forall e, \varphi, P, Q, \mathcal{E}.\\ \mathsf{reducible}(e) \implies\\ (\forall \varsigma, e_2, \varsigma_2.\ \varsigma; e \to \varsigma_2; e_2 \implies \varsigma_2 = \varsigma \land \varphi(e_2)) \implies\\ \vdash \big((\forall e'.\ \varphi(e') \Rightarrow \{P\}\ e'\ \{v.\ Q\}_{\mathcal{E}}) \Rightarrow \{\triangleright P\}\ e\ \{v.\ Q\}_{\mathcal{E}}\big)\end{array}$$

Note that φ is a meta-logic predicate—it does not depend on any world or resources being owned.

The following specializations cover all cases of a heap-manipulating lambda calculus like $F_{\mu!}$.

$$\begin{array}{l}\forall e, e', P, Q, \mathcal{E}.\\ \mathsf{reducible}(e) \implies\\ (\forall \varsigma, e_2, \varsigma_2.\ \varsigma; e \to \varsigma_2; e_2 \implies \varsigma_2 = \varsigma \land e_2 = e') \implies\\ \vdash (\{P\}\ e'\ \{v.\ Q\}_{\mathcal{E}} \Rightarrow \{\triangleright P\}\ e\ \{v.\ Q\}_{\mathcal{E}})\end{array}$$

$$\begin{array}{l}\forall e, \varsigma, \varphi, \mathcal{E}.\\ \mathsf{atomic}(e) \implies\\ \big(\forall e_2, \varsigma_2.\ \varsigma; e \to \varsigma_2; e_2 \implies \varphi(e_2, \varsigma_2)\big) \implies\\ \vdash (\{\lfloor \varsigma \rfloor\}\ e\ \{v.\ \exists \varsigma'.\ \lfloor \varsigma' \rfloor \land \varphi(v, \varsigma')\}_{\mathcal{E}})\end{array}$$

The first is restricted to deterministic pure reductions, like β-reduction. The second is suited to proving triples for (possibly non-deterministic) atomic expressions; for example, with $e \triangleq\ !\ell$ (dereferencing ℓ) and $\varsigma \triangleq h \cdot \ell \mapsto w$ and $\varphi(v, \varsigma') \triangleq \varsigma' = (h \cdot \ell \mapsto w) \land v = w$, one obtains the axiom $\forall h, \ell, w.\ \{\lfloor h \cdot \ell \mapsto w \rfloor\}\ !\ell\ \{v.\ v = w \land \lfloor h \cdot \ell \mapsto w \rfloor\}$.

Part II
Working with Iris

7 Monoid constructions

We will use the notation $|M|^+ \triangleq |M| \setminus \{\bot_M\}$ for the carrier of monoid M without zero. When we define a carrier, a zero element is always implicitly added (we do not explicitly give it), and all cases

of multiplication that are not defined (including those involving a zero element) go to that element.

To disambiguate which monoid an element is part of, we use the notation $a : M$ to denote an a s.t. $a \in |M|$.

When defining a monoid, we will show some *frame-preserving updates* $a \rightsquigarrow B$ that it supports. Remember that

$$a \rightsquigarrow B \triangleq \Box\forall a_f.\ a \mathbin{\#} a_f \Rightarrow \exists b \in B.\ b \mathbin{\#} a_f.$$

The rule FpUpd (and, later, GhostUpd) allows us to use such updates in Hoare proofs. The following principles generally hold for frame-preserving updates.

$$\frac{a \rightsquigarrow B}{a \rightsquigarrow B \cup B'} \qquad\qquad \frac{a \rightsquigarrow B}{a \cdot a_f \rightsquigarrow \{b \cdot a_f \mid b \in B\}}$$

Some of our constructions require or preserve *cancellativity*:

$$M \text{ cancellative} \triangleq \forall a_f, a, b \in |M|.\ a_f \cdot a = a_f \cdot b \neq \bot \Rightarrow a = b$$

7.1 Exclusive monoid

Given a set X, we define a monoid such that at most one $x \in X$ can be owned. Let $\mathrm{Ex}(X)$ be the monoid with carrier $X \uplus \{\epsilon\}$ and multiplication

$$a \cdot b \triangleq \begin{cases} a & \text{if } b = \epsilon \\ b & \text{if } a = \epsilon \end{cases}$$

The frame-preserving update

$$\frac{\textsc{ExUpd}}{x \in X}{x \rightsquigarrow a}$$

is easily shown, as the only possible frame for x is ϵ.

Exclusive monoids are cancellative.

Proof of cancellativity. If $a_f = \epsilon$, then the statement is trivial. If $a_f \neq \epsilon$, then we must have $a = b = \epsilon$, as otherwise one of the two products would be $\bot$. □

7.2 Product monoid

Given a family $(M_i)_{i \in I}$ of monoids (I countable), we construct a product monoid. Let $\prod_{i \in I} M_i$ be the monoid with carrier $\prod_{i \in I} |M_i|^+$ and point-wise multiplication, non-zero when *all* individual multiplications are non-zero. For $f \in \prod_{i \in I} |M_i|^+$, we write $f[i \mapsto a]$ for the disjoint union $f \uplus [i \mapsto a]$.

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

$$\frac{\textsc{ProdUpd}}{a \rightsquigarrow_{M_i} B}{f[i \mapsto a] \rightsquigarrow \{f[i \mapsto b] \mid b \in B\}}$$

Proof of ProdUpd. Assume some frame g and let $c \triangleq g(i)$. Since $f[i \mapsto a] \mathbin{\#} g$, we get $f \mathbin{\#} g$ and $a \mathbin{\#}_{M_i} c$. Thus there exists $b \in B$ such that $b \mathbin{\#}_{M_i} c$. It suffices to show $f[i \mapsto b] \mathbin{\#} g$. Since multiplication is defined pointwise, this is the case if all components are compatible. For i, we know this from $b \mathbin{\#}_{M_i} c$. For all the other components, from $f \mathbin{\#} g$. □

If every M_i is cancellative, then so is $\prod_{i \in I} M_i$.

Proof of cancellativity. Let $a, b, a_f \in \prod_{i \in I} |M_i|^+$, and assume $a_f \cdot a = a_f \cdot b \neq \bot$. By the definition of multiplication, this means that for all $i \in I$ we have $a_f(i) \cdot a(i) = a_f(i) \cdot b(i) \neq \bot_{M_i}$. As all base monoids are cancellative, we obtain $\forall i \in I.\ a(i) = b(i)$ from which we immediately get $a = b$. □

7.3 Fractional monoid

Given a set X, we define a monoid representing fractional ownership of at most one $x \in X$. Let $\textsc{Frac}(X)$ be the monoid with carrier $(((0,1] \cap \mathbb{Q}) \times X) \uplus \{\epsilon\}$ and multiplication

$$\begin{aligned}(q,a) \cdot (q',a') &\triangleq (q+q',a) \qquad \text{if } a = a' \text{ and } q + q' \leq 1\\ (q,a) \cdot \epsilon &\triangleq (q,a)\\ \epsilon \cdot (q,a) &\triangleq (q,a).\end{aligned}$$

We get the following frame-preserving update.

$$\frac{x \in X}{(1,x) \rightsquigarrow a}\ \textsc{FracUpd}$$

Proof of FracUpd. Assume some $f \# (1,x)$. This can only be $f = \epsilon$, so showing $f \# a$ is trivial. □

$\textsc{Frac}(X)$ is cancellative.

Proof of cancellativitiy. If $a_f = \epsilon$, we are trivially done. So let $a_f = (q_f, x_f)$. If $a = \epsilon$, then $b = \epsilon$ as otherwise the fractions could not match up. Again, we are trivially done. So let $a = (q_a, x_a)$ and $b = (q_b, x_b)$. We have $(q_f + q_a, x_f = x_a) = (q_g + q_b, x_f = x_b)$. We have to show $q_a = q_b$ and $x_a = x_b$. These are immediate. □

7.4 Finite partial function monoid

Given a countable set X and a monoid M, we construct a monoid representing finite partial functions from X to (non-unit, non-zero elements of) M. Let $\textsc{FpFun}(X, M)$ be the product monoid $\prod_{x \in X} M$, as defined in Section 7.2 but restricting the carrier to functions f where the set $\mathrm{dom}(f) \triangleq \{x \mid f(x) \neq \epsilon_M\}$ is finite. This is well-defined as the set of these f contains the unit and is closed under multiplication. (We identify finite partial functions from X to $|M|^+ \setminus \{\epsilon_M\}$ and total functions from X to $|M|^+$ with finite ϵ_M-support.)

We use two frame-preserving updates:

$$\frac{a \in |M|^+}{f \rightsquigarrow \{f[x \mapsto a] \mid x \notin \mathrm{dom}(f)\}}\ \textsc{FpFunAlloc} \qquad \frac{a \rightsquigarrow_M B}{f[i \mapsto a] \rightsquigarrow \{f[i \mapsto b] \mid b \in B\}}\ \textsc{FpFunUpd}$$

Rule FpFunUpd simply restates ProdUpd.

Proof of FpFunAlloc. Assume some $g \# f$. Since $\mathrm{dom}(f \cdot g)$ is finite, there will be some undefined element $x \notin \mathrm{dom}(f \cdot g)$. Let $f' \triangleq f[x \mapsto a]$. This is compatible with g, so we are done. □

We write $[x \mapsto a]$ for the function mapping x to a and everything else in X to ϵ.

7.5 Disposable monoid

Given a monoid M, we construct a monoid where, having full ownership of an element a of M, one can throw it away, transitioning to a dead element. Let $\textsc{Disp}(M)$ be the monoid with carrier $|M|^+ \uplus \{\dagger\}$ and multiplication

$$\begin{aligned}a \cdot b &\triangleq a \cdot_M b \qquad \text{if } a \#_M b\\ \dagger \cdot \dagger &\triangleq \dagger\\ \epsilon_M \cdot \dagger &\triangleq \dagger \cdot \epsilon_M \triangleq \dagger\end{aligned}$$

The unit is the same as in M.

The frame-preserving updates are

$$\frac{a \in |M|^+ \setminus \{\epsilon_M\} \qquad a \rightsquigarrow_M B}{a \rightsquigarrow B}\ \textsc{DispUpd} \qquad \frac{a \in |M|^+ \setminus \{\epsilon_M\} \qquad \forall b \in |M|^+.\ a \# b \Rightarrow b = \epsilon_M}{a \rightsquigarrow \dagger}\ \textsc{Dispose}$$

Proof of DispUpd. Assume a frame f. If $f = \dagger$, then $a = \epsilon_M$, which is a contradiction. Thus $f \in |M|^+$ and we can use $a \rightsquigarrow_M B$. □

Proof of Dispose. The second premiss says that a has no non-trivial frame in M. To show the update, assume a frame f in $\textsc{Disp}(M)$. Like above, we get $f \in |M|^+$, and thus $f = \epsilon_M$. But $\dagger \# \epsilon_M$ is trivial, so we are done. □

7.6 Authoritative monoid

Given a monoid M, we construct a monoid modeling someone owning an *authoritative* element x of M, and others potentially owning fragments $a \leq_M x$ of x. (If M is an exclusive monoid, the construction is very similar to a half-ownership monoid with two asymmetric halves.) Let $\textsc{Auth}(M)$ be the monoid with carrier

$$\left\{(x, a) \;\middle|\; x \in |\textsc{Ex}(|M|^+)|^+ \land a \in |M|^+ \land (x = \epsilon_{\textsc{Ex}(|M|^+)} \lor a \leq_M x)\right\}$$

and multiplication

$$(x, a) \cdot (y, b) \triangleq (x \cdot y, a \cdot b) \quad \text{if } x \# y \land a \# b \land (x \cdot y = \epsilon_{\textsc{Ex}(|M|^+)} \lor a \cdot b \leq_M x \cdot y)$$

Note that $(\epsilon_{\textsc{Ex}(|M|^+)}, \epsilon_M)$ is the unit and asserts no ownership whatsoever, but (ϵ_M, ϵ_M) asserts that the authoritative element is ϵ_M.

Let $x, a \in |M|^+$. We write $\bullet x$ for full ownership $(x, \epsilon_M) : \textsc{Auth}(M)$ and $\circ a$ for fragmental ownership $(\epsilon_{\textsc{Ex}(|M|^+)}, a)$ and $\bullet x, \circ a$ for combined ownership (x, a). If x or a is $\bot_M$, then the sugar denotes $\bot_{\textsc{Auth}(M)}$.

Frame-preserving updates are possible if we assume M cancellative:

$$\frac{M \text{ cancellative} \qquad a' \# b}{\bullet a \cdot b, \circ a \rightsquigarrow \bullet a' \cdot b, \circ a'}\ \textsc{AuthUpd}$$

Proof of AuthUpd. Assume some frame composeable with $\bullet a \cdot b, \circ a$: It must be of the form (ϵ, a_f) s.t. $a \# a_f$ and $a \cdot a_f \leq a \cdot b$. Note that $a \# b$. By cancellativity, we have $a_f \leq b$.

Now we have to show $a' \# a_f$ and $a' \cdot a_f \leq a' \cdot b$. The second part follows immediately from $a_f \leq b$. The first part follows from the inequality we just proved and $a' \# b$. □

7.7 Fractional heap monoid

By combining the fractional, finite partial function, and authoritative monoids, we construct two flavors of heaps with fractional permissions and mention their important frame-preserving updates. Hereinafter, we assume the set *Val* of values is countable.

Given a set Y, define $\textsc{FHeap}(Y) \triangleq \textsc{FpFun}(\mathit{Val}, \textsc{Frac}(Y))$ representing a fractional heap with codomain Y. From §§7.3 and 7.4 we obtain the following frame-preserving updates as well as the fact that $\textsc{FHeap}(Y)$ is cancellative.

$$h[x \mapsto (1, y)] \rightsquigarrow h[x \mapsto (1, y')]\ \textsc{FHeapUpd} \qquad h \rightsquigarrow \{\, h[x \mapsto (1, y)] \mid x \in \mathit{Val} \,\}\ \textsc{FHeapAlloc}$$

We will write qh with $h : \mathsf{Val} \xrightarrow{\text{fin}} Y$ for the function in $\textsc{FHeap}(Y)$ mapping every $x \in \text{dom}(h)$ to $(q, h(x))$, and everything else to ϵ.

Define $\text{AFHeap}(Y) \triangleq \text{Auth}(\text{FHeap}(Y))$ representing an authoritative fractional heap with codomain Y. We easily obtain the following frame-preserving updates.

$$\textsc{AFHeapUpd}\quad (\bullet\, h[x \mapsto (1,y)], \circ\, [x \mapsto (1,y)]) \rightsquigarrow (\bullet\, h[x \mapsto (1,y')], \circ\, [x \mapsto (1,y')])$$

$$\textsc{AFHeapAdd}\quad \frac{x \notin \text{dom}(h)}{\bullet\, h \rightsquigarrow (\bullet\, h[x \mapsto (q,y)], \circ\, [x \mapsto (q,y)])} \qquad \textsc{AFHeapRemove}\quad (\bullet\, h[x \mapsto (q,y)], \circ\, [x \mapsto (q,y)]) \rightsquigarrow \bullet\, h$$

7.8 STS with tokens monoid

Given a state-transition system (STS) $(\mathcal{S}, \to)$, a set of tokens TokSet, and an assignment $\mathcal{T} : \mathcal{S} \to \mathcal{P}(\mathsf{TokSet})$ of *protocol-owned* tokens to each state, we construct a monoid modeling an authoritative current state and permitting transitions given a *bound* on the current state and a set of *locally-owned* tokens.

We first lift the transition relation to $\mathcal{S} \times \mathcal{P}(\mathsf{TokSet})$ and define upwards closure:

$$\begin{aligned}(s,T) \to (s',T') &\triangleq s \to s' \land \mathcal{T}(s) \uplus T = \mathcal{T}(s') \uplus T' \\ \mathsf{frame}(s,T) &\triangleq (s, \mathsf{TokSet} \setminus (\mathcal{T}(s) \uplus T)) \\ \uparrow(S,T) &\triangleq \{s' \in \mathcal{S} \mid \exists s \in S.\, \mathsf{frame}(s,T) \to^* \mathsf{frame}(s',T)\}\end{aligned}$$

We have

If $(s,T) \to (s',T')$
and $T_f \# (T \uplus \mathcal{T}(s))$,
then $\mathsf{frame}(s,T_f) \to \mathsf{frame}(s',T_f)$.

Proof. This follows directly by framing the tokens in $\mathsf{TokSet} \setminus (T_f \uplus T \uplus \mathcal{T}(s))$ around the given transition, which yields $(s, \mathsf{TokSet} \setminus (T_f \uplus \mathcal{T}(s))) \to (s', T' \uplus (\mathsf{TokSet} \setminus (T_f \uplus T \uplus \mathcal{T}(s))))$. This is exactly what we have to show, since we know $\mathcal{T}(s) \uplus T = \mathcal{T}(s') \uplus T'$. □

Let $\text{STS}_{\mathcal{S}}$ be the monoid with carrier

$$\left\{ (s,S,T) \in \text{Ex}(\mathcal{S}) \times \mathcal{P}(\mathcal{S}) \times \mathcal{P}(\mathsf{TokSet}) \;\middle|\; \begin{array}{l}(s = \epsilon \lor s \in S) \land \uparrow(S,T) = S \land \\ S \neq \emptyset \land \forall s \in S.\, \mathcal{T}(s) \# T\end{array} \right\}$$

and multiplication

$$(s,S,T) \cdot (s',S',T') \triangleq (s'' \triangleq s \cdot_{\text{Ex}(\mathcal{S})} s', S'' \triangleq S \cap S', T'' \triangleq T \cup T') \quad \text{if } (s = \epsilon \lor s' = \epsilon) \land T \# T' \land S'' \neq \emptyset \land (s'' \neq \epsilon \Rightarrow s'' \in S'')$$

Some sugar makes it more convenient to assert being at least in a certain state and owning some tokens: $(s,T) : \text{STS}_{\mathcal{S}} \triangleq (\epsilon, \uparrow(\{s\},T), T) : \text{STS}_{\mathcal{S}}$, and $s : \text{STS}_{\mathcal{S}} \triangleq (s, \emptyset) : \text{STS}_{\mathcal{S}}$.

We will need the following frame-preserving update.

$$\textsc{StsStep}\quad \frac{(s,T) \to^* (s',T')}{(s,S,T) \rightsquigarrow (s', \uparrow(\{s'\},T'), T')}$$

Proof of StsStep. Assume some upwards-closed S_f, T_f (the frame cannot be authoritative) s.t. $s \in S_f$ and $T_f \# (T \uplus \mathcal{T}(s))$. We have to show that this frame combines with our final monoid element, which is the case if $s' \in S_f$ and $T_f \# T'$. By upward-closedness, it suffices to show $\mathsf{frame}(s,T_f) \to^* \mathsf{frame}(s',T_f)$. This follows by induction on the path $(s,T) \to^* (s',T')$, and using the lemma proven above for each step. □

8 Derived constructions

In this section we describe some constructions that we will use throughout the rest of the appendix.

8.1 Global monoid

Hereinafter we assume the global monoid (served up as a parameter to Iris) is obtained from a family of monoids $(M_i)_{i\in I}$ by first applying the construction for finite partial functions to each (§7.4), and then applying the product construction (§7.2):

$$M \triangleq \prod_{i\in I} \textsc{FpFun}(\mathbb{N}, M_i)$$

We write $\boxed{a : M_i}^\gamma$ (or just $\boxed{a}^\gamma$ if M_i is clear from the context) for $\boxed{[i \mapsto [\gamma \mapsto a]]}$ when $a \in |M_i|^+$, and for False when $a = \bot_{M_i}$. In other words, $\boxed{a : M_i}^\gamma$ asserts that in the current state of monoid M_i, the name γ is allocated and has at least value a.

From FpUpd and the multiplications and frame-preserving updates in §7.2 and §7.4, we have the following derived rules.

NewGhost
$$\mathsf{True} \Rrightarrow \exists\gamma.\, \boxed{a : M_i}^\gamma$$

GhostUpd
$$\frac{a \rightsquigarrow_{M_i} B}{\boxed{a : M_i}^\gamma \Rrightarrow \exists b \in B.\, \boxed{b : M_i}^\gamma}$$

GhostEq
$$\boxed{a : M_i}^\gamma * \boxed{b : M_i}^\gamma \Leftrightarrow \boxed{a \cdot b : M_i}^\gamma$$

GhostZero
$$\boxed{\bot : M_i}^\gamma \Rightarrow \mathsf{False}$$

8.2 STSs with interpretation

Building on §7.8, after constructing the monoid $\textsc{Sts}_{\mathcal{S}}$ for a particular STS, we can use an invariant to tie an interpretation, $\varphi : \mathcal{S} \to \mathsf{Prop}$, to the STS's current state, recovering CaReSL-style reasoning [5].

An STS invariant asserts authoritative ownership of an STS's current state and that state's interpretation:

$$\mathsf{STSInv}(\mathcal{S}, \varphi, \gamma) \triangleq \exists s \in \mathcal{S}.\, \boxed{(s, \mathcal{S}, \emptyset) : \textsc{Sts}_{\mathcal{S}}}^\gamma * \varphi(s)$$
$$\mathsf{STS}(\mathcal{S}, \varphi, \gamma, \iota) \triangleq \boxed{\mathsf{STSInv}(\mathcal{S}, \varphi, \gamma)}^\iota$$

We can specialize NewInv, InvOpen, and InvClose to STS invariants:

NewSts
$$\frac{\mathsf{infinite}(\mathcal{E})}{\triangleright\varphi(s) \Rrightarrow_{\mathcal{E}} \exists \iota \in \mathcal{E}, \gamma.\, \mathsf{STS}(\mathcal{S}, \varphi, \gamma, \iota) * \boxed{(s, \mathsf{TokSet} \setminus \mathcal{T}(s)) : \textsc{Sts}_{\mathcal{S}}}^\gamma}$$

StsOpen
$$\mathsf{STS}(\mathcal{S}, \varphi, \gamma, \iota) \vdash \boxed{(s_0, T) : \textsc{Sts}_{\mathcal{S}}}^\gamma \ {}^{\{\iota\}}\!\!\Longleftrightarrow^{\emptyset} \exists s \in \uparrow(\{s_0\}, T).\, \triangleright\varphi(s) * \boxed{(s, \uparrow(\{s_0\}, T), T) : \textsc{Sts}_{\mathcal{S}}}^\gamma$$

StsClose
$$\mathsf{STS}(\mathcal{S}, \varphi, \gamma, \iota), (s, T) \to^* (s', T') \vdash \triangleright\varphi(s') * \boxed{(s, S, T) : \textsc{Sts}_{\mathcal{S}}}^\gamma \ {}^{\emptyset}\!\!\Rrightarrow^{\{\iota\}} \boxed{(s', T') : \textsc{Sts}_{\mathcal{S}}}^\gamma$$

Proof. NewSts uses NewGhost to allocate $\boxed{(s, \uparrow(s, T), T) : \textsc{Sts}_{\mathcal{S}}}^\gamma$ where $T \triangleq \mathsf{TokSet} \setminus \mathcal{T}(s)$, and NewInv.

StsOpen just uses InvOpen and InvClose on ι, and the monoid equality $(s, \uparrow(\{s_0\}, T), T) = (s, \mathcal{S}, \emptyset) \cdot (\epsilon, \uparrow(\{s_0\}, T), T)$.

StsClose applies StsStep and InvClose. □

Using these view shifts, we can prove STS variants of the invariant rules Inv and VSInv (compare the former to CaReSL's island update rule [5]):

$$\textsc{Sts}\quad \frac{\forall s \in \uparrow(\{s_0\}, T).\ \{\triangleright\varphi(s) * P\}\ e\ \{v.\ \exists s', T'.\ (s,T) \to^* (s',T') * \triangleright\varphi(s') * Q\}_{\mathcal{E}} \qquad e \text{ phys. atomic}}{\mathsf{STS}(\mathcal{S},\varphi,\gamma,\iota) \vdash \left\{\boxed{(s_0,T) : \mathrm{STS}_{\mathcal{S}}}^{\gamma} * P\right\}\ e\ \left\{v.\ \exists s', T'.\ \boxed{(s',T') : \mathrm{STS}_{\mathcal{S}}}^{\gamma} * Q\right\}_{\mathcal{E}\uplus\{\iota\}}}$$

$$\textsc{VSSts}\quad \frac{\forall s \in \uparrow(\{s_0\}, T).\ \triangleright\varphi(s) * P \overset{\mathcal{E}_1}{\Rrightarrow}{}^{\mathcal{E}_2}\ \exists s', T'.\ (s,T) \to^* (s',T') * \triangleright\varphi(s') * Q}{\mathsf{STS}(\mathcal{S},\varphi,\gamma,\iota) \vdash \boxed{(s_0,T) : \mathrm{STS}_{\mathcal{S}}}^{\gamma} * P \overset{\mathcal{E}_1\uplus\{\iota\}}{\Rrightarrow}{}^{\mathcal{E}_2\uplus\{\iota\}}\ \exists s', T'.\ \boxed{(s',T') : \mathrm{STS}_{\mathcal{S}}}^{\gamma} * Q}$$

Proof of Sts. We have to show

$$\left\{\boxed{(s_0,T) : \mathrm{STS}_{\mathcal{S}}}^{\gamma} * P\right\}\ e\ \left\{v.\ \exists s', T'.\ \boxed{(s',T') : \mathrm{STS}_{\mathcal{S}}}^{\gamma} * Q\right\}_{\mathcal{E}\uplus\{\iota\}}$$

where v, s', T' are free in Q.

First, by ACsq with StsOpen and StsClose (after moving $(s,T) \to^* (s',T')$ into the view shift using VSBoxOut), it suffices to show

$$\left\{\exists s \in \uparrow(\{s_0\}, T).\ \triangleright\varphi(s) * \boxed{(s, \uparrow(\{s_0\}, T), T)}^{\gamma} * P\right\}$$
$$e$$
$$\left\{v.\ \exists s, T, S, s', T'.\ (s,T) \to^* (s',T') * \triangleright\varphi(s') * \boxed{(s,S,T) : \mathrm{STS}_{\mathcal{S}}}^{\gamma} * Q(v,s',T')\right\}_{\mathcal{E}}$$

Now, use Exist to move the s from the precondition into the context and use Csq to (i) fix the s and T in the postcondition to be the same as in the precondition, and (ii) fix $S \triangleq \uparrow(\{s_0\}, T)$. It remains to show:

$$\left\{s \in \uparrow(\{s_0\}, T) * \triangleright\varphi(s) * \boxed{(s, \uparrow(\{s_0\}, T), T)}^{\gamma} * P\right\}$$
$$e$$
$$\left\{v.\ \exists s', T'.\ (s,T) \to^* (s',T') * \triangleright\varphi(s') * \boxed{(s, \uparrow(\{s_0\}, T), T)}^{\gamma} * Q(v,s',T')\right\}_{\mathcal{E}}$$

Finally, use BoxOut to move $s \in \uparrow(\{s_0\}, T)$ into the context, and Frame on $\boxed{(s, \uparrow(\{s_0\}, T), T)}^{\gamma}$:

$$s \in \uparrow(\{s_0\}, T) \vdash \{\triangleright\varphi(s) * P\}\ e\ \{v.\ \exists s', T'.\ (s,T) \to^* (s',T') * \triangleright\varphi(s') * Q(v,s',T')\}_{\mathcal{E}}$$

This holds by our premise. □

Proof of VSSts. This is similar to above, so we only give the proof in short notation:

Context: $\boxed{\mathsf{STSInv}(\mathcal{S},\varphi,\gamma)}^{\iota}$
$\left\{\boxed{(s_0,T)}^{\gamma} * P\right\}_{\mathcal{E}_1\uplus\{\iota\}}$
$\left\{\exists s.\ \triangleright\varphi(s) * \boxed{(s,S,T)}^{\gamma} * P\right\}_{\mathcal{E}_1}$ by StsOpen
Context: $s \in S \triangleq \uparrow(\{s_0\}, T)$
$\left\{\exists s', T'.\ \triangleright\varphi(s') * Q(s',T') * \boxed{(s,S,T)}^{\gamma}\right\}_{\mathcal{E}_2}$ by premiss
Context: $(s,T) \to^* (s',T')$
$\left\{\boxed{(s',T')}^{\gamma} * Q(s',T')\right\}_{\mathcal{E}_2\uplus\{\iota\}}$ by StsClose

□

8.3 Authoritative monoids with interpretation

Building on §7.6, after constructing the monoid $\textsc{Auth}(M)$ for a cancellative monoid M, we can tie an interpretation, $\varphi : |M|^+ \to \mathsf{Prop}$, to the authoritative element of M, recovering reasoning that is close to the sharing rule in [3].

Let $\varphi_\bot$ be the extension of φ to $|M|$ with $\varphi_\bot(\bot) = \mathsf{False}$. Now define

$$\mathsf{AuthInv}(M, \varphi, \gamma) \triangleq \exists a \in |M|.\ \boxed{\bullet a : \textsc{Auth}(M)}^{\gamma} * \varphi_\bot(a)$$
$$\mathsf{Auth}(M, \varphi, \gamma, \iota) \triangleq M\ \mathsf{cancellative} \wedge \boxed{\mathsf{AuthInv}(M, \varphi, \gamma)}^{\iota}$$

The frame-preserving updates for $\textsc{Auth}(M)$ gives rise to the following view shifts:

NewAuth
$$\frac{\mathsf{infinite}(\mathcal{E}) \qquad M\ \mathsf{cancellative}}{\triangleright\varphi_\bot(a) \Rrightarrow_{\mathcal{E}} \exists \iota \in \mathcal{E}, \gamma.\ \mathsf{Auth}(M, \varphi, \gamma, \iota) * \boxed{\circ a : \textsc{Auth}(M)}^{\gamma}}$$

AuthOpen
$$\mathsf{Auth}(M, \varphi, \gamma, \iota) \vdash \boxed{\circ a : \textsc{Auth}(M)}^{\gamma} \Longrightarrow^{\{\iota\}}_{\emptyset} \exists a_f.\ \triangleright\varphi_\bot(a \cdot a_f) * \boxed{\bullet a \cdot a_f, \circ a : \textsc{Auth}(M)}^{\gamma}$$

AuthClose
$$\mathsf{Auth}(M, \varphi, \gamma, \iota) \vdash \triangleright\varphi_\bot(b \cdot a_f) * \boxed{\bullet a \cdot a_f, \circ a : \textsc{Auth}(M)}^{\gamma} \Rrightarrow^{\emptyset}_{\{\iota\}} \boxed{\circ b : \textsc{Auth}(M)}^{\gamma}$$

These view shifts in turn can be used to prove variants of the invariant rules:

Auth
$$\frac{\forall a_f.\ \{\triangleright\varphi_\bot(a \cdot a_f) * P\}\ e\ \{v.\ \exists b.\ \triangleright\varphi_\bot(b \cdot a_f) * Q\}_{\mathcal{E}} \qquad e\ \text{phys. atomic}}{\mathsf{Auth}(M, \varphi, \gamma, \iota) \vdash \left\{\boxed{\circ a : \textsc{Auth}(M)}^{\gamma} * P\right\}\ e\ \left\{v.\ \exists b.\ \boxed{\circ b : \textsc{Auth}(M)}^{\gamma} * Q\right\}_{\mathcal{E} \uplus \{\iota\}}}$$

VSAuth
$$\frac{\forall a_f.\ \triangleright\varphi_\bot(a \cdot a_f) * P \Rrightarrow^{\mathcal{E}_1}_{\mathcal{E}_2} \exists b.\ \triangleright\varphi_\bot(b \cdot a_f) * Q(b)}{\mathsf{Auth}(M, \varphi, \gamma, \iota) \vdash \boxed{\circ a : \textsc{Auth}(M)}^{\gamma} * P \Rrightarrow^{\mathcal{E}_1 \uplus \{\iota\}}_{\mathcal{E}_2 \uplus \{\iota\}} \exists b.\ \boxed{\circ b : \textsc{Auth}(M)}^{\gamma} * Q(b)}$$

8.4 Ghost heap

We define a simple ghost heap with fractional permissions. Some modules require a few ghost names per module instance to properly manage ghost state, but would like to expose to clients a single logical name (avoiding clutter). In such cases (*e.g.,* §13), we use these ghost heaps.

We seek to implement the following interface:

$$\exists \hookrightarrow : \mathsf{Val} \times \mathbb{Q}_> \times \mathsf{Val} \to \mathsf{Prop}.$$
$$\forall x, q, v.\ x \overset{q}{\hookrightarrow} v \Rightarrow x \overset{q}{\hookrightarrow} v \wedge q \in (0, 1]$$
$$\forall x, q_1, q_2, v, w.\ x \overset{q_1}{\hookrightarrow} v * x \overset{q_2}{\hookrightarrow} w \Leftrightarrow x \overset{q_1+q_2}{\hookrightarrow} v * v = w$$
$$\forall v.\ \mathsf{True} \Rrightarrow_{\emptyset} \exists x.\ x \overset{1}{\hookrightarrow} v$$
$$\forall x, v, w.\ x \overset{1}{\hookrightarrow} v \Rrightarrow_{\emptyset} x \overset{1}{\hookrightarrow} w$$

We write $x \overset{-}{\hookrightarrow} v$ for $\exists q.\ x \overset{q}{\hookrightarrow} v$ and $x \hookrightarrow v$ for $x \overset{1}{\hookrightarrow} v$. Note that $x \overset{-}{\hookrightarrow} v$ is duplicable but cannot be boxed (as it depends on resources); *i.e.,* we have $x \overset{-}{\hookrightarrow} v \Leftrightarrow x \overset{-}{\hookrightarrow} v * x \overset{-}{\hookrightarrow} v$ but not $x \overset{-}{\hookrightarrow} v \Rightarrow \Box x \overset{-}{\hookrightarrow} v$.

To implement this interface, allocate an instance γ_G of $\textsc{FHeap}(\mathit{Val})$ and define

$$x \overset{q}{\hookrightarrow} v \triangleq \begin{cases} \boxed{x \mapsto (q, v)}^{\gamma_G} & \text{if } q \in (0, 1] \\ \mathsf{False} & \text{otherwise} \end{cases}$$

The view shifts in the specification follow immediately from GhostUpd and the frame-preserving updates in §7.7. The first implication is immediate from the definition. The second implication follows by case distinction on $q_1 + q_2 \in (0, 1]$.

9 Logically atomic specifications

9.1 Logically atomic triples

Logically atomic triples are defined as follows:

$$\langle x.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M} \triangleq\ \mathcal{E}_M \# \mathcal{E} \land \Box\left(\begin{array}{l}\forall P, Q, R, \mathcal{E}_R.\\ \quad \langle x.\ P \Longleftrightarrow \alpha \mid R(x), \mathcal{E}_R \mid v.\ \beta \Rrightarrow Q(x,v)\rangle_{\mathcal{E}}^{\mathcal{E}_M}\\ \quad \Rightarrow \{P\}\ e\ \{v.\ \exists x.\ Q(x,v)\}_{\top}\end{array}\right)$$

$$\langle x.\ P \Longleftrightarrow \alpha \mid R, \mathcal{E}_R \mid v.\ \beta \Rrightarrow Q\rangle_{\mathcal{E}}^{\mathcal{E}_M} \triangleq$$
$$\mathsf{timeless}(P) \land \mathcal{E}_R \# \mathcal{E} \cup \mathcal{E}_M \land (P\ {}^{-\mathcal{E}_M}\!\!\Longleftrightarrow^{-\mathcal{E}_M-\mathcal{E}_R}\ \exists x.\ \alpha * R) \land (\forall x, v.\ \beta * R\ {}^{-\mathcal{E}_M-\mathcal{E}_R}\!\!\Rrightarrow^{-\mathcal{E}_M}\ Q)$$

We call the second, auxiliary definition an *atomic shift*. Note that all components of the conjunction are pure, and hence so is the entire atomic shift. When omitted, we take $\mathcal{E}/\mathcal{E}_M$ to be empty and the client's frame R/invariants $\mathcal{E}_R$ to be existentially quantified; for example,

$$\langle x.\ P \Longleftrightarrow \alpha \mid v.\ \beta \Rrightarrow Q\rangle_{\mathcal{E}}^{\mathcal{E}_M}$$

denotes

$$\exists R, \mathcal{E}_R.\ \langle x.\ P \Longleftrightarrow \alpha \mid R, \mathcal{E}_R \mid v.\ \beta \Rrightarrow Q\rangle_{\mathcal{E}}^{\mathcal{E}_M}.$$

To minimize clutter, we often write a list of binders $x_1, \ldots, x_n$ in the precondition when x has product sort.

9.2 Derived rules

The derived rules in Figure 1 enable using logically atomic triples (in a proof using the functions they specify) usually without dealing with atomic shifts. In forthcoming proof outlines, we apply LAHoare to get the ball rolling, shifting proof obligations from Hoare triples to logically atomic triples. LAUnshare complements the treatment of masks in LAFrame, while LASubst provides substitution for the first bound variable in a triple, which also permits changing its sort. For the rest, compare with rules in §6 (AFrame, ACsq, Exist, BoxOut, False, Inv), §8.2 (Sts), and §8.3 (Auth).

As usual, we leave the free variables of propositions implicit in the rules. For the proofs, though, we make them explicit.

Proof of LAIntro. After unfolding the definition of logically atomic triples, this rule is a straightforward instance of the introduction rules for $\Box$, conjunction, universal quantification, and implication. □

Proof of LAAtomic. We use LAIntro. The side-condition on the masks is trivial, as $\mathcal{E}$ is empty. So let P, R, $\mathcal{E}_R$, and Q be given and assume

$$\langle x.\ P \Longleftrightarrow \alpha(x) \mid R, \mathcal{E}_R \mid v.\ \beta(x,v) \Rrightarrow Q(x,v)\rangle_{\emptyset}^{\mathcal{E}_M}. \tag{3}$$

We have to show $\{P\}\ e\ \{v.\ \exists x.\ Q(x,v)\}_{\top}$. From (3) we obtain that $\mathcal{E}_R \# \mathcal{E}_M$ and, by framing $\mathcal{E}_M$,

$$P\ {}^{\top}\!\!\Longleftrightarrow^{-\mathcal{E}_R}\ \exists x.\ \alpha(x) * R(x) \tag{4}$$
$$\forall x, v.\ \beta(x,v) * R(x)\ {}^{-\mathcal{E}_R}\!\!\Rrightarrow^{\top}\ Q(x,v). \tag{5}$$

On the premise, use Frame for $R(x)$ and $-\mathcal{E}_R - \mathcal{E}_M$ to arrive at

$$\forall x.\ \{\alpha(x) * R(x)\}\ e\ \{v.\ \beta(x,v) * R(x)\}_{-\mathcal{E}_R}.$$

Hitting this with Csq and Exist, we obtain

$$\{\exists x.\ \alpha(x) * R(x)\}\ e\ \{v.\ \exists x.\ \beta(x,v) * R(x)\}_{-\mathcal{E}_R}.$$

Since e is physically atomic, we can use ACsq with (4) and (5) to obtain our goal. □

$$\textsc{LAIntro}\quad \frac{P, R, \mathcal{E}_R, Q \mid \langle x.\ P \Longleftrightarrow \alpha \mid R, \mathcal{E}_R \mid v.\ \beta \Rrightarrow Q\rangle_{\mathcal{E}}^{\mathcal{E}_M} \vdash_{\Box} \{P\}\ e\ \{v.\ \exists x.\ Q\}_{\top} \qquad \mathcal{E}_M \# \mathcal{E}}{\langle x.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M}}$$

$$\textsc{LAAtomic}\quad \frac{\forall x.\ \{\alpha(x)\}\ e\ \{v.\ \beta\}_{\mathcal{E}_M} \qquad e\ \text{atomic}}{\langle x.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle^{\mathcal{E}_M}} \qquad \textsc{LAHoare}\quad \frac{\langle x.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M} \qquad \forall x.\ \mathsf{timeless}(\alpha)}{\forall x.\ \{\alpha\}\ e\ \{v.\ \beta\}_{\top}}$$

$$\textsc{LAUnshare}\quad \frac{\langle x.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}\uplus\mathcal{E}'}^{\mathcal{E}_M}}{\langle x.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M\uplus\mathcal{E}'}} \qquad \textsc{LASubst}\quad \frac{\langle x.\ \alpha(x)\rangle\ e\ \langle v.\ \beta(x,v)\rangle_{\mathcal{E}}^{\mathcal{E}_M}}{\langle y.\ \alpha(f(y))\rangle\ e\ \langle v.\ \beta(f(y),v)\rangle_{\mathcal{E}}^{\mathcal{E}_M}} \qquad \textsc{LAFrame}\quad \frac{\langle x.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M} \qquad \mathcal{E}' \# \mathcal{E}_M}{\langle x.\ \alpha * P\rangle\ e\ \langle v.\ \beta * P\rangle_{\mathcal{E}\uplus\mathcal{E}'}^{\mathcal{E}_M}}$$

$$\textsc{LACsq}\quad \frac{\forall x.\ \alpha\ {}^{\mathcal{E}\uplus\mathcal{E}'}\!\Longleftrightarrow^{\mathcal{E}}\ \alpha' \qquad \langle x.\ \alpha'\rangle\ e\ \langle v.\ \beta'\rangle_{\mathcal{E}}^{\mathcal{E}_M} \qquad \forall x, v.\ \beta'\ {}^{\mathcal{E}}\!\Rightarrow^{\mathcal{E}\uplus\mathcal{E}'}\ \beta \qquad \mathcal{E}' \# \mathcal{E}_M}{\langle x.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}\uplus\mathcal{E}'}^{\mathcal{E}_M}}$$

$$\textsc{LAExist}\quad \frac{\langle x, y.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M}}{\langle x.\ \exists y.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M}} \qquad \textsc{LABoxOut}\quad \frac{\Box P \vdash_{\Box} \langle v.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M}}{\langle x.\ \alpha * \Box P\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M}} \qquad \textsc{LAFalse}\quad \langle \mathsf{False}\rangle\ e\ \langle v.\ \beta\rangle^{\mathcal{E}_M}$$

$$\textsc{LAInv}\quad \frac{\langle x.\ \triangleright I * \alpha\rangle\ e\ \langle v.\ \triangleright I * \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M} \qquad \iota \notin \mathcal{E}_M}{\boxed{I}^{\iota} \vdash \langle x.\ \alpha\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}\uplus\{\iota\}}^{\mathcal{E}_M}}$$

$$\textsc{LASts}\quad \frac{\langle x, s.\ s \in \uparrow(\{s_0\}, T) * \triangleright\varphi(s) * \alpha\rangle\ e\ \langle v.\ \exists s', T'.\ (s,T) \to^* (s',T') * \triangleright\varphi(s') * \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M} \qquad \iota \notin \mathcal{E}_M}{\mathsf{STS}(\mathcal{S}, \varphi, \gamma, \iota) \vdash \left\langle x.\ \boxed{(s_0, T) : \textsc{Sts}_{\mathcal{S}}}^{\gamma} * \alpha\right\rangle\ e\ \left\langle v.\ \exists s', T'.\ \boxed{(s', T') : \textsc{Sts}_{\mathcal{S}}}^{\gamma} * \beta\right\rangle_{\mathcal{E}\uplus\{\iota\}}^{\mathcal{E}_M}}$$

$$\textsc{LAAuth}\quad \frac{\langle x, a_f.\ \triangleright\varphi_{\bot}(a \cdot a_f) * \alpha\rangle\ e\ \langle v.\ \exists b.\ \triangleright\varphi_{\bot}(b \cdot a_f) * \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M} \qquad \iota \notin \mathcal{E}_M}{\boxed{\mathsf{AuthInv}(M, \varphi, \gamma)}^{\iota} \vdash \left\langle x.\ \boxed{\circ\, a : \textsc{Auth}(M)}^{\gamma} * \alpha\right\rangle\ e\ \left\langle v.\ \exists b.\ \boxed{\circ\, b : \textsc{Auth}(M)}^{\gamma} * \beta\right\rangle_{\mathcal{E}\uplus\{\iota\}}^{\mathcal{E}_M}}$$

Figure 1: Proof rules for logically atomic triples.

Proof of LAHoare. Let x be given. It suffices to show $\{\alpha(x)\}\ e\ \{v.\ \beta(x,v)\}_\top$. Set $P \triangleq \alpha(x)$ and $R(y) \triangleq y = x$ and $Q(y,v) \triangleq \beta(y,v) \land y = x$ and $\mathcal{E}_R \triangleq \emptyset$. By Csq with $\forall v.\ (\exists y.\ Q(y,v)) \Rightarrow \beta(x,v)$, it suffices to show $\{P\}\ e\ \{v.\ \exists y.\ Q(y,v)\}_\top$. From our first premise, we obtain $\mathcal{E} \# \mathcal{E}_M$ and that it suffices to show

$$\langle y.\ P \Longleftrightarrow \alpha(y) \mid R, \mathcal{E}_R \mid v.\ \beta(y,v) \Rrightarrow Q(y,v)\rangle_{\mathcal{E}}^{\mathcal{E}_M}.$$

By our second premise, we have $\mathsf{timeless}(P)$. As $\mathcal{E}_R$ is empty, we have $\mathcal{E}_R \# \mathcal{E}_M \uplus \mathcal{E}$. The three view shifts follow from $P \Leftrightarrow \exists y.\ \alpha(y) * y = x$ and $\forall y, v.\ \big(\beta(y,v) * y = x \Rightarrow Q(y,v)\big)$. □

Proof of LAUnshare. From our first premise, we obtain $\mathcal{E} \uplus \mathcal{E}' \# \mathcal{E}_M$. It follows that $\mathcal{E} \# \mathcal{E}_M \uplus \mathcal{E}'$, satisfying the mask condition of our goal. Let P, R, $\mathcal{E}_R$, and Q be given and assume

$$\langle x.\ P \Longleftrightarrow \alpha(x) \mid R, \mathcal{E}_R \mid v.\ \beta(x,v) \Rrightarrow Q(x,v)\rangle_{\mathcal{E}}^{\mathcal{E}_M \uplus \mathcal{E}'}. \tag{6}$$

By our premise, it suffices to show

$$\langle x.\ P \Longleftrightarrow \alpha(x) \mid R, \mathcal{E}_R \mid v.\ \beta(x,v) \Rrightarrow Q(x,v)\rangle_{\mathcal{E} \uplus \mathcal{E}'}^{\mathcal{E}_M}. \tag{7}$$

From (6), we obtain that all of $\mathcal{E}$, $\mathcal{E}'$, $\mathcal{E}_M$, and $\mathcal{E}_R$ are pairwise disjoint; P is timeless; and

$$P \ {}^{-\mathcal{E}_M - \mathcal{E}'}\!\Longleftrightarrow^{-\mathcal{E}_M - \mathcal{E}' - \mathcal{E}_R}\ \exists x.\ \alpha(x) * R(x)$$
$$\forall x, v.\ \beta(x,v) * R(x) \ {}^{-\mathcal{E}_M - \mathcal{E}' - \mathcal{E}_R}\!\Rrightarrow^{-\mathcal{E}_M - \mathcal{E}'}\ Q(x,v).$$

To prove (7), it suffices to observe $\mathcal{E}_R \# \mathcal{E} \uplus \mathcal{E}' \uplus \mathcal{E}_M$ and to frame $\mathcal{E}'$ into these view shifts. VSFrame's side-conditions, $-\mathcal{E}_M - \mathcal{E}' \# \mathcal{E}'$ and $-\mathcal{E}_M - \mathcal{E}' - \mathcal{E}_R \# \mathcal{E}'$, are immediate. □

Proof of LASubst. From our first premise, we obtain $\mathcal{E} \uplus \mathcal{E}_M$, satisfying the mask condition of our goal. Let P, R, $\mathcal{E}_R$, and Q be given and assume

$$\langle y.\ P \Longleftrightarrow \alpha(f(y)) \mid R, \mathcal{E}_R \mid v.\ \beta(f(y),v) \Rrightarrow Q(y,v)\rangle_{\mathcal{E}}^{\mathcal{E}_M}. \tag{8}$$

By our premise with $R'(x) \triangleq \exists y.\ x = f(y) * R(y)$ and $Q'(\_, v) \triangleq \exists y.\ Q(y,v)$, it suffices to show

$$\langle x.\ P \Longleftrightarrow \alpha(x) \mid R', \mathcal{E}_R \mid v.\ \beta(x,v) \Rrightarrow Q'(x,v)\rangle_{\mathcal{E}}^{\mathcal{E}_M} \tag{9}$$

to obtain $\{P\}\ e\ \{v.\ \exists x.\ Q'(x,v)\}_\top$. (Using Csq, this entails our goal: $\{P\}\ e\ \{v.\ \exists y.\ Q(y,v)\}_\top$.) From (8) we obtain that $\mathcal{E}_R$, $\mathcal{E}$, and $\mathcal{E}_M$ are pairwise disjoint; P is timeless; and

$$P \ {}^{-\mathcal{E}_M}\!\Longleftrightarrow^{-\mathcal{E}_M - \mathcal{E}_R}\ \exists y.\ \alpha(f(y)) * R(y)$$
$$\forall y, v.\ \beta(f(y),v) * R(y) \ {}^{-\mathcal{E}_M - \mathcal{E}_R}\!\Rrightarrow^{-\mathcal{E}_M}\ Q(y,v).$$

By the definitions of R' and Q', we have:

$$(\exists y.\ \alpha(f(y)) * R(y)) \Leftrightarrow (\exists x.\ \alpha(x) * R'(x))$$
$$\forall v.\ (\exists x.\ \beta(x,v) * R'(x)) \Rightarrow (\exists y.\ \beta(f(y),v) * R(y))$$
$$\forall v.\ (\exists y.\ Q(y,v)) \Rightarrow (\exists x.\ Q'(x,v)).$$

Thus, we may derive the view shifts of (9). □

Proof of LAFrame. From our first premise, we obtain $\mathcal{E} \# \mathcal{E}_M$. Together with our side-condition, this proves the mask condition of our goal. Let P, R, $\mathcal{E}_R$, and Q be given and assume

$$\langle x.\ P \Longleftrightarrow \alpha(x) * \varphi(x) \mid R, \mathcal{E}_R \mid v.\ \beta(x,v) * \varphi(x) \Rrightarrow Q(x,v)\rangle_{\mathcal{E} \uplus \mathcal{E}'}^{\mathcal{E}_M}. \tag{10}$$

By our premise with $R'(x) \triangleq \varphi(x) * R(x)$, it suffices to show

$$\langle x.\ P \Longleftrightarrow \alpha(x) \mid R', \mathcal{E}_R \mid v.\ \beta(x,v) \Rrightarrow Q(x,v)\rangle_{\mathcal{E}}^{\mathcal{E}_M}. \tag{11}$$

From (10) we obtain that all of $\mathcal{E}$, $\mathcal{E}'$, $\mathcal{E}_M$, and $\mathcal{E}_R$ are pairwise disjoint; P is timeless; and

$$P \ {}^{-\mathcal{E}_M}\!\Longleftrightarrow^{-\mathcal{E}_M-\mathcal{E}_R} \exists x.\ \alpha(x) * \varphi(x) * R(x)$$
$$\forall x, v.\ \beta(x,v) * \varphi(x) * R(x) \ {}^{-\mathcal{E}_M-\mathcal{E}_R}\!\Rrightarrow^{-\mathcal{E}_M} \ Q(x,v).$$

Observe that $\mathcal{E}_R \# \mathcal{E} \uplus \mathcal{E}_M$. Thus, by the definition of R', these view shifts prove (11). □

Proof of LACsq. From our second premise, we obtain $\mathcal{E} \# \mathcal{E}_M$. Together with our side-condition, this shows $\mathcal{E} \uplus \mathcal{E}' \# \mathcal{E}_M$, satisfying the mask condition of our goal. Let P, R, $\mathcal{E}_R$, and Q be given and assume

$$\langle x.\ P \Longleftrightarrow \alpha(x) \mid R, \mathcal{E}_R \mid v.\ \beta(x,v) \Rrightarrow Q(x,v) \rangle^{\mathcal{E}_M}_{\mathcal{E} \uplus \mathcal{E}'}. \tag{12}$$

By our second premise with $\mathcal{E}'_R \triangleq \mathcal{E}_R \uplus \mathcal{E}'$, it suffices to show

$$\langle x.\ P \Longleftrightarrow \alpha'(x) \mid R, \mathcal{E}'_R \mid v.\ \beta'(x,v) \Rrightarrow Q(x,v) \rangle^{\mathcal{E}_M}_{\mathcal{E}}. \tag{13}$$

From (12) we obtain that all of $\mathcal{E}$, $\mathcal{E}'$, $\mathcal{E}_R$, and $\mathcal{E}_M$ are pairwise disjoint; P is timeless; and

$$P \ {}^{-\mathcal{E}_M}\!\Longleftrightarrow^{-\mathcal{E}_M-\mathcal{E}_R} \exists x.\ \alpha(x) * R(x) \tag{14}$$
$$\forall x, v.\ \beta(x,v) * R(x) \ {}^{-\mathcal{E}_M-\mathcal{E}_R}\!\Rrightarrow^{-\mathcal{E}_M} \ Q(x,v). \tag{15}$$

We frame our first and third premisses by $-\mathcal{E}_M - \mathcal{E}_R - \mathcal{E} - \mathcal{E}'$ (VSFrame's side-conditions, $-\mathcal{E}_M - \mathcal{E}_R - \mathcal{E} - \mathcal{E}' \# \mathcal{E} \cup \mathcal{E}'$ and $-\mathcal{E}_M - \mathcal{E}_R - \mathcal{E} - \mathcal{E}' \# \mathcal{E}$, are immediate):

$$\forall x.\ \alpha(x) \ {}^{-\mathcal{E}_M-\mathcal{E}_R}\!\Longleftrightarrow^{-\mathcal{E}_M-\mathcal{E}_R-\mathcal{E}'} \ \alpha'(x) \tag{16}$$
$$\forall x, v.\ \beta'(x,v) \ {}^{-\mathcal{E}_M-\mathcal{E}_R-\mathcal{E}'}\!\Rrightarrow^{-\mathcal{E}_M-\mathcal{E}_R} \ \beta(x,v). \tag{17}$$

To prove (13), it suffices to show three view shifts:

- $P \ {}^{-\mathcal{E}_M}\!\Rrightarrow^{-\mathcal{E}_M-\mathcal{E}'_R} \exists x.\ \alpha'(x) * R(x)$.
  $\{P\}_{-\mathcal{E}_M}$
  $\{\exists x.\ \alpha(x) * R(x)\}_{-\mathcal{E}_M-\mathcal{E}_R}$ by (14)
  $\{\exists x.\ \alpha'(x) * R(x)\}_{-\mathcal{E}_M-\mathcal{E}_R-\mathcal{E}'}$ by (16)
- $\exists x.\ \alpha'(x) * R(x) \ {}^{-\mathcal{E}_M-\mathcal{E}'_R}\!\Rrightarrow^{-\mathcal{E}_M} \ P$.
  $\{\exists x.\ \alpha'(x) * R(x)\}_{-\mathcal{E}_M-\mathcal{E}_R-\mathcal{E}'}$
  $\{\exists x.\ \alpha(x) * R(x)\}_{-\mathcal{E}_M-\mathcal{E}_R}$ by (16)
  $\{P\}_{-\mathcal{E}_M}$ by (14)
- $\forall x, v.\ \beta'(x,v) * R(x) \ {}^{-\mathcal{E}_M-\mathcal{E}'_R}\!\Rrightarrow^{-\mathcal{E}_M} \ Q(x,v)$
  $\{\beta'(x,v) * R(x)\}_{-\mathcal{E}_M-\mathcal{E}_R-\mathcal{E}'}$
  $\{\beta(x,v) * R(x)\}_{-\mathcal{E}_M-\mathcal{E}_R}$ by (17)
  $\{Q(x,v)\}_{-\mathcal{E}_M}$ by (15)

These outlines use VSTrans for transitivity. VSTrans's side-condition (the same for each outlined view shift) is immediate: $-\mathcal{E}_M - \mathcal{E}_R \subseteq -\mathcal{E}_M \subseteq (-\mathcal{E}_M) \cup (-\mathcal{E}_M - \mathcal{E}_R - \mathcal{E})$. □

Proof of LAExist. From our first premise, we obtain $\mathcal{E} \# \mathcal{E}_M$, satisfying the mask condition of our goal. Let P, R, $\mathcal{E}_R$, and Q be given and assume

$$\langle x.\ P \Longleftrightarrow \exists y.\ \alpha(x,y) \mid R, \mathcal{E}_R \mid v.\ \beta(x,v) \Rrightarrow Q(x,v) \rangle^{\mathcal{E}_M}_{\mathcal{E}}. \tag{18}$$

By our premise with $R'(x,\_) \triangleq R(x)$ and $Q'(x,\_,v) \triangleq Q(x,v)$, it suffices to show

$$\langle x, y.\ P \Longleftrightarrow \alpha(x,y) \mid R', \mathcal{E}_R \mid v.\ \beta(x,v) \Rrightarrow Q'(x,y,v) \rangle^{\mathcal{E}_M}_{\mathcal{E}} \tag{19}$$

to obtain $\{P\}\ e\ \{v.\ \exists x, y.\ Q'(x, y, v)\}$. This is (by the definition of Q') equivalent to our goal: $\{P\}\ e\ \{v.\ \exists x.\ Q(x, v)\}$. From (18) we obtain that $\mathcal{E}$, $\mathcal{E}_R$, and $\mathcal{E}_M$ are pairwise disjoint; P is timeless; and

$$P\ {}^{-\mathcal{E}_M}\!\Longleftrightarrow^{-\mathcal{E}_M-\mathcal{E}_R}\ \exists x, y.\ \alpha(x, y) * R(x)$$
$$\forall x, v.\ \beta(x, v) * R(x)\ {}^{-\mathcal{E}_M-\mathcal{E}_R}\!\Rrightarrow^{-\mathcal{E}_M}\ Q(x, v).$$

From these and the definition of R', we derive the view shifts needed to prove (19). □

Proof of LABoxOut. First, the top-down direction. From our premise, we obtain $\mathcal{E} \# \mathcal{E}_M$, satisfying the mask condition of our goal. Let P, R, $\mathcal{E}_R$, and Q be given and assume

$$\langle x.\ P \Longleftrightarrow \alpha(x) * \Box P' \mid R, \mathcal{E}_R \mid v.\ \beta(x, v) \Rrightarrow Q(x, v)\rangle^{\mathcal{E}_M}_{\mathcal{E}}. \tag{20}$$

From this, we can obtain

$$P \Longleftrightarrow_{-\mathcal{E}_M} P * \Box P'$$

since boxed assertions are duplicable. Hence it suffices to show

$$\{P * \Box P'\}\ e\ \{v.\ \exists x.\ Q(x, v)\}. \tag{21}$$

By BoxOut, we can move the boxed part of the precondition to the context, so assume $\Box P'$. Applying our premise (which we can do, now that we have strengthened our context), it suffices to show

$$\langle x.\ P \Longleftrightarrow \alpha(x) \mid R, \mathcal{E}_R \mid v.\ \beta(x, v) \Rrightarrow Q(x, v)\rangle^{\mathcal{E}_M}_{\mathcal{E}} \tag{22}$$

which easily follows from (20) and our context.

Now, for bottom-up, assume $\Box P'$. In this context, we have

$$\forall x.\ \alpha(x) \Longleftrightarrow \alpha(x) * \Box P'$$

so the goal follows from the premise via LACsq. □

Proof of LAFalse. We use LAIntro. So we can assume

$$\langle P \Longleftrightarrow \mathsf{False} \mid R, \mathcal{E}_R \mid \beta \Rrightarrow Q\rangle^{\mathcal{E}_M}_{\mathcal{E}} \tag{23}$$

and have to show

$$\{P\}\ e\ \{v.\ \exists x.\ Q(x, v)\}$$

From (23), we easily obtain

$$P \Longleftrightarrow_{-\mathcal{E}_M} \mathsf{False}$$

so by the rule of consequence and False, we are done. □

Proof of LAInv. With $\boxed{I}^{\iota}$ in context, we may derive

$$\forall x.\ \alpha(x)\ {}^{\mathcal{E}\uplus\{\iota\}}\!\Longleftrightarrow^{\mathcal{E}}\ \triangleright I * \alpha(x)$$
$$\forall x, y.\ \triangleright I * \beta(x, y)\ {}^{\mathcal{E}}\!\Rrightarrow^{\mathcal{E}\uplus\{\iota\}}\ \beta(x, y)$$

from InvOpen/InvClose. Apply LACsq to our first premise and these view shifts. LACsq's side-condition follows from $\iota \notin \mathcal{E}_M$. □

Proof of LASts. We have to show

$$\left\langle x.\ \boxed{(s_0, T)}^{\gamma} * \alpha(x)\right\rangle\ e\ \left\langle v.\ \exists s', T'.\ \boxed{(s', T')}^{\gamma} * \beta(x, v, s', T')\right\rangle^{\mathcal{E}_M}_{\mathcal{E}\uplus\{\iota\}}.$$

Set $S_0 \triangleq \uparrow(\{s_0\}, T)$. By LACsq with the following simple consequences of StsOpen and StsClose

$$\forall x.\ \boxed{(s_0, T)}^{\gamma} * \alpha(x) \ {}^{\mathcal{E}\uplus\{\iota\}}\!\Longleftrightarrow^{\mathcal{E}} \exists s \in S_0.\ \triangleright\varphi(s) * \boxed{(s, S_0, T)}^{\gamma} * \alpha(x)$$
$$\forall x, v.\ \left(\exists s, s', T'.\ (s, T) \to^* (s', T') * \triangleright\varphi(s') * \boxed{(s, S_0, T)}^{\gamma} * \beta(x, v, s', T')\right) \ {}^{\mathcal{E}}\!\Rrightarrow^{\mathcal{E}\uplus\{\iota\}}$$
$$\left(\exists s', T'.\ \boxed{(s', T')}^{\gamma} * \beta(x, v, s', T')\right),$$

it suffices to show

$$\left\langle x.\ \exists s \in S_0.\ \triangleright\varphi(s) * \boxed{(s, S_0, T)}^{\gamma} * \alpha(x)\right\rangle$$
$$e$$
$$\left\langle v.\ \exists s, s', T'.\ (s, T) \to^* (s', T') * \triangleright\varphi(s') * \boxed{(s, S_0, T)}^{\gamma} * \beta(x, v, s', T')\right\rangle_{\mathcal{E}}^{\mathcal{E}_M}$$

By LAExist (binding the s in the pre-condition) and LACsq (fixing the s in the post-condition to match that in the pre-condition), it suffices to show

$$\left\langle x, s.\ s \in S_0 * \triangleright\varphi(s) * \boxed{(s, S_0, T)}^{\gamma} * \alpha(x)\right\rangle$$
$$e$$
$$\left\langle v.\ \exists s', T'.\ (s, T) \to^* (s', T') * \triangleright\varphi(s') * \boxed{(s, S_0, T)}^{\gamma} * \beta(x, v, s', T')\right\rangle_{\mathcal{E}}^{\mathcal{E}_M}$$

By LAFrame, it suffices to show our premise, so we are done. □

Proof of LAAuth. We have to show

$$\left\langle x.\ \boxed{a : \textsc{Auth}(M)}^{\gamma} * \alpha(x)\right\rangle\ e\ \left\langle v.\ \exists b.\ \boxed{b : \textsc{Auth}(M)}^{\gamma} * \beta(x, v, b)\right\rangle_{\mathcal{E}\uplus\{\iota\}}^{\mathcal{E}_M}.$$

By LACsq with the following simple consequences of AuthOpen and AuthClose

$$\forall x.\ \boxed{a : \textsc{Auth}(M)}^{\gamma} * \alpha(x) \ {}^{\mathcal{E}\uplus\{\iota\}}\!\Longleftrightarrow^{\mathcal{E}} \exists a_f.\ \triangleright\varphi_\bot(a \cdot a_f) * \boxed{\bullet\, a \cdot a_f, \circ\, a : \textsc{Auth}(M)}^{\gamma} * \alpha(x)$$
$$\forall x, v.\ \left(\exists a_f, b.\ \triangleright\varphi_\bot(b \cdot a_f) * \boxed{\bullet\, a \cdot a_f, \circ\, a : \textsc{Auth}(M)}^{\gamma} * \beta(x, v, b)\right) \ {}^{\mathcal{E}}\!\Rrightarrow^{\mathcal{E}\uplus\{\iota\}}$$
$$\left(\exists b.\ \boxed{b : \textsc{Auth}(M)}^{\gamma} * \beta(x, v, b)\right),$$

it suffices to show

$$\left\langle x.\ \exists a_f.\ \triangleright\varphi_\bot(a \cdot a_f) * \boxed{\bullet\, a \cdot a_f, \circ\, a : \textsc{Auth}(M)}^{\gamma} * \alpha(x)\right\rangle$$
$$e$$
$$\left\langle v.\ \exists a_f, b.\ \triangleright\varphi_\bot(b \cdot a_f) * \boxed{\bullet\, a \cdot a_f, \circ\, a : \textsc{Auth}(M)}^{\gamma} * \beta(x, v, b)\right\rangle_{\mathcal{E}}^{\mathcal{E}_M}$$

By LAExist (binding the a_f in the pre-condition) and LACsq (fixing the a_f in the post-condition to match that in the pre-condition), it suffices to show

$$\left\langle x, a_f.\ \triangleright\varphi_\bot(a \cdot a_f) * \boxed{\bullet\, a \cdot a_f, \circ\, a : \textsc{Auth}(M)}^{\gamma} * \alpha(x)\right\rangle$$
$$e$$
$$\left\langle v.\ \exists b.\ \triangleright\varphi_\bot(b \cdot a_f) * \boxed{\bullet\, a \cdot a_f, \circ\, a : \textsc{Auth}(M)}^{\gamma} * \beta(x, v, b)\right\rangle_{\mathcal{E}}^{\mathcal{E}_M}$$

By LAFrame, it suffices to show our premise, so we are done. □

9.3 Disjunction rule is unsound

The following rule, analogous to Disj, is *not* sound (for simplicity, we restrict preconditions to propositions).

$$\frac{\langle P\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M} \qquad \langle Q\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M}}{\langle P \vee Q\rangle\ e\ \langle v.\ \beta\rangle_{\mathcal{E}}^{\mathcal{E}_M}}$$

TaDA rule	Iris derived rule
Frame	LAFrame
Substitution	LASubst
Atomicity weakening	–
Open region	LASts
Use atomic	LASts
Update region	–
Make atomic	LAIntro

Table 1: Correspondence between TaDA and Iris proof rules for logically atomic triples.

Consider the meaning of the triple $\langle P\rangle\ e\ \langle v.\ \beta\rangle^{\mathcal{E}_M}_{\mathcal{E}}$. It says that the "function" e can be called if the client can provide P at any point in time. In particular, the environment must guarantee that P always holds. The same goes for Q in the second premise. The conclusion, however, allows the environment to switch between P and Q. If e opens the atomic shift several times, it could see P first, and later Q. This is not covered by the premisses: Both assume a stable choice between P and Q.

For the same reason, a version of LAExist that moves the y all the way out to a universal quantifier would be unsound.

9.4 Relation to TaDA rules

Our logically atomic triples closely resembles the one from TaDA [2]. The is most apparent in the similarity of the proof rules. In Table 1, we give a (sometimes rough) correspondence between proof rules of TaDA and Iris. The atomicity context of TaDA corresponds to having an atomic shift in the (normal, propositional) context, while the atomic tracking resource corresponds to either owning P (if the update has not been performed yet) or Q (if an update already happened).

Our triple does not have what's called a *private pre/postcondition* in TaDA. The reason for this is that we did not find a good use for that in our examples. It would be easy to extend our sugar for logically atomic triples to support this. Without private pre/postconditions, of course we don't have a rule corresponding to atomicity weakening. One could see LAHoare as an extreme form of that rule though, where the *entire* pre- and postcondition is moved from the public to the private part (using TaDA terminology).

The "Open region" rule in TaDA allows accessing a region without updating its state. In our case, this is just a special case of LASts.

Since atomic shifts are (unlike the atomic updates of TaDA) not tied to updating the state of some island, LAIntro can be much shorter than the corresponding "Make atomic".

It would be possible to give a rule corresponding to "Update region". That would correspond to applying an atomic shift around a logically atomic triple. Currently, we are instead directly using the view shifts constituting an atomic shift. For a future version of this work, we plan to give a proof rule instead.

10 Warm-up: Locks

In this section, we demonstrate logically atomic triples, borrowing examples from the introductory part of the TaDA paper [2]. We give a logically atomic specification for locks (§10.1) and use it to derive a sequential (CAP-style) specification (§10.2). We give an implementation of locks and prove that it satisfies the logically atomic spec (§10.3). Finally, we summarize the proof outline conventions used in this and subsequent sections (§10.4). (The lock example employs conventions for separate verification discussed in §11.1.)

10.1 Specification

Let the expressions `lock`, `unlock`, and `newLock` and the mask $\mathcal{E}_{\mathsf{lk}}$ be given. Our logically atomic spec for locks relates these to an abstract predicate Lock:

$$
\begin{aligned}
&\exists \mathsf{Lock} : \mathsf{Val} \times \mathsf{Val} \to \mathsf{Prop}.\\
&\quad \forall x, v.\ \mathsf{timeless}(\mathsf{Lock}(x, v)) \land\\
&\quad \forall x, v, w.\ \mathsf{Lock}(x, v) * \mathsf{Lock}(x, w) \Rightarrow \mathsf{False} \land\\
&\quad \{\mathsf{True}\}\ \texttt{newLock}()\ \{x.\ \mathsf{Lock}(x, 0)\} \land\\
&\quad \forall x.\ \langle \mathsf{Lock}(x, 1)\rangle\ \texttt{unlock}(x)\ \langle \mathsf{Lock}(x, 0)\rangle^{\mathcal{E}_{\mathsf{lk}}}\\
&\quad \forall x.\ \langle v.\ \mathsf{Lock}(x, v)\rangle\ \texttt{lock}(x)\ \langle \mathsf{Lock}(x, 1) * v = 0\rangle^{\mathcal{E}_{\mathsf{lk}}}
\end{aligned}
$$

10.2 CAP-style specification

Given an implementation of locks and a mask $\mathcal{E}_{\mathsf{cap}}$ s.t.

$$\mathcal{E}_{\mathsf{lk}} \subseteq \mathcal{E}_{\mathsf{cap}} \qquad\qquad \mathsf{infinite}(\mathcal{E}_{\mathsf{cap}} \setminus \mathcal{E}_{\mathsf{lk}})$$

we can prove the following CAP-style spec (as a weaker spec for the same implementation):

$$
\begin{aligned}
&\exists \mathsf{isCAPLock}, \mathsf{CAPLocked}.\\
&\quad \forall n.\ \mathsf{timeless}(\mathsf{CAPLocked}(n)) \land\\
&\quad \forall n.\ \mathsf{CAPLocked}(n) * \mathsf{CAPLocked}(n) \Rightarrow \mathsf{False} \land\\
&\quad \forall S.\ \mathsf{timeless}(S) \Rightarrow \{S\}\ \texttt{newLock}()\ \{x.\ \exists n.\ \Box\mathsf{isCAPLock}(x, S, n)\} \land\\
&\quad \forall x, S, n.\ \mathsf{isCAPLock}(x, S, n) \Rightarrow \{S * \mathsf{CAPLocked}(n)\}\ \texttt{unlock}(x)\ \{\mathsf{True}\} \land\\
&\quad \forall x, S, n.\ \mathsf{isCAPLock}(x, n) \Rightarrow \{\mathsf{True}\}\ \texttt{lock}(x)\ \{\mathsf{CAPLocked}(n) * S\}
\end{aligned}
$$

The proof uses a monoid $\textsc{Ex}(\{\textsc{Key}\})$ controlling ownership of a key $\textsc{Key}$. Define:

$$
\begin{aligned}
\mathsf{CAPInv}(x, S, \gamma) &\triangleq \mathsf{Lock}(x, 0) * \boxed{\textsc{Key}}^{\gamma} * S \lor \mathsf{Lock}(x, 1)\\
\mathsf{isCAPLock}(x, S, \gamma) &\triangleq \exists \iota \notin \mathcal{E}_{\mathsf{lk}}.\ \mathsf{timeless}(S) \land \boxed{\mathsf{CAPInv}(x, S, \gamma)}^{\iota}\\
\mathsf{CAPLocked}(\gamma) &\triangleq \boxed{\textsc{Key}}^{\gamma}
\end{aligned}
$$

Proof of ***newLock****.* We have $\mathsf{timeless}(S)$ in context. After applying the triple for `newLock`, we create an appropriate monoid and invariant:

$\{S\}_{\top}$

`newLock()`

$\{x.\ S * \mathsf{Lock}(x, 0)\}$

Create new $\textsc{Key}$ monoid γ with state $\textsc{Key}$ ($\textsc{NewGhost}$)

$\left\{x.\ S * \mathsf{Lock}(x, 0) * \exists \gamma.\ \boxed{\textsc{Key}}^{\gamma}\right\}$

$\{x.\ \mathsf{CAPInv}(x, S, \gamma)\}$

Create new invariant $\iota \in \mathcal{E}_{\mathsf{cap}} \setminus \mathcal{E}_{\mathsf{lk}}$ from $\mathsf{CAPInv}(x, S, \gamma)$ ($\textsc{NewInv}$)

$\left\{x.\ \Box\exists \iota.\ \boxed{\mathsf{CAPInv}(x, S, \gamma)}^{\iota}\right\}$

$\{x.\ \Box\mathsf{isCAPLock}(x, S, \gamma)\}$

□

Proof of ***unlock****.* We have $\mathsf{isCAPLock}(x, S, \gamma)$ in context and use the triple for `unlock`:

$\{S * \mathsf{CAPLocked}(\gamma)\}_\top$

LAHoare

$\langle S * \mathsf{CAPLocked}(\gamma)\rangle_{-\mathcal{E}_{\mathsf{lk}}}$

Open invariant ι

$\left\langle S * \boxed{\text{KEY}}^\gamma * \mathsf{CAPInv}(x, S, \gamma)\right\rangle_{-\mathcal{E}_{\mathsf{lk}},\iota}$

$\left\langle S * \boxed{\text{KEY}}^\gamma * \mathsf{Lock}(x, 1)\right\rangle_{-\mathcal{E}_{\mathsf{lk}},\iota}$

$\texttt{unlock}(x)$

$\left\langle S * \boxed{\text{KEY}}^\gamma * \mathsf{Lock}(x, 0)\right\rangle_{-\mathcal{E}_{\mathsf{lk}},\iota}$

$\langle \mathsf{CAPInv}(x, S, \gamma)\rangle_{-\mathcal{E}_{\mathsf{lk}},\iota}$

$\langle \mathsf{True}\rangle_{-\mathcal{E}_{\mathsf{lk}}}$

$\{\mathsf{True}\}_\top$

□

Proof of `lock`. We have $\mathsf{isCAPLock}(x, S, \gamma)$ in context and use the triple for `lock`:

$\{\mathsf{True}\}_\top$

LAHoare

$\langle \mathsf{True}\rangle_{-\mathcal{E}_{\mathsf{lk}}}$

Open invariant ι

$\left\langle S * \boxed{\text{KEY}}^\gamma * \mathsf{Lock}(x, 0) \lor \mathsf{Lock}(x, 1)\right\rangle_{-\mathcal{E}_{\mathsf{lk}},\iota}$

$\left\langle \mathsf{Lock}(x, v) * (v = 0 * S * \boxed{\text{KEY}}^\gamma \lor v = 1)\right\rangle_{-\mathcal{E}_{\mathsf{lk}},\iota}$

$\texttt{lock}(x)$

$\left\langle \mathsf{Lock}(x, 1) * (v = 0 * S * \boxed{\text{KEY}}^\gamma \lor v = 1) * v = 0\right\rangle_{-\mathcal{E}_{\mathsf{lk}},\iota}$

$\left\langle S * \mathsf{Lock}(x, 1) * \boxed{\text{KEY}}^\gamma\right\rangle_{-\mathcal{E}_{\mathsf{lk}},\iota}$

$\langle S * \mathsf{CAPLocked}(\gamma)\rangle_{-\mathcal{E}_{\mathsf{lk}}}$

$\{S * \mathsf{CAPLocked}(\gamma)\}_\top$

□

10.3 Implementation

We aim to prove that the following standard implementation of spinlocks satisfies the logically atomic spec for locks.

$$
\begin{aligned}
\texttt{newLock} &\triangleq \lambda\_.\ \mathsf{ref}\, 0\\
\texttt{unlock} &\triangleq \lambda x.\ x := 0\\
\texttt{lock} &\triangleq \textbf{rec}\ \mathit{loop}(x).\\
&\qquad \textbf{let}\ b = \mathsf{cas}(x, 0, 1)\ \textbf{in}\\
&\qquad \textbf{if}\ b\ \textbf{then}\ ()\ \textbf{else}\ \mathit{loop}(x)
\end{aligned}
$$

We assume that the abstract predicate $- \mapsto -$ and expressions ref, !, :=, cas and mask $\mathcal{E}_{\mathsf{ref}}$ satisfy the following logically atomic specification for references (see §11 for an implementation).

$$
\begin{aligned}
&\forall r, v.\ \mathsf{timeless}(r \mapsto v) \land\\
&\forall r, v, w.\ r \mapsto v * r \mapsto w \Rightarrow \mathsf{False} \land\\
&\forall v.\ \{\mathsf{True}\}\ \mathsf{ref}\ \ v\ \{r.\ r \mapsto v\} \land\\
&\forall r.\ \langle v.\ r \mapsto v\rangle\ !r\ \langle x.\ r \mapsto v \land x = v\rangle^{\mathcal{E}_{\mathsf{ref}}} \land\\
&\forall r, v.\ \langle r \mapsto \_\rangle\ r := v\ \langle x.\ r \mapsto v \land x = ()\rangle^{\mathcal{E}_{\mathsf{ref}}} \land\\
&\forall r, v_1, v_2.\ \left\langle v.\ r \mapsto v\right\rangle\ \mathsf{cas}(r, v_1, v_2)\ \left\langle b.\ \begin{array}{l} b = \mathsf{true} \land v = v_1 \land r \mapsto v_2 \lor\\ b = \mathsf{false} \land v \neq v_1 \land r \mapsto v\end{array}\right\rangle^{\mathcal{E}_{\mathsf{ref}}}
\end{aligned}
$$

Define

$$
\begin{aligned}
\mathcal{E}_{\mathsf{lk}} &\triangleq \mathcal{E}_{\mathsf{ref}}\\
\mathsf{Lock}(x, v) &\triangleq x \mapsto v
\end{aligned}
$$

Note that, in contrast to TaDA, there is no need to introduce an invariant for this lock. The timelessness and separation properties follow immediately from those for references.

Proof of ***newLock****.* As expected, the proof is rather short:

$\{\mathsf{True}\}_\top$

ref 0

$\{x.\ x \mapsto 0\}_\top$

$\{x.\ \mathsf{Lock}(x, 0)\}_\top$

□

Proof of ***unlock****.* After LAIntro, it suffices to show $\{P\}\ x := 0\ \{Q\}_\top$ with

$$\langle P \Longleftrightarrow x \mapsto 1 \mid R, \mathcal{E}_R \mid x \mapsto 0 \Rrightarrow Q \rangle^{\mathcal{E}_{\mathsf{ref}}}$$

in context, for some P, Q, R, and $\mathcal{E}_R$. We use these client-supplied view shifts around the logically atomic triple for assignment:

$\{P\}_\top$

LAHoare
- $\langle P \rangle_{-\mathcal{E}_{\mathsf{ref}}}$
- Open $\mathcal{E}_R$
  - Check masks: $\mathcal{E}_{\mathsf{ref}} \subseteq \mathcal{E}_{\mathsf{ref}}$
  - $\langle x \mapsto 1 * R \rangle_{-\mathcal{E}_{\mathsf{ref}}, \mathcal{E}_R}$
  - $x := 0$
  - $\langle x \mapsto 0 * R \rangle_{-\mathcal{E}_{\mathsf{ref}}, \mathcal{E}_R}$
- $\langle Q \rangle_{-\mathcal{E}_{\mathsf{ref}}}$

$\{Q\}_\top$

□

Proof of ***lock****.* After LAIntro, it suffices to show $\{P\}\ \texttt{unlock}(x)\ \{Q\}$ assuming

$$\langle v.\ P \Longleftrightarrow x \mapsto v \mid R(v), \mathcal{E}_R \mid x \mapsto 1 * v = 0 \Rrightarrow Q \rangle^{\mathcal{E}_{\mathsf{ref}}}$$

Context: $\forall y.\ \{\triangleright P\}\ \mathit{loop}(y)\ \{Q\}$

$\{P\}_\top$

LAHoare
- $\langle P \rangle_{-\mathcal{E}_{\mathsf{ref}}}$
- Open $\mathcal{E}_R$
  - Check masks: $\mathcal{E}_{\mathsf{ref}} \subseteq \mathcal{E}_{\mathsf{ref}}$
  - $\langle x \mapsto v * R(v) \rangle_\emptyset$
  - **let** $b = \mathsf{cas}(x, 0, 1)$ **in**
  - $\langle ((b = \mathsf{true} \land v = 0 \land x \mapsto 1) \lor (b = \mathsf{false} \land v \neq 0 \land x \mapsto v)) * R(v) \rangle_\emptyset$
  - $\langle (b = \mathsf{true} * v = 0 * x \mapsto 1 * R(v)) \lor (b = \mathsf{false} * x \mapsto v * R(v)) \rangle_\emptyset$
- $\langle b = \mathsf{true} * Q \lor b = \mathsf{false} * P \rangle_{-\mathcal{E}_{\mathsf{ref}}}$

$\{b = \mathsf{true} * Q \lor b = \mathsf{false} * P\}_\top$

if b **then** $\{Q\}_\top\ ()\ \{Q\}_\top$ **else** $\{P\}_\top\ \mathit{loop}(x)\ \{Q\}_\top$

$\{Q\}_\top$

□

10.4 Proof outline conventions

For normal Hoare-style proofs (with proof steps in curly braces), we implicitly use framing and the rule of consequence. Explicitly applied rules are denoted with a vertical bar, where the premiss is shown on the right, and the name is given (sideways) left of the bar. The mask, usually annotated at each proof step, can be omitted if it did not change from the previous step on the same proof (indentation) level, or from the previous step on the next outer level if this is the first step in an

indentation. Masks are given as $-\mathcal{E}_1, \mathcal{E}_2, \ldots$, which is short-hand for $\top \setminus (\mathcal{E}_1 \uplus \mathcal{E}_2 \uplus \ldots)$. In general, we use $-$ for mask difference, and we write ι for $\{\iota\}$.

When we write *Context:* P for some pure assertion, that's an application of Exist to move all free variables of P to the context, and an application of BoxOut to make P generally available.

When we prove a logically atomic triple, we implicitly use LAIntro to get the proof going.

Proof steps that are enclosed in angle brackets correspond to working with logically atomic triples. The mask annotated here is the shared mask of the triple. There will always be an outer application of LAHoare. The first proof step within this application needs to be checked for timelessness, as per the second premise of LAHoare. Immediately after this step, the mask annotation is $-\mathcal{E}_M$, where $\mathcal{E}_M$ is the module mask taken from the logically atomic triple we are ultimately going to use. We never use LAUnshare, so the module mask does not change. Therefore, we omit it.

Variables introduced by the binder in the precondition of atomic triples are used as if they were in the context (which, for the purpose of the pre- and postcondition, they are). When we write *Context:* P for some pure assertion, the free variables are moved to the aforementioned binder using LAExist. The proposition P itself is not actually moved anywhere (it cannot be moved to the global context, as that does not have the free variables bound). It is elided from the future steps in this sub-tree of the proof, and implicitly made use of when necessary. At the "center" of the proof (where the function is actually called), the remaining resources and islands are framed around the specification triple of the function. This may leave some unused variables bound in the triple, which are removed using LASubst (with f being a projection function).

When opening an island ι with $-\mathcal{E}$ being the current mask, the fact that we will write $-\mathcal{E}, \iota$ in the next step means we have to check that ι is disjoint from $\mathcal{E}$. This suffices for both mask-related side-conditions of LAInv: The $\mathcal{E}_S$ of that rule becomes $-\mathcal{E}, \iota$, so $\iota \notin \mathcal{E}_S$ is trivial. Since $\mathcal{E}$ will contain $\mathcal{E}_M$, we also get $\iota \notin \mathcal{E}_M$.

Similar so for the case of opening an atomic shift:

$$\langle x.\ P \Longleftrightarrow \alpha \mid R, \mathcal{E}_R \mid v.\ \beta \Rrightarrow Q \rangle_{\mathcal{E}}^{\mathcal{E}_M}$$

We are applying Csq or LACsq to make use of the view shifts. In the proof, with $-\mathcal{E}$ being the current mask and $-\mathcal{E}, \mathcal{E}_R$ the new one, we will have to check that we can actually apply the view shifts. This amounts to proving $-\mathcal{E}_M \subseteq -\mathcal{E}$. To make sure this is satisfied, we explicitly check

$$\mathcal{E} \subseteq \mathcal{E}_M$$

each time we open an atomic shift. That makes sense: the already open (disabled) islands must be in the module mask, specifying which islands the client proving the atomic shift had to entirely avoid.

11 References as channels

In Figures 2–4 we give logically atomic specifications for references and channels and an encoding of references in terms of channels. We prove

$$\begin{aligned}
&\forall \mathcal{E}_{\mathsf{chan}}, \mathcal{E}_{\mathsf{ref}}; e_{\mathsf{newch}}, e_{\mathsf{send}}, e_{\mathsf{recv}}. \\
&\quad \varphi_{\mathsf{chan}}(\mathcal{E}_{\mathsf{chan}}, e_{\mathsf{newch}}, e_{\mathsf{send}}, e_{\mathsf{recv}}) \wedge \\
&\quad \mathcal{E}_{\mathsf{chan}} \subseteq \mathcal{E}_{\mathsf{ref}} \wedge \\
&\quad \mathsf{infinite}(\mathcal{E}_{\mathsf{ref}} \setminus \mathcal{E}_{\mathsf{chan}}) \Rrightarrow_{\mathcal{E}_{\mathsf{ref}}} \\
&\qquad \varphi_{\mathsf{ref}}(\mathcal{E}_{\mathsf{ref}}, \mathsf{ref}, !, :=, \mathsf{cas})
\end{aligned} \tag{24}$$

in §11.3, after briefly discussing such lemmas (§11.1) and defining the reference invariant (§11.2). The invariant organizes the ghost state used in our proof. (We offer a rough guide to this organization in §11.2.)

$$
\begin{array}{l}
\varphi_{\mathsf{ref}}(\mathcal{E}_{\mathsf{ref}}, e_{\mathsf{ref}}, e_{\mathsf{get}}, e_{\mathsf{set}}, e_{\mathsf{cas}}) \triangleq \Box\exists{\mapsto} \in \mathsf{Val} \times \mathsf{Val} \to \mathsf{Prop}. \\
\quad \forall r, v.\ \mathsf{timeless}(r \mapsto v) \wedge \\
\quad \forall r, v, w.\ r \mapsto v * r \mapsto w \Rightarrow \mathsf{False} \wedge \\
\quad \forall v.\ \{\mathsf{True}\}\ e_{\mathsf{ref}}\ v\ \{r.\ r \mapsto v\} \wedge \\
\quad \forall r.\ \langle v.\ r \mapsto v \rangle\ e_{\mathsf{get}}\ r\ \langle x.\ r \mapsto v \wedge x = v \rangle^{\mathcal{E}_{\mathsf{ref}}} \wedge \\
\quad \forall r, v.\ \langle r \mapsto \_ \rangle\ e_{\mathsf{set}}\ r\ v\ \langle x.\ r \mapsto v \wedge x = () \rangle^{\mathcal{E}_{\mathsf{ref}}} \wedge \\
\quad \forall r, v_1, v_2.\ \left\langle v.\ r \mapsto v \right\rangle\ e_{\mathsf{cas}}(r, v_1, v_2)\ \left\langle b.\ \begin{array}{l} b = \mathsf{true} \wedge v = v_1 \wedge r \mapsto v_2 \vee \\ b = \mathsf{false} \wedge v \neq v_1 \wedge r \mapsto v \end{array} \right\rangle^{\mathcal{E}_{\mathsf{ref}}}
\end{array}
$$

Figure 2: Reference specification.

$$
\begin{array}{l}
\varphi_{\mathsf{chan}}(\mathcal{E}_{\mathsf{chan}}, e_{\mathsf{newch}}, e_{\mathsf{send}}, e_{\mathsf{recv}}) \triangleq \Box\exists{\prec} \in \mathsf{Val} \times \mathsf{Bag} \to \mathsf{Prop}. \\
\quad \forall c, M.\ \mathsf{timeless}(c \prec M) \wedge \\
\quad \forall c, M, M'.\ c \prec M * c \prec M' \Rightarrow \mathsf{False} \wedge \\
\quad \{\mathsf{True}\}\ e_{\mathsf{newch}}\ \{c.\ c \prec \emptyset\} \wedge \\
\quad \forall c, m.\ \langle M.\ c \prec M \rangle\ e_{\mathsf{send}}(c, m)\ \langle x.\ c \prec M \uplus \{m\} \wedge x = () \rangle^{\mathcal{E}_{\mathsf{chan}}} \wedge \\
\quad \forall c.\ \langle M.\ c \prec M \rangle\ e_{\mathsf{recv}}\ c\ \langle m.\ c \prec M \setminus \{m\} \wedge m \in M \rangle^{\mathcal{E}_{\mathsf{chan}}}
\end{array}
$$

Figure 3: Channel specification. (m has sort Val.)

11.1 Pattern for separate verification

While we will generally *not* be so explicit in subsequent sections of this appendix, the pattern we used in defining φ_{chan} and φ_{ref} and in stating (24) scales to verifying a *stack* of abstractions, $(\varphi_i)_{i \in 1..n}$. The "mask annotation burden" does not grow and all but the first lemma has the form

> If $\varphi_i(\mathcal{E}_i)$,
> and $\mathcal{E}_i \subseteq \mathcal{E}$,
> and $\mathsf{infinite}(\mathcal{E} \setminus \mathcal{E}_i)$,
> then $\varphi_{i+1}(\mathcal{E})$.

This can be simplified for modules that do not need to allocate invariants. It can be made more complicated if a module directly depends on multiple other modules (*i.e.,* if the dependency graph does not form a stack). Consider, for example, a module φ depending on modules $(\varphi_j)_{j \in J}$. The lemma for proving $\varphi(\mathcal{E})$ would assume $\varphi_j(\mathcal{E}_j)$ for each j and $\mathcal{E} \supseteq \bigcup_j \mathcal{E}_j$ and $\mathsf{infinite}(\mathcal{E} \setminus \bigcup_j \mathcal{E}_j)$. We would also assume $\mathcal{E}_j \# \mathcal{E}_{j'}$ if, say, we needed to apply view shifts from module j using $\mathcal{E}_j$ around a logically atomic triple from module j'.

Once proved, we can apply (24) to *any* implementation of the channel specification. Later, we will verify code against the reference specification. Those verifications work no matter how we choose to implement φ_{ref}. Ultimately, we will close things up by defining a language with channels, proving φ_{chan}, then applying (24). We could take a more torturous path, starting with a language that satisfies the reference specification by construction, implementing channels using references, proving a lemma anlogous to (24) for that encoding, and finally applying (24).

Let expressions e_{newch}, e_{send}, and e_{recv} be given. Define

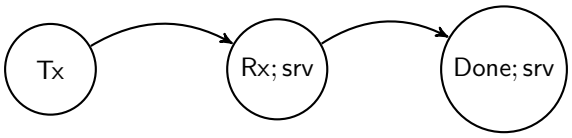
$$
\begin{aligned}
\mathsf{ref}\ e &\triangleq \textbf{let}\ r = e_{\mathsf{newch}}\ \textbf{in fork}\ \mathsf{srv}\ r\ e; r \\
!e &\triangleq \mathsf{rpc}\ e\ \mathsf{Get} \\
e := e' &\triangleq \mathsf{rpc}\ e\ \mathsf{Set}(e') \\
\mathsf{cas}(e, e_1, e_2) &\triangleq \mathsf{rpc}\ e\ \mathsf{Cas}(e_1, e_2)
\end{aligned}
$$

where

$$
\begin{aligned}
\mathsf{rpc} \triangleq\ & \lambda r.\ \lambda m. \\
& \textbf{let}\ d = e_{\mathsf{newch}}\ \textbf{in} \\
& e_{\mathsf{send}}(r, (d, m)); e_{\mathsf{recv}}\ d \\
\mathsf{srv} \triangleq\ & \lambda r.\ \textbf{rec}\ \mathit{loop}(v). \\
& \textbf{let}\ (d, m) = e_{\mathsf{recv}}\ r\ \textbf{in} \\
& \textbf{let}\ \mathit{reply} = \lambda m'.\ \lambda v'.\ (e_{\mathsf{send}}(d, m'); \mathit{loop}\ v')\ \textbf{in} \\
& \textbf{case}\ m\ \textbf{of} \\
& \quad \mathsf{Get} \Rightarrow \mathit{reply}\ v\ v \\
& \mid \mathsf{Set}(w) \Rightarrow \mathit{reply}\ ()\ w \\
& \mid \mathsf{Cas}(v_1, v_2) \Rightarrow \\
& \quad \textbf{let}\ b = (v = v_1)\ \textbf{in} \\
& \quad \textbf{let}\ v' = \textbf{if}\ b\ \textbf{then}\ v_2\ \textbf{else}\ v\ \textbf{in} \\
& \quad \mathit{reply}\ b\ v'.
\end{aligned}
$$

Figure 4: Encoding references. (We assume a language with channels, recursive functions, sums, and products. See §12 for details.)

Let $\mathcal{S}_{\mathsf{RPC}}$ denote the STS



with three states and one token, srv. Let ι_{ref}, γ_{ref}, $\mathcal{E}_{\mathsf{ref}}$, and $\mathcal{E}_{\mathsf{rpc}}$ be given. Define

$$
\begin{aligned}
r \xmapsto[\gamma_{\mathsf{ref}}]{\mathsf{srv}} v &\triangleq \boxed{\circ\, 1/3[r \mapsto v] : \mathrm{AFHEAP}(Val)}^{\gamma_{\mathsf{ref}}} \\
r \xmapsto[\gamma_{\mathsf{ref}}]{\mathsf{cli}} v &\triangleq \boxed{\circ\, 2/3[r \mapsto v]}^{\gamma_{\mathsf{ref}}} \\
\mathsf{RefInv} &\triangleq \exists h.\ \boxed{\bullet\, 1h}^{\gamma_{\mathsf{ref}}} * \mathsf{isRefs}(\mathrm{dom}(h)) \\
\mathsf{isRefs}(S) &\triangleq \mathop{\Large *}_{r \in S} \exists M.\ r \prec M * \mathsf{isRef}(r, M) \\
\mathsf{isRef}(r, M) &\triangleq \mathop{\Large *}_{tx \in M} \exists d, m.\ tx = (d, m) \wedge \mathsf{Waiting}(r, d, m) \\
\mathsf{Waiting}(r, d, m) &\triangleq \exists \iota \in \mathcal{E}_{\mathsf{rpc}}; \gamma; P; Q.\ \boxed{\mathsf{STSInv}(\mathcal{S}_{\mathsf{RPC}}, \mathsf{Rpc}(d, P, Q), \gamma)}^{\iota} \wedge \\
&\qquad \boxed{(\mathsf{Tx}, \{\mathsf{srv}\}) : \mathrm{STS}_{\mathcal{S}_{\mathsf{RPC}}}}^{\gamma} \wedge \\
&\qquad \mathsf{isReq}(P, Q, r, m) \\
\mathsf{Rpc}(d, P, Q)(s) &\triangleq \exists M.\ d \prec M * \mathsf{RpcAux}(P, Q, s, M) \\
\mathsf{RpcAux}(P, Q, s, M) &\triangleq s = \mathsf{Tx} \wedge M = \emptyset * P \vee \\
&\qquad s = \mathsf{Rx} \wedge \exists v, m.\ M = \{m\} * Q(v, m) \vee \\
&\qquad s = \mathsf{Done} \wedge M = \emptyset \\
\mathsf{isReq}(P, Q, r, m) &\triangleq m = \mathsf{Get} \wedge \langle v.\ P \Longleftrightarrow r \mapsto v \mid x.\ r \mapsto v \wedge x = v \Rrightarrow Q(v, x) \rangle^{\mathcal{E}_{\mathsf{ref}}} \vee \\
&\qquad \exists v.\ m = \mathsf{Set}(v) \wedge \langle w.\ P \Longleftrightarrow r \mapsto w \mid x.\ r \mapsto v \wedge x = () \Rrightarrow Q(w, x) \rangle^{\mathcal{E}_{\mathsf{ref}}} \vee \\
&\qquad \exists v_1, v_2.\ m = \mathsf{Cas}(v_1, v_2) \wedge \left\langle \begin{array}{l} v.\ P \Longleftrightarrow r \mapsto v \mid \\ b.\ b = \mathsf{true} \wedge v = v_1 \wedge r \mapsto v_2 \vee \\ b = \mathsf{false} \wedge v \neq v_1 \wedge r \mapsto v \Rrightarrow Q(v, b) \end{array} \right\rangle^{\mathcal{E}_{\mathsf{ref}}}
\end{aligned}
$$

Figure 5: Reference invariant with an auxiliary STS and server- and client-side ghost assertions. To reduce clutter, we generally leave parameters implicit; for example, the fully explicit version of $\mathsf{isRefs}(S)$ is $\mathsf{isRefs}(\mathcal{E}_{\mathsf{rpc}}, \mathcal{E}_{\mathsf{ref}})(S)$.

11.2 Reference invariant

In Figure 5, we define an RPC STS ($\mathcal{S}_{\mathsf{RPC}}$), server- and client-side ghost assertions ($\overset{\mathsf{srv}}{\longmapsto}$, $\overset{\mathsf{cli}}{\longmapsto}$), and the reference invariant (RefInv).

RefInv owns the authoritative copy of a ghost heap, h, mapping channel names that represent references to fractions of the values they contain. The server and clients of reference r own fractions of r's fragment, $[r \mapsto v]$, of this ghost heap. These fractions are written $r \overset{\mathsf{srv}}{\longmapsto} v$ and $r \overset{\mathsf{cli}}{\longmapsto} v$. (The split isn't 50/50. The client must own more than half so that we can prove $r \overset{\mathsf{cli}}{\longmapsto} v * r \overset{\mathsf{cli}}{\longmapsto} w \Rightarrow \mathsf{False}$, as required by φ_{ref}.) This fragment is kept in sync with the server's current state; see the statement of lemma Srv. To help a client when responding to an RPC, the server obtains the client's fraction from isReq, a request-specific atomic shift; for example,

$$\mathsf{isReq}(P, Q, r, \mathsf{Get}) \Leftrightarrow \langle v.\ P \Longleftrightarrow r \mapsto v \mid x.\ r \mapsto v \wedge x = v \Rrightarrow Q(v, x) \rangle^{\mathcal{E}_{\mathsf{ref}}}$$

is precisely the atomic shift assumed in proving (via rule LAIntro) the logically atomic specification for dereference in φ_{ref}. When it's ready to update its physical state and these ghosts, the server combines its fragment (kept locally), its current client's fragment (from isReq), and the authoritative copy (from the reference invariant) in order to apply AFHeapUpd.

Alongside its ghost heap h, the invariant asserts $\mathsf{isRefs}(\mathrm{dom}(h))$. For each $r \in \mathrm{dom}(h)$, the invariant owns a channel assertion $r \prec M$, where M contains pending RPC requests for reference r. For each pending request $tx \in M$, the invariant asserts that tx has the form (d, m) for some destination d and payload m satisfying $\mathsf{Waiting}(r, d, m)$. On receiving tx, the server for r plucks $\mathsf{Waiting}(r, d, m)$ out of the reference invariant. (Thus the reference invariant tracks RPC's that have been sent but not yet received.)

The assertion $\mathsf{Waiting}(r, d, m)$ contains everything the server for r needs to help the client that's waiting for a reply to the request (d, m): Knowledge of an RPC STS (q.v.), ownership of a corresponding ghost assertion ("we're at least in state Tx and we own the server token"), and the atomic shift isReq.

The RPC STS, $\mathcal{S}_{\mathsf{RPC}}$, comprises three states and one token. State Tx represents a request that hasn't been received by the server, Rx a response that hasn't been received by the client, and Done a finished RPC. We use the server token, srv, to model the fact that only the server for a particular reference, r, responds to requests on r. This works because tokens are conserved: A thread—knowing that an instance of the STS is at least in state Tx and owning the server token—can conclude that the STS is precisely in Tx. We need no analogous *client* token. The STS interpretation, $\mathsf{Rpc}(d, P, Q)$, owns a channel resource, $d \prec M$, and ensures that M is empty except in state Rx. A client—on receiving a message from d—can conclude the STS was in state Rx.

The point of all this bookkeeping is to permit a reference cell's client to pass isReq and a freshly-allocated RPC STS to the proper server, and then to wait for the server's response through that STS.

11.3 Reference verification

Creating the ref invariant Observe that $\mathsf{isRefs}(\emptyset)$ is vacuous. Using this observation, we can initialize our ref invariant. Let masks $\mathcal{E}_{\mathsf{chan}}$, $\mathcal{E}_{\mathsf{ref}}$ and expressions e_{newch}, e_{send}, e_{recv} be given and assume

$$\varphi_{\mathsf{chan}}(\mathcal{E}_{\mathsf{chan}}, e_{\mathsf{newch}}, e_{\mathsf{send}}, e_{\mathsf{recv}}), \qquad \mathcal{E}_{\mathsf{chan}} \subseteq \mathcal{E}_{\mathsf{ref}}, \qquad \mathsf{infinite}(\mathcal{E}_{\mathsf{ref}} \setminus \mathcal{E}_{\mathsf{chan}}).$$

Split $\mathcal{E}_{\mathsf{ref}} \setminus \mathcal{E}_{\mathsf{chan}}$ into $\mathcal{E}$ and $\mathcal{E}_{\mathsf{rpc}}$ such that both are infinite and $\mathcal{E} \# \mathcal{E}_{\mathsf{rpc}}$. Create a new, empty ghost heap $\boxed{\bullet 1\emptyset : \mathrm{AFHeap}(\mathit{Val})}^{\gamma_{\mathsf{ref}}}$. Build an invariant $\boxed{\mathsf{RefInv}(\gamma_{\mathsf{ref}}, \mathcal{E}_{\mathsf{rpc}}, \mathcal{E}_{\mathsf{ref}})}^{\iota_{\mathsf{ref}}}$ around this empty ghost heap, drawing ι_{ref} from $\mathcal{E}$. Instantiate the existential in φ_{ref} by $\mapsto \triangleq \overset{\mathsf{cli}}{\underset{\gamma_{\mathsf{ref}}}{\longmapsto}}$. It suffices to prove the

conjuncts in $\varphi_{\mathsf{ref}}(\mathcal{E}_{\mathsf{ref}}, \mathsf{ref}, !, :=, \mathsf{cas})$ assuming

$$\begin{aligned}\Theta_{\mathsf{ref}} \triangleq\ & \forall c, M.\ \mathsf{timeless}(c \prec M);\\ & \forall c, M, M'.\ c \prec M * c \prec M' \Rightarrow \mathsf{False};\\ & \{\mathsf{True}\}\ e_{\mathsf{newch}}\ \{c.\ c \prec \emptyset\};\\ & \forall c, m.\ \langle M.\ c \prec M\rangle\ e_{\mathsf{send}}(c, m)\ \langle x.\ c \prec M \uplus \{m\} \wedge x = ()\rangle^{\mathcal{E}_{\mathsf{chan}}};\\ & \forall c.\ \langle M.\ c \prec M\rangle\ e_{\mathsf{recv}}\ c\ \langle m.\ c \prec M \setminus \{m\} \wedge m \in M\rangle^{\mathcal{E}_{\mathsf{chan}}};\\ & \mathcal{E}_{\mathsf{chan}} \subseteq \mathcal{E}_{\mathsf{ref}};\\ & \boxed{\mathsf{RefInv}(\gamma_{\mathsf{ref}}, \mathcal{E}_{\mathsf{rpc}}, \mathcal{E}_{\mathsf{ref}})}^{\iota_{\mathsf{ref}}};\\ & \iota_{\mathsf{ref}} \in (\mathcal{E}_{\mathsf{ref}} \setminus \mathcal{E}_{\mathsf{chan}}) \setminus \mathcal{E}_{\mathsf{rpc}};\\ & \mathcal{E}_{\mathsf{rpc}} \subseteq \mathcal{E}_{\mathsf{ref}} \setminus \mathcal{E}_{\mathsf{chan}}; \mathsf{infinite}(\mathcal{E}_{\mathsf{rpc}}).\end{aligned}$$

In the following we tacitly assume every context extends Θ_{ref}.

Simple properties As ghosts are timeless, we have $\mathsf{timeless}(r \overset{\mathsf{cli}}{\longmapsto} v)$. The separation property is also immedate: By combining the given ghosts we obtain a fraction 4/3, which is absurd.

11.3.1 Client-side proofs

The specifications for assignment, dereference, and CAS follow by LAIntro and a lemma for rpc:

$$\begin{array}{l}\textsc{Rpc}\\ \mathsf{isReq}(P, Q, r, m) \vdash \{P\}\ \mathsf{rpc}\ r\ m\ \{m'.\ \exists v.\ Q(v, m')\}_{\top}.\end{array}$$

We prove Rpc in Figure 6.

11.3.2 Server-side proofs

Verifying allocation involves verifying the server code. To that end, we define isReply in Figure 7. The idea is that $\mathsf{isReply}(v_0, m, m', v')$ relates the server's inputs to its outputs: If the server is running with reference contents v_0 and handles request m, then it will respond to the request with message m' and, on its next iteration, run with reference contents v'.

Our proof of allocation relies on a lemma for the server loop which, in turn, relies on a lemma for the server's call to e_{send}.

$$\begin{array}{l}\textsc{Srv}\\ \{r \overset{\mathsf{srv}}{\longmapsto} v_0\}\ \mathsf{srv}\ r\ v_0\ \{\mathsf{False}\}\end{array} \qquad \begin{array}{l}\textsc{SrvSend}\\ \mathsf{isReply}(v_0, m, m', v') \vdash\\ \quad \{r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{Waiting}(r, d, m)\}\ e_{\mathsf{send}}(d, m')\ \{x.\ x = () \wedge r \overset{\mathsf{srv}}{\longmapsto} v'\}\end{array}$$

We prove these top-down, in Figures 9, 10, and 11. In SrvSend, a server actually helps its client. For assignment and CAS requests, the server must update ghosts. We factor this out as a simple lemma

$$\begin{array}{l}\textsc{SrvUpd}\\ r \overset{\mathsf{srv}}{\longmapsto} v_0 * r \overset{\mathsf{cli}}{\longmapsto} v_0 \Rrightarrow_{\{\iota_{\mathsf{ref}}\}} r \overset{\mathsf{srv}}{\longmapsto} v' * r \overset{\mathsf{cli}}{\longmapsto} v'\end{array}$$

that we prove in Figure 8.

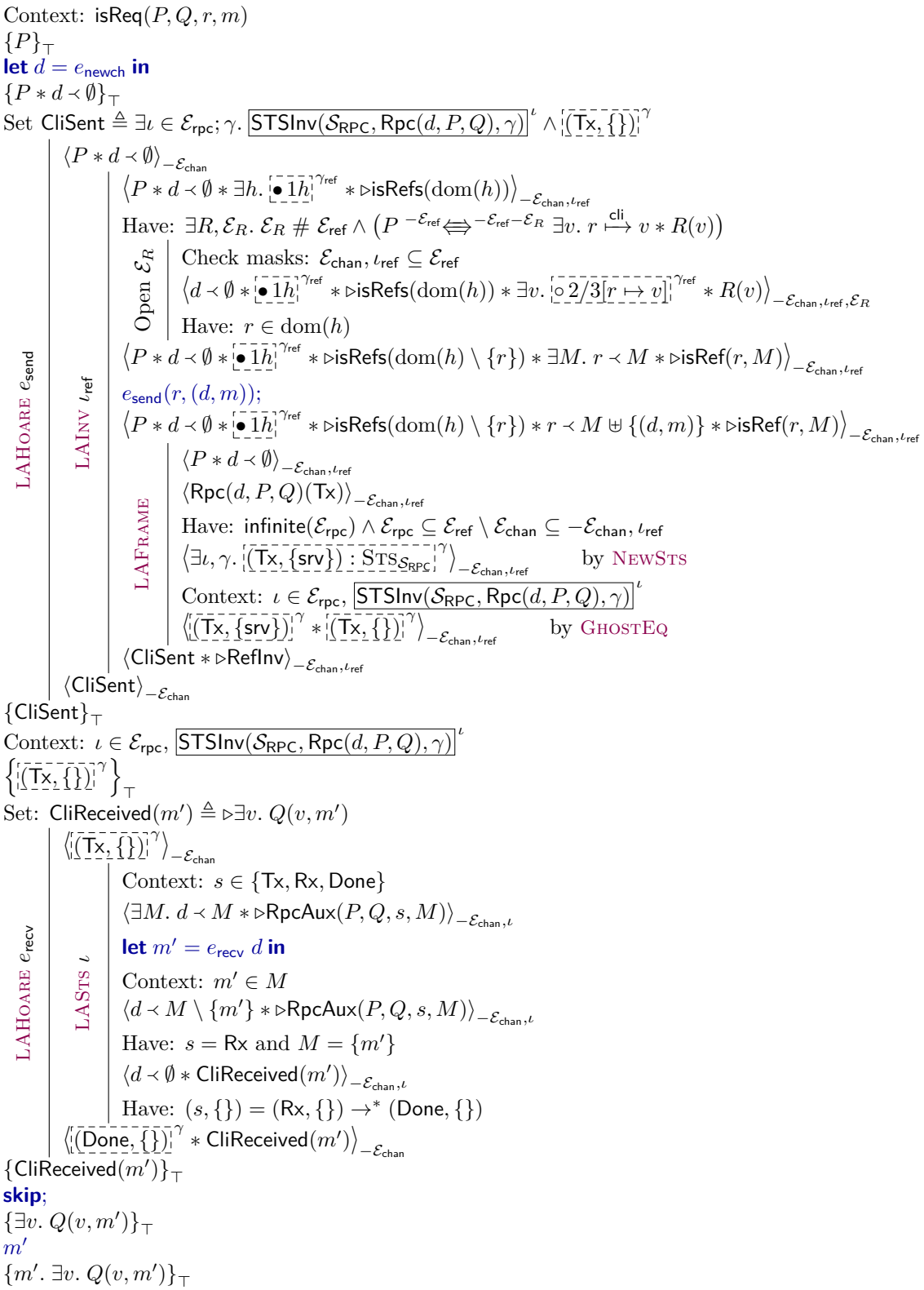


Figure 6: Proof outline for RPC.

$$
\begin{aligned}
\mathsf{isReply}(v_0, m, m', v') \triangleq\ & (m = \mathsf{Get} \land m' = v_0 \land v' = v_0) \lor \\
& (\exists w.\ m = \mathsf{Set}(w) \land m' = () \land v' = w) \lor \\
& (\exists v_1, v_2.\ m = \mathsf{Cas}(v_1, v_2) \land \mathsf{isCasReply}(v_0, v_1, v_2, m', v') \\
\mathsf{isCasReply}(v_0, v_1, v_2, b, v') \triangleq\ & (v_0 = v_1 \land b = \mathsf{true} \land v' = v_2) \lor \\
& (v_0 \neq v_1 \land b = \mathsf{false} \land v' = v_0)
\end{aligned}
$$

Figure 7: Auxiliary definitions for verifying the server.

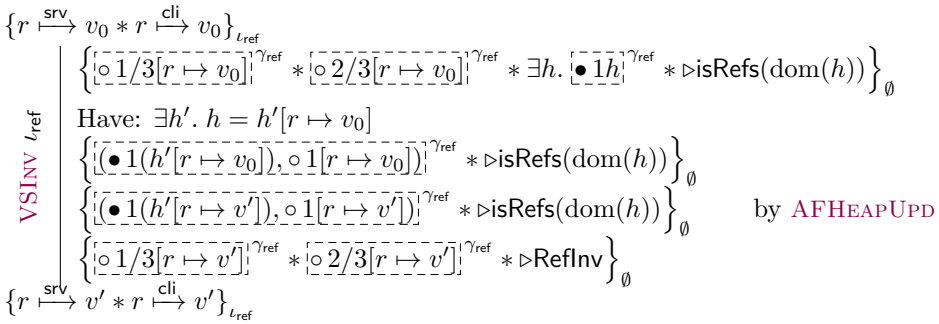


Figure 8: Proof outline for SrvUpd.

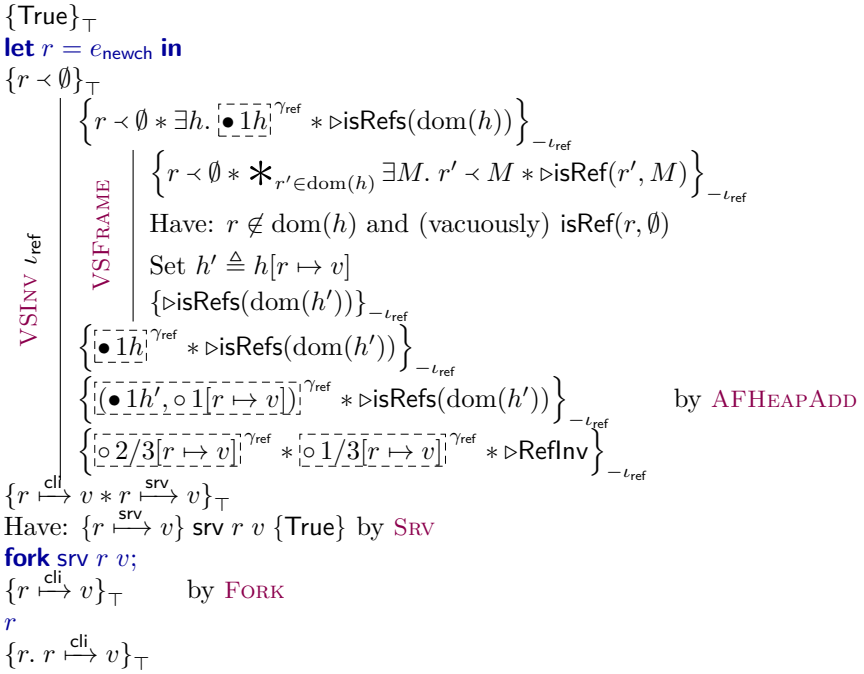


Figure 9: Proof outline for allocation, $\{\mathsf{True}\}\ \mathsf{ref}\, v\ \{r.\ r \overset{\mathsf{cli}}{\longmapsto} v\}_{\top}$.

Context: $\forall x.\ \{\triangleright r \overset{\mathsf{srv}}{\longmapsto} x\}\ \mathit{loop}\ x\ \{\mathsf{False}\}$
Set $\mathsf{SrvReceived}(tx) \triangleq \exists d, m.\ tx = (d, m) \land \triangleright \mathsf{Waiting}(r, d, m)$
$\{r \overset{\mathsf{srv}}{\longmapsto} v_0\}_\top$
LAHOARE e_{recv}
$\langle r \overset{\mathsf{srv}}{\longmapsto} v_0 \rangle_{-\mathcal{E}_{\mathsf{chan}}}$
LAINV ι_{ref}
$\left\langle \boxed{\circ\, 1/3 [r \mapsto v_0]}^{\gamma_{\mathsf{ref}}} * \exists h.\ \boxed{\bullet\, 1 h}^{\gamma_{\mathsf{ref}}} * \triangleright \mathsf{isRefs}(\mathrm{dom}(h)) \right\rangle_{-\mathcal{E}_{\mathsf{chan}}, \iota_{\mathsf{ref}}}$
Have: $r \in \mathrm{dom}(h)$
LAFRAME
$\langle \triangleright \mathsf{isRefs}(\mathrm{dom}(h) \setminus \{r\}) * \exists M.\ r \prec M * \triangleright \mathsf{isRef}(r, M) \rangle_{-\mathcal{E}_{\mathsf{chan}}, \iota_{\mathsf{ref}}}$
let $tx = e_{\mathsf{recv}}\ r$ **in**
Context: $tx \in M$
$\langle \triangleright \mathsf{isRefs}(\mathrm{dom}(h) \setminus \{r\}) * r \prec M \setminus \{tx\} * \triangleright \mathsf{isRef}(r, M \setminus \{tx\}) * \mathsf{SrvReceived}(tx) \rangle_{-\mathcal{E}_{\mathsf{chan}}, \iota_{\mathsf{ref}}}$
$\langle \triangleright \mathsf{isRefs}(\mathrm{dom}(h)) * \mathsf{SrvReceived}(tx) \rangle_{-\mathcal{E}_{\mathsf{chan}}, \iota_{\mathsf{ref}}}$
$\left\langle \boxed{\circ\, 1/3 [r \mapsto v_0]}^{\gamma_{\mathsf{ref}}} * \triangleright \mathsf{RefInv} * \mathsf{SrvReceived}(tx) \right\rangle_{-\mathcal{E}_{\mathsf{chan}}, \iota_{\mathsf{ref}}}$
$\langle r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{SrvReceived}(tx) \rangle_{-\mathcal{E}_{\mathsf{chan}}}$
$\{r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{SrvReceived}(tx)\}_\top$
let $(d, m) = tx$ **in**
$\{r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{Waiting}(r, d, m)\}_\top$
let $\mathit{reply} = \lambda m'.\ \lambda v'.\ (e_{\mathsf{send}}(d, m'); \mathit{loop}\ v')$ **in**
$\{r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{Waiting}(r, d, m)\}_\top$
Context: $\mathit{reply} = \lambda m'.\ \lambda v'.\ (e_{\mathsf{send}}(d, m'); \mathit{loop}\ v')$
Have REPLY: $\forall m', v'.\ \mathsf{isReply}(v_0, m, m', v') \Rightarrow \{r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{Waiting}(r, d, m)\}\ \mathit{reply}\ d\ m'\ \{\mathsf{False}\}$
Proof of REPLY
Context: $\mathsf{isReply}(v_0, m, m', v')$
$\{r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{Waiting}(r, d, m)\}_\top$
$e_{\mathsf{send}}(d, m');$
$\{\triangleright r \overset{\mathsf{srv}}{\longmapsto} v'\}_\top$ by SRVSEND, MONO
$\mathit{loop}\ v'$
$\{\mathsf{False}\}_\top$
$\{r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{Waiting}(r, d, m)\}_\top$
case m
of $\mathsf{Get} \Rightarrow$ Context: $m = \mathsf{Get}$; Have: $\mathsf{isReply}(v_0, m, v_0, v_0)$
$\{r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{Waiting}(r, d, m)\}_\top\ \mathit{reply}\ v_0\ v_0\ \{\mathsf{False}\}_\top$ by REPLY
| $\mathsf{Set}(w) \Rightarrow$ Context: $m = \mathsf{Set}(w)$; Have: $\mathsf{isReply}(v_0, m, (), w)$
$\{r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{Waiting}(r, d, m)\}_\top\ \mathit{reply}\ ()\ w\ \{\mathsf{False}\}_\top$ by REPLY
| $\mathsf{Cas}(v_1, v_2) \Rightarrow$ Context: $m = \mathsf{Cas}(v_1, v_2)$
$\{r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{Waiting}(r, d, m)\}_\top$
FRAME
$\{\mathsf{True}\}_\top$
let $b = (v = v_1)$ **in**
Context: $(v = v_1 \land b = \mathsf{true}) \lor (v \neq v_1 \land b = \mathsf{false})$
let $v' =$ **if** b **then** v_2 **else** v **in**
Context: $(b = \mathsf{true} \land v' = v_2) \lor (b = \mathsf{false} \land v' = v)$
Have: $\mathsf{isReply}(v_0, m, b, v')$
$\{r \overset{\mathsf{srv}}{\longmapsto} v_0 * \mathsf{Waiting}(r, d, m)\}_\top\ \mathit{reply}\ b\ v'\ \{\mathsf{False}\}_\top$ by REPLY
$\{\mathsf{False}\}_\top$

Figure 10: Proof outline for SRV.

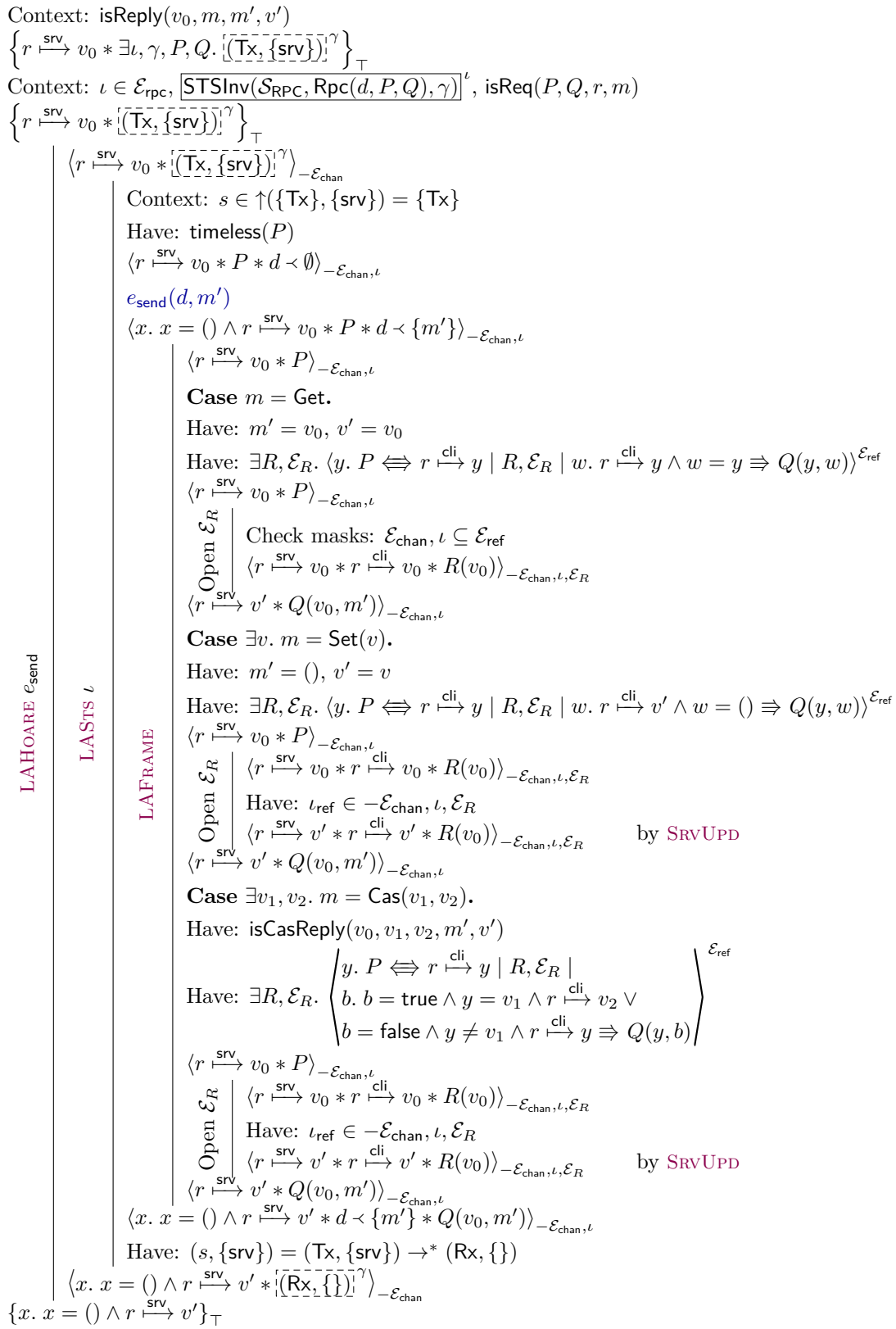


Figure 11: Proof outline for SRVSEND.

11.4 Fractional heaps

In this section, we implement fractional heaps atop (any implementation of) logically atomic references (cf. the interface in §8.4). Let masks $\mathcal{E}_{\mathsf{ref}}$, $\mathcal{E}_{\mathsf{fref}}$ and expressions e_{ref}, e_{get}, e_{set}, e_{cas} be given and assume

$$\varphi_{\mathsf{ref}}(\mathcal{E}_{\mathsf{ref}}, e_{\mathsf{ref}}, e_{\mathsf{get}}, e_{\mathsf{set}}, e_{\mathsf{cas}}) \qquad \mathcal{E}_{\mathsf{ref}} \subseteq \mathcal{E}_{\mathsf{fref}} \qquad \mathsf{infinite}(\mathcal{E}_{\mathsf{fref}} \setminus \mathcal{E}_{\mathsf{ref}})$$

We aim to view shift to

$$\begin{aligned}
&\square\exists\mapsto : \mathsf{Val} \times \mathbb{Q}_{>} \times \mathsf{Val} \to \mathsf{Prop}.\\
&\quad \forall r, q, v.\ \mathsf{timeless}(r \overset{q}{\mapsto} v) \land\\
&\quad \forall r, q, v.\ r \overset{q}{\mapsto} v \Rightarrow r \overset{q}{\mapsto} v * q \in (0, 1] \land\\
&\quad \forall r, q_1, q_2, v, w.\ r \overset{q_1}{\mapsto} v * r \overset{q_2}{\mapsto} w \Leftrightarrow r \overset{q_1+q_2}{\mapsto} v * v = w \land\\
&\quad \forall v.\ \left\{\mathsf{True}\right\} e_{\mathsf{ref}}\ v \left\{r.\ r \overset{1}{\mapsto} v\right\} \land\\
&\quad \forall r.\ \left\langle v, q.\ r \overset{q}{\mapsto} v\right\rangle e_{\mathsf{get}}\ r \left\langle w.\ r \overset{q}{\mapsto} v \land w = v\right\rangle^{\mathcal{E}_{\mathsf{fref}}}\\
&\quad \forall r, v.\ \left\langle r \overset{1}{\mapsto} _\right\rangle e_{\mathsf{set}}\ r\ v \left\langle x.\ r \overset{1}{\mapsto} v \land x = ()\right\rangle^{\mathcal{E}_{\mathsf{fref}}}\\
&\quad \forall r, v_1, v_2.\ \left\langle v.\ r \overset{1}{\mapsto} v\right\rangle e_{\mathsf{cas}}(r, v_1, v_2) \left\langle b.\ \begin{array}{l} b = \mathsf{true} \land v = v_1 \land r \overset{1}{\mapsto} v_2 \lor\\ b = \mathsf{false} \land v \neq v_1 \land r \overset{1}{\mapsto} v \end{array}\right\rangle^{\mathcal{E}_{\mathsf{fref}}}
\end{aligned}$$

To this end, we allocate an instance γ_F of the monoid $\mathrm{AFHEAP}(\mathit{Val})$ from §7.7. Then we establish an invariant $\iota_F \in \mathcal{E}_{\mathsf{fref}} \setminus \mathcal{E}_{\mathsf{ref}}$ preserving

$$\exists h.\ \boxed{\bullet 1 h : \mathrm{AFHEAP}(\mathit{Val})}^{\gamma_F} * \mathop{\Large *}_{r \in \mathrm{dom}(h)} r \mapsto h(r)$$

Finally, we define

$$r \overset{q}{\mapsto} v \triangleq \boxed{\circ\, r \mapsto (q, v)}^{\gamma_F}$$

The spec now follows from the frame-preserving updates for authoritative fractional heaps and the following view shifts.

$$\begin{aligned}
&r \overset{1}{\mapsto} v \ {}^{\iota_F}\!\Longrightarrow\!\!\!\!\Longleftarrow^{\iota_F}\ r \mapsto v\\
&r \overset{q}{\mapsto} v \ {}^{\iota_F}\!\Longrightarrow\!\!\!\!\Longleftarrow^{\emptyset}\ r \overset{q}{\mapsto} v * r \mapsto v * \exists h.\ \boxed{\bullet\, 1(h[r \mapsto v])}^{\gamma_F} * \mathop{\Large *}_{x \in \mathrm{dom}(h)} x \mapsto h(x)
\end{aligned}$$

We will write $r \overset{-}{\mapsto} v$ for $\exists q.\ r \overset{q}{\mapsto} v$.

12 Language foundations

In this section, we define a language with asynchronous channels. After deriving basic proof rules for the language, we prove the channel interface (φ_{chan} from §11) so that later examples can use (encoded) references.

12.1 Grammar

The syntax of the language assumes disjoint, countably infinite sets *Chan* and *Var* of channel names and variables. The language has asynchronous channels, recursive functions, sums, and products.

$$
\begin{aligned}
c, d &\in \mathit{Chan} \\
x, y, f &\in \mathit{Var} \\
v, w &::= x \mid c \mid \mathbf{rec}\ f(x).\ e \mid () \mid (v, v) \mid \mathbf{inj}_i\ v \\
e &::= v \mid e\ e \mid (e, e) \mid e.i \mid \mathbf{inj}_i\ e \mid \\
&\quad\ \mathbf{case}\ e\ \mathbf{of}\ \mathbf{inj}_1\ x \Rightarrow e \mid \mathbf{inj}_2\ x \Rightarrow e \mid \\
&\quad\ \mathbf{newch} \mid \mathbf{send}(e, e) \mid \mathbf{tryrecv}\ e \mid \mathbf{fork}\ e \\
K &::= [] \mid \mathbf{send}(K, e) \mid \mathbf{send}(v, K) \mid \mathbf{tryrecv}\ K \mid \cdots
\end{aligned}
$$

We define some convenient derived forms:

$$\mathsf{true} \triangleq \mathbf{inj}_1\ () \qquad \mathsf{false} \triangleq \mathbf{inj}_2\ () \qquad \mathbf{if}\ e\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 \triangleq \mathbf{case}\ e\ \mathbf{of}\ \mathbf{inj}_1\ \_ \Rightarrow e_1 \mid \mathbf{inj}_2\ \_ \Rightarrow e_2$$

$$\mathsf{None} \triangleq \mathbf{inj}_1\ () \qquad\qquad \mathsf{Some}(e) \triangleq \mathbf{inj}_2\ e$$

$$\lambda x.\ e \triangleq \mathbf{rec}\ \_(x).\ e \qquad \mathbf{let}\ x = e\ \mathbf{in}\ e' \triangleq (\lambda x.\ e')\ e \qquad e; e' \triangleq \mathbf{let}\ \_ = e\ \mathbf{in}\ e'$$

12.2 Operational semantics

The semantics of the (pure) functional part are completely standard and therefore omitted.

$$
\begin{aligned}
c &\in \mathit{Chan} \\
M &\in \mathit{Bag} \quad \text{(finite bags of values)} \\
C &\in \mathit{Chan} \xrightarrow{\text{fin}} \mathit{Bag}
\end{aligned}
$$

$$
\begin{aligned}
C; \mathbf{newch} &\to C[c \mapsto \emptyset]; c \\
C[c \mapsto M]; \mathbf{send}(c, m) &\to C[c \mapsto M \uplus \{m\}]; () \\
C[c \mapsto \emptyset]; \mathbf{tryrecv}\ c &\to C[c \mapsto \emptyset]; \mathsf{None} \\
C[c \mapsto M \uplus \{m\}]; \mathbf{tryrecv}\ c &\to C[c \mapsto M]; \mathsf{Some}(m)
\end{aligned}
$$

12.3 Basic Hoare triples

Rules from the operational semantics. Using the lifting lemmas (§6.5), we obtain basic rules from the operational semantics:

$$\{\lfloor C \rfloor\}\ \mathbf{newch}\ \{c.\ \lfloor C[c \mapsto \emptyset] \rfloor\} \qquad \{\lfloor C[c \mapsto M] \rfloor\}\ \mathbf{send}(c, m)\ \{x.\ x = () \land \lfloor C[c \mapsto M \uplus \{m\}] \rfloor\}$$

$$\{\lfloor C[c \mapsto M \uplus \{m\}] \rfloor\}\ \mathbf{tryrecv}\ c\ \{x.\ x = \mathsf{Some}(m) \land \lfloor C[c \mapsto M] \rfloor\}$$

$$\{\lfloor C[c \mapsto \emptyset] \rfloor\}\ \mathbf{tryrecv}\ c\ \{x.\ x = \mathsf{None} \land \lfloor C[c \mapsto \emptyset] \rfloor\}$$

$$\frac{\{P\}\ v_1\ \{w.\ w = v_1 \land Q\}}{\{\triangleright P\}\ (v_1, v_2).1\ \{w.\ w = v_1 \land Q\}} \qquad \frac{\{P\}\ v_2\ \{w.\ w = v_2 \land Q\}}{\{\triangleright P\}\ (v_1, v_2).2\ \{w.\ w = v_2 \land Q\}}$$

$$\frac{\{P\}\ e_1[v/x]\ \{w.\ Q\}}{\{\triangleright P\}\ \mathbf{case}\ \mathbf{inj}_1\ v\ \mathbf{of}\ \mathbf{inj}_1\ x \Rightarrow e_1 \mid \mathbf{inj}_2\ x \Rightarrow e_2\ \{w.\ Q\}}$$

$$\frac{\{P\}\ e_2[v/x]\ \{w.\ Q\}}{\{\triangleright P\}\ \mathbf{case}\ \mathbf{inj}_2\ v\ \mathbf{of}\ \mathbf{inj}_1\ x \Rightarrow e_1 \mid \mathbf{inj}_2\ x \Rightarrow e_2\ \{w.\ Q\}} \qquad \frac{\{P\}\ e[\mathbf{rec}\ f(x).\ e/f, v/x]\ \{w.\ Q\}}{\{\triangleright P\}\ (\mathbf{rec}\ f(x).\ e)\ v\ \{w.\ Q\}}$$

Note that the $\lfloor C \rfloor$ assertions above are the physical assertions saying that the global state is C.

Rules for atomic reductions. We would like a single rule for **tryrecv** that applies in all cases. Furthermore, we will only rely on logical atomicity in the following. Hence we derive the following triples:

$$\langle C, M.\ \lfloor C[c \mapsto M] \rfloor \rangle\ \mathbf{send}(c, v)\ \langle x.\ x = () \land \lfloor C[c \mapsto M \uplus \{v\}] \rfloor \rangle$$

$$\left\langle C, M.\ \lfloor C[c \mapsto M] \rfloor \right\rangle\ \mathbf{tryrecv}\ c\ \left\langle \begin{array}{l} x.\ M = \emptyset \land x = \mathsf{None} \land \lfloor C[c \mapsto \emptyset] \rfloor \lor \\ \quad \exists v.\ v \in M \land x = \mathsf{Some}(v) \land \lfloor C[c \mapsto M \setminus \{v\}] \rfloor \end{array} \right\rangle$$

Proof. The triple for **send** follows immediately via LAAtomic.

For **tryrecv**, by LAAtomic, it suffices to show

$$\forall C, M. \left\{ \lfloor C[c \mapsto M] \rfloor \right\}\ \mathbf{tryrecv}\ c\ \left\{ \begin{array}{l} x.\ M = \emptyset \land x = \mathsf{None} \land \lfloor C[c \mapsto \emptyset] \rfloor \lor \\ \quad \exists v.\ v \in M \land x = \mathsf{Some}(v) \land \lfloor C[c \mapsto M \setminus \{v\}] \rfloor \end{array} \right\}$$

So assume some C, M. If M is empty, use the corresponding base rule for **tryrecv**, and Csq. Otherwise, let $v \in M$ and use the corresponding base rule for **tryrecv** with $M \setminus \{v\}$, and Csq. □

Rules for pure reductions. For pure reductions, the following (easier to use) rules are derivable:

ProjL
$$\{\triangleright P\}\ (v_1, v_2).1\ \{w.\ w = v_1 \land P\}$$

ProjR
$$\{\triangleright P\}\ (v_1, v_2).2\ \{w.\ w = v_2 \land P\}$$

Case
$$\frac{\forall v'.\ \{v = \mathbf{inj}_1\ v' \land P\}\ e_1[v'/x]\ \{w.\ Q\} \qquad \forall v'.\ \{v = \mathbf{inj}_2\ v' \land P\}\ e_2[v'/x]\ \{w.\ Q\}}{\{\triangleright P\}\ \mathbf{case}\ v\ \mathbf{of}\ \mathbf{inj}_1\ x \Rightarrow e_1 \mid \mathbf{inj}_2\ x \Rightarrow e_2\ \{w.\ Q\}}$$

Rec
$$\frac{\hat{f} : \mathsf{Val} \mid (\forall x.\ \{\triangleright P\}\ \hat{f}\ x\ \{w.\ Q\}) \vdash_{\Box} \forall v.\ \{P\}\ e[\hat{f}/f, v/x]\ \{w.\ Q\}}{\forall v.\ \{\triangleright P\}\ (\mathbf{rec}\ f(x).\ e)\ v\ \{w.\ Q\}}$$

Proof. The projections are immediate, using rule Ret.

(Case) By rules Exist and Disj, it suffices to show

$$\forall v'.\ \{v = \mathbf{inj}_1\ v' \land \triangleright P\}\ \mathbf{case}\ v\ \mathbf{of}\ \mathbf{inj}_1\ x \Rightarrow e_1 \mid \mathbf{inj}_2\ x \Rightarrow e_2\ \{w.\ Q\} \tag{25}$$

$$\forall v'.\ \{v = \mathbf{inj}_2\ v' \land \triangleright P\}\ \mathbf{case}\ v\ \mathbf{of}\ \mathbf{inj}_1\ x \Rightarrow e_1 \mid \mathbf{inj}_2\ x \Rightarrow e_2\ \{w.\ Q\} \tag{26}$$

We prove (25); the second triple is analogous. By BoxOut on the equality and substitution, it suffices to show

$$\forall v'.\ \{v = \mathbf{inj}_1\ v' \land \triangleright P\}\ \mathbf{case}\ \mathbf{inj}_1\ v'\ \mathbf{of}\ \mathbf{inj}_1\ x \Rightarrow e_1 \mid \mathbf{inj}_2\ x \Rightarrow e_2\ \{w.\ Q\}$$

By Mono and the axiom for commuting $\triangleright$ with conjunction, it suffices to show

$$\forall v'.\ \{\triangleright(v = \mathbf{inj}_1\ v' \land P)\}\ \mathbf{case}\ \mathbf{inj}_1\ v'\ \mathbf{of}\ \mathbf{inj}_1\ x \Rightarrow e_1 \mid \mathbf{inj}_2\ x \Rightarrow e_2\ \{w.\ Q\}$$

This is immediate, using the basic rule and our first premise.

(Rec) By Löb we can assume

$$\triangleright \forall v.\ \{\triangleright P\}\ (\mathbf{rec}\ f(x).\ e)\ v\ \{w.\ Q\}$$

to show our goal. Assume we are given some v. Since $\Box$ commutes with both $\triangleright$ and universal quantification, we can use BoxOut to move our assumption into the precondition of the triple, so our goal becomes:

$$\{(\triangleright \forall v.\ \{\triangleright P\}\ (\mathbf{rec}\ f(x).\ e)\ v\ \{w.\ Q\}) * \triangleright P\}\ (\mathbf{rec}\ f(x).\ e)\ v\ \{w.\ Q\}$$

By the basic rule for **rec**, it suffices to show

$$\{(\forall v.\ \{\triangleright P\}\ (\mathbf{rec}\ f(x).\ e)\ v\ \{w.\ Q\}) * P\}\ e[\mathbf{rec}\ f(x).\ e/f, v/x]\ \{w.\ Q\}$$

Using BoxOut again, we can transform this goal to

$$(\forall v.\ \{\triangleright P\}\ (\mathbf{rec}\ f(x).\ e)\ v\ \{w.\ Q\}) \vdash_{\Box} \{P\}\ e[\mathbf{rec}\ f(x).\ e/f, v/x]\ \{w.\ Q\}$$

Choosing $\hat{f} \triangleq \mathbf{rec}\ f(x).\ e$, this is an instance of our premise.

Note that f is a *language-level* variable, while $\hat{f}$ is a *logic-level* variable of sort Val, representing a language-level term. In actual proofs, we are usually going to gloss over this distinction, and give both the same name, so the substitution looks like a no-op. □

Rules for derived forms. The following rules may be derived:

Lam
$$\frac{\{P\}\ e[v/x]\ \{w.\ Q\}}{\{\triangleright P\}\ (\lambda x.\ e)\ v\ \{w.\ Q\}}$$

Let
$$\frac{\{P\}\ e[v/x]\ \{w.\ Q\}}{\{\triangleright P\}\ \mathbf{let}\ x = v\ \mathbf{in}\ e\ \{w.\ Q\}}$$

Seq
$$\frac{\{P\}\ e\ \{w.\ Q\}}{\{\triangleright P\}\ v; e\ \{w.\ Q\}}$$

If
$$\frac{\{b = \mathsf{true} \wedge P\}\ e_1\ \{v.\ Q\} \qquad \{b = \mathsf{false} \wedge P\}\ e\ \{v.\ Q\}}{\{\triangleright P\}\ \mathbf{if}\ b\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2\ \{v.\ Q\}}$$

12.4 Fractional physical resources

The ref module assumed the ability to own an individual channel without caring about the others: $c \prec M$. However, the only possible assertion about the physical state is fixing the complete current state. Iris is powerful enough to let us re-derive the usual separation-logical assertions from these basics (in this case for channel names, not heap locations). We will go one step further than necessary and provide fractional permissions as well: $c \prec M$ is just short for $c \overset{1}{\prec} M$

To do that, define BHeap $\triangleq$ AFHeap(Bag), using the construction defined in §7.7.

Now create an instance γ_C of BHeap and an invariant ι_C preserving

$$\exists C.\ \lfloor C \rfloor * \boxed{\bullet\, 1C}^{\gamma_C}$$

(We leave implicit the coercions between elements of BHeap and the finite partial functions underlying physical state assertions.)

We define $c \overset{q}{\prec} M \triangleq \boxed{\circ\, c \mapsto (q, M)}^{\gamma_C}$ and $\mathcal{E}_{\mathsf{chan}} \triangleq \{\iota_C\}$. Now we can easily show that owning $\prec$ suffices to send and receive messages:

NewCh
$$\left\langle \mathsf{True} \right\rangle\ \mathbf{newch}\ \left\langle c.\ c \overset{1}{\prec} \emptyset \right\rangle^{\mathcal{E}_{\mathsf{chan}}}$$

Send
$$\left\langle M.\ c \overset{1}{\prec} M \right\rangle\ \mathbf{send}(c, m)\ \left\langle x.\ x = () \wedge c \overset{1}{\prec} M \uplus \{m\} \right\rangle^{\mathcal{E}_{\mathsf{chan}}}$$

TryRecv
$$\left\langle M.\ c \overset{1}{\prec} M \right\rangle\ \mathbf{tryrecv}\ c\ \left\langle x.\ \begin{array}{l} M = \emptyset \wedge x = \mathsf{None} \wedge c \overset{1}{\prec} \emptyset \vee \\ (\exists m.\ m \in M \wedge x = \mathsf{Some}(m) \wedge c \overset{1}{\prec} M \setminus \{m\}) \end{array} \right\rangle^{\mathcal{E}_{\mathsf{chan}}}$$

Proof outline for Send.

Context: $\left\langle M.\ P \Longleftrightarrow c \overset{1}{\prec} M \mid R, \mathcal{E}_R \mid x.\ x = () \wedge c \overset{1}{\prec} M \uplus \{m\} \Rrightarrow Q \right\rangle^{\mathcal{E}_{\mathsf{chan}}}$

$\{P\}_\top$

LAHoare

$\langle P\rangle_\top$

Open $\mathcal{E}_R$

Check masks: $\emptyset \subseteq \mathcal{E}_{\mathsf{chan}}$

$\langle c \prec M * R(M)\rangle_{-\mathcal{E}_R}$

Open ι_C, Frame $R(M)$

$\left\langle \lfloor C[c \mapsto M]\rfloor * \boxed{\bullet C[c \mapsto M], \circ c \mapsto M}^{\gamma_C} \right\rangle_{-\mathcal{E}_R, \iota_C}$

send(c, m)

$\left\langle x = () * \lfloor C[c \mapsto M \uplus \{m\}]\rfloor * \boxed{\bullet C[c \mapsto M], \circ c \mapsto M}^{\gamma_C} \right\rangle_{-\mathcal{E}_R, \iota_C}$

By AFHeapUpd: $(\bullet C[c \mapsto M], \circ c \mapsto M) \rightsquigarrow (\bullet C[c \mapsto M \uplus \{m\}], \circ c \mapsto M \uplus \{m\})$

$\left\langle x = () * \lfloor C[c \mapsto M \uplus \{m\}]\rfloor * \boxed{\bullet C[c \mapsto M \uplus \{m\}], \circ c \mapsto M \uplus \{m\}}^{\gamma_C} \right\rangle_{-\mathcal{E}_R, \iota_C}$

$\left\langle x = () * c \overset{1}{\prec} M \uplus \{m\} * R(M) \right\rangle_{-\mathcal{E}_R}$

$\{Q(M, x)\}_\top$

□

The other proofs are similar.

12.5 Blocking receive

The **tryrecv** expression that comes with the language is non-blocking: It always returns an option immediately, but may return None if there was nothing to receive. We lift this to a blocking operation that waits until there is a message to deliver:

$$\begin{aligned} \mathsf{recv} \triangleq\ & \textbf{rec}\ \mathit{loop}(c). \\ & \textbf{let}\ v = \textbf{tryrecv}\ c\ \textbf{in} \\ & \textbf{case}\ v\ \textbf{of}\ \mathsf{None} \Rightarrow \mathit{loop}\ c \mid \mathsf{Some}(m) \Rightarrow m \end{aligned}$$

To satisfy φ_{chan}, we want to show the following logically atomic specification for this code:

Recv

$$\left\langle M.\ c \overset{1}{\prec} M \right\rangle\ \mathsf{recv}\ c\ \left\langle m.\ m \in M \wedge c \overset{1}{\prec} M \setminus \{m\} \right\rangle^{\mathcal{E}_{\mathsf{chan}}}$$

The proof makes crucial use of the fact that we can *abort* an atomic update in case there was no message to receive.

Proof outline for Recv.

Context: $\left\langle M.\ P \Longleftrightarrow c \overset{1}{\prec} M \mid R, \mathcal{E}_R \mid m.\ m \in M \wedge c \overset{1}{\prec} M \setminus \{m\} \Rrightarrow Q \right\rangle^{\mathcal{E}_{\mathsf{chan}}}$

Context: $\forall x.\ \{\triangleright P\}\ \mathit{loop}\ c\ \{m.\ \exists M.\ Q(M, m)\}$

$\{P\}_\top$

LAHoare

$\langle P \rangle_{-\mathcal{E}_{\mathsf{chan}}}$

Open $\mathcal{E}_R$

Check masks: $\mathcal{E}_{\mathsf{chan}} \subseteq \mathcal{E}_{\mathsf{chan}}$

$\left\langle c \overset{1}{\prec} M * R(M) \right\rangle_{-\mathcal{E}_{\mathsf{chan}}, \mathcal{E}_R}$

let $v =$ **tryrecv** c **in**

$$\left\langle \begin{array}{l} ((M = \emptyset \wedge v = \mathsf{None} \wedge c \overset{1}{\prec} \emptyset) \vee \\ (\exists m.\ m \in M \wedge v = \mathsf{Some}(m) \wedge c \overset{1}{\prec} M \setminus \{m\})) * R(M) \end{array} \right\rangle_{-\mathcal{E}_{\mathsf{chan}}, \mathcal{E}_R}$$

Left: $\left\langle v = \mathsf{None} \wedge c \overset{1}{\prec} \emptyset * R(\emptyset) \right\rangle_{-\mathcal{E}_{\mathsf{chan}}, \mathcal{E}_R}$

Right: $\left\langle v = \mathsf{Some}(m) \wedge m \in M * c \overset{1}{\prec} M \setminus \{m\} * R(M) \right\rangle_{-\mathcal{E}_{\mathsf{chan}}, \mathcal{E}_R}$

$\langle P * v = \mathsf{None} \vee \exists M, m.\ Q(M, m) * v = \mathsf{Some}(m) \rangle_{-\mathcal{E}_{\mathsf{chan}}}$

$\{P * v = \mathsf{None} \vee \exists M, m.\ Q(M, m) * v = \mathsf{Some}(m)\}_\top$

case v

of None $\Rightarrow$

$\{P\}_\top$ *loop* c $\{m.\ Q(M, m)\}_\top$

| Some(m) $\Rightarrow$

$\{\exists M.\ Q(M, m)\}_\top$ m $\{m.\ \exists M.\ Q(M, m)\}_\top$

□

This completes the implementation of the channel specification in Figure 3.

13 MCAS

We seek to implement the following spec for multi-word compare-and-swap, based on the MCAS example from TaDA [2].

$$
\begin{array}{l}
\exists \mathsf{isMCL}, \mathsf{MCP}. \\
\quad \forall n, x, v.\ \mathsf{timeless}(\mathsf{MCP}(n, x, v)) \land \\
\quad \forall n, x, v, w.\ \mathsf{MCP}(n, x, v) * \mathsf{MCP}(n, x, w) \Rightarrow \mathsf{False} \land \\
\quad \forall \mathcal{E}.\ \mathcal{E} \supseteq \mathcal{E}_{\mathsf{ref}} \land \mathrm{infinite}(\mathcal{E} \setminus \mathcal{E}_{\mathsf{ref}}) \Rightarrow \{\mathsf{True}\}\ \texttt{newMCL()}\ \{l.\ \exists n.\ \Box \mathsf{isMCL}(l, \mathcal{E}, n)\} \land \\
\quad \forall l, \mathcal{E}, n, x, v.\ \mathsf{isMCL}(l, \mathcal{E}, n) \Rightarrow x \mapsto v \Rrightarrow_{\mathcal{E}} \mathsf{MCP}(n, x, v) \land \\
\quad \forall l, \mathcal{E}, n, x, v.\ \mathsf{isMCL}(l, \mathcal{E}, n) \Rightarrow \{\mathsf{MCP}(n, x, v)\}\ \texttt{unmakeMCP}(l)\ \{x \mapsto v\} \land \\
\quad \forall l, \mathcal{E}, n, x, w.\ \mathsf{isMCL}(l, \mathcal{E}, n) \Rightarrow \langle v.\ \mathsf{MCP}(n, x, v) \rangle\ \texttt{write}(l, x, w)\ \langle \mathsf{MCP}(n, x, w) \rangle^{\mathcal{E}} \\
\quad \forall l, \mathcal{E}, n, x.\ \mathsf{isMCL}(l, \mathcal{E}, n) \Rightarrow \langle v.\ \mathsf{MCP}(n, x, v) \rangle\ \texttt{read}(l, x)\ \langle w.\ \mathsf{MCP}(n, x, v) \land v = w \rangle^{\mathcal{E}} \\
\quad \forall l, \mathcal{E}, n, x, v_0, v_1.\ \mathsf{isMCL}(l, \mathcal{E}, n) \Rightarrow \\
\qquad \left\langle v.\ \mathsf{MCP}(n, x, v) \right\rangle\ \texttt{cas}(l, x, v_0, v_1)\ \left\langle b.\ \begin{array}{l} b = \mathsf{true} \land v = v_0 \land \mathsf{MCP}(n, x, v_1) \lor \\ b = \mathsf{false} \land v \neq v_0 \land \mathsf{MCP}(n, x, v) \end{array} \right\rangle^{\mathcal{E}} \\
\quad \forall l, \mathcal{E}, n, x, y, v_0, v_1, w_0, w_1.\ \ \mathsf{isMCL}(l, \mathcal{E}, n) \Rightarrow \\
\qquad \langle v, w.\ \mathsf{MCP}(n, x, v) * \mathsf{MCP}(n, y, w) \rangle \\
\qquad\quad \texttt{dcas}(l, x, y, v_0, w_0, v_1, w_1) \\
\qquad \left\langle b.\ \begin{array}{l} b = \mathsf{true} \land v = v_0 \land w = w_0 \land \mathsf{MCP}(n, x, v_1) * \mathsf{MCP}(n, y, w_1) \lor \\ b = \mathsf{false} \land (v \neq v_0 \lor w \neq w_0) \land \mathsf{MCP}(n, x, v) * \mathsf{MCP}(n, y, w) \end{array} \right\rangle^{\mathcal{E}} \\
\quad \forall l, \mathcal{E}, n, x, y, z, v_0, v_1, w_0, w_1, u_0, u_1.\ \ \mathsf{isMCL}(l, \mathcal{E}, n) \Rightarrow \\
\qquad \langle v, w, u.\ \mathsf{MCP}(n, x, v) * \mathsf{MCP}(n, y, w) * \mathsf{MCP}(n, t, u) \rangle \\
\qquad\quad \texttt{3cas}(l, x, y, z, v_0, w_0, u_0, v_1, w_1, u_1) \\
\qquad \left\langle b.\ \begin{array}{l} b = \mathsf{true} \land v = v_0 \land w = w_0 \land u = u_0 \land \mathsf{MCP}(n, x, v_1) * \mathsf{MCP}(n, y, w_1) * \mathsf{MCP}(n, z, u_1) \lor \\ b = \mathsf{false} \land (v \neq v_0 \lor w \neq w_0 \lor u \neq u_0) \land \mathsf{MCP}(n, x, v) * \mathsf{MCP}(n, y, w) * \mathsf{MCP}(n, z, u) \end{array} \right\rangle^{\mathcal{E}}
\end{array}
$$

As discussed in §8.4, we use a ghost heap to map a single *logical name* n to a tuple of ghost names. Furthermore, because MCAS provides atomic triples, we have to keep track of the set of invariants it could potentially use.

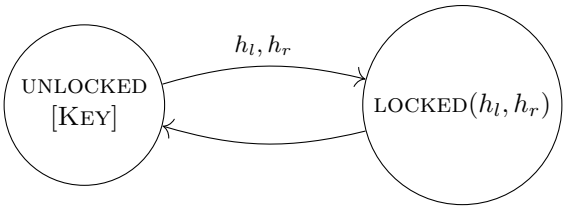
This specification is slightly stronger than the one in TaDA in that it forces `makeMCP` (adding a memory cell to the MCAS-managed heap) to be a purely logical operation—a view shift. We do not require the same for `unmakeMCP` as there are too many $\triangleright$'s accumulating in the proof to make that work out.

We are now going to show that the following implementation of `newMCL`, `unmakeMCP`, and `write` satisfy (parts of) the specification.

$$
\begin{aligned}
\texttt{newMCL} &\triangleq \lambda\_.\ \texttt{newLock}() \\
\texttt{unmakeMCP} &\triangleq \lambda l.\ \texttt{lock}(l); \texttt{unlock}(l) \\
\texttt{write} &\triangleq \lambda(l, x, w).\ \texttt{lock}(l); x := w; \texttt{unlock}(l)
\end{aligned}
$$

The missing operations can be implemented just like `write`, and proven using the same reasoning. We assume an implementation of the logically atomic specs for references (§11) and locks (§10.1) and that $\mathcal{E}_{\mathsf{ref}} = \mathcal{E}_{\mathsf{lk}}$.

The MCAS library will have at its core the following protocol:



This defines an STS $\mathcal{S}$. Note that there are infinitely many states $\textsc{locked}(h_l, h_r)$ for all possible values of h_l and h_r, with a transition from $\textsc{unlocked}$ to each of them. The transitions to $\textsc{unlocked}$ require giving up the $\textsc{Key}$ token, so it can only be taken if the current thread holds this token. Let $\textsc{Mcas} \triangleq \textsc{Sts}_{\mathcal{S}}$ be the monoid describing the STS.

Furthermore, let $\textsc{LHeap} \triangleq \textsc{FpFun}(\mathit{Val}, \textsc{Ex}(\mathit{Val}))$ be the monoid of (non-fractional) heaps. MCAS uses $\textsc{Auth}(\textsc{LHeap})$ to track its logical state.

The state interpretations and invariants are defined as follows:

$$\begin{aligned}
\mathsf{MCAS}(\gamma_M)(\textsc{unlocked}) &\triangleq \exists h.\ \boxed{\bullet h : \textsc{Auth}(\textsc{LHeap})}^{\gamma_M} * h \\
\mathsf{MCAS}(\gamma_M)(\textsc{locked}(h_l, h_r)) &\triangleq \mathrm{dom}(h_l) = \mathrm{dom}(h_r) * \exists h_u.\ \boxed{\bullet h_l \cdot h_u}^{\gamma_M} * h_r * h_u \\
\mathsf{MLock}(\textsc{unlocked}) &\triangleq 0 \\
\mathsf{MLock}(\textsc{locked}(h_l, h_r)) &\triangleq 1 \\
\mathsf{MCLInv}(l, n) &\triangleq \exists \gamma_M, \gamma_K.\ n \overset{-}{\hookrightarrow} (\gamma_M, \gamma_K) * \\
&\qquad \mathsf{STSInv}(\mathcal{S}, a \mapsto \mathsf{MCAS}(\gamma_M)(a) * \mathsf{Lock}(l, \mathsf{MLock}(a)), \gamma_K)
\end{aligned}$$

We are factoring out the lock state from the remainder of the state interpretation to simplify the proofs. Note that $\mathsf{MCAS}(\gamma_M)$ is timeless for all states, and hence MCLInv is timeless.

Note that even the interpretation of the current real state of the locked part of the heap, resides in the invariant. This is different from the TaDA invariant. It prevents the owner of the lock from trying to re-add the same cell to the library, which is crucial to make the operation of adding a memory cell a view shift.

Define the user-facing abstract predicates to be

$$\begin{aligned}
\mathsf{MCP}(n, x, v) &\triangleq \exists \gamma_M, \gamma_K.\ n \overset{-}{\hookrightarrow} (\gamma_M, \gamma_K) * \boxed{\circ\, x \mapsto v}^{\gamma_M} \\
\mathsf{isMCL}(l, \mathcal{E}, n) &\triangleq \mathcal{E} \supseteq \mathcal{E}_{\mathsf{ref}} * \exists \iota.\ \iota \in \mathcal{E} \setminus \mathcal{E}_{\mathsf{ref}} * \boxed{\mathsf{MCLInv}(l, n)}^{\iota}
\end{aligned}$$

You can find the proofs of `newMCL` in Figure 12, `makeMCP` (the view shift adding a cell to the library) in Figure 13, `unmakeMCP` in Figure 14, and `write` in Figure 15.

Context: $\mathcal{E} \supseteq \mathcal{E}_{\mathsf{ref}}$, $\mathsf{infinite}(\mathcal{E} \setminus \mathcal{E}_{\mathsf{ref}})$
$\{\mathsf{True}\}_\top$
`newLock()`
$\{l.\ \mathsf{Lock}(l, \mathsf{MLock}(\textsc{unlocked}))\}_\top$
$\left\{l.\ \mathsf{Lock}(l, \mathsf{MLock}(\textsc{unlocked})) * \exists \gamma_M.\ \boxed{\bullet \emptyset : \textsc{Auth}(\textsc{LHeap})}^{\gamma_M}\right\}_\top$ by NewGhost
Set $\varphi \triangleq \lambda a.\ \mathsf{MCAS}(\gamma_M)(a) * \mathsf{Lock}(l, \mathsf{MLock}(a))$
$\{l.\ \varphi(\textsc{unlocked})\}_\top$
$\{l.\ \exists \gamma_K.\ \mathsf{STSInv}(\mathcal{S}, \varphi, \gamma_K)\}_\top$ by NewGhost
$\{l.\ \exists n.\ \mathsf{MCLInv}(l, n)\}_\top$ by view shift in §8.4
$\{l.\ \square\mathsf{isMCL}(l, \mathcal{E}, n)\}_\top$ by NewInv

Figure 12: Proof of `newMCL`.

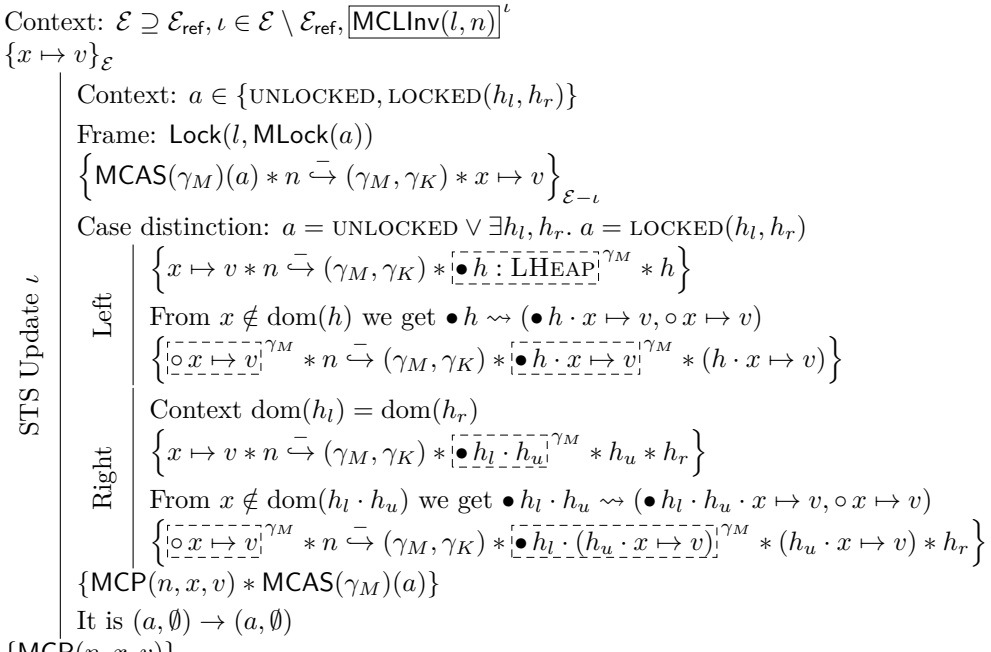


Figure 13: Proof of `makeMCP`: $x \mapsto v \Rrightarrow_{\mathcal{E}} \mathsf{MCP}(n, x, v)$.

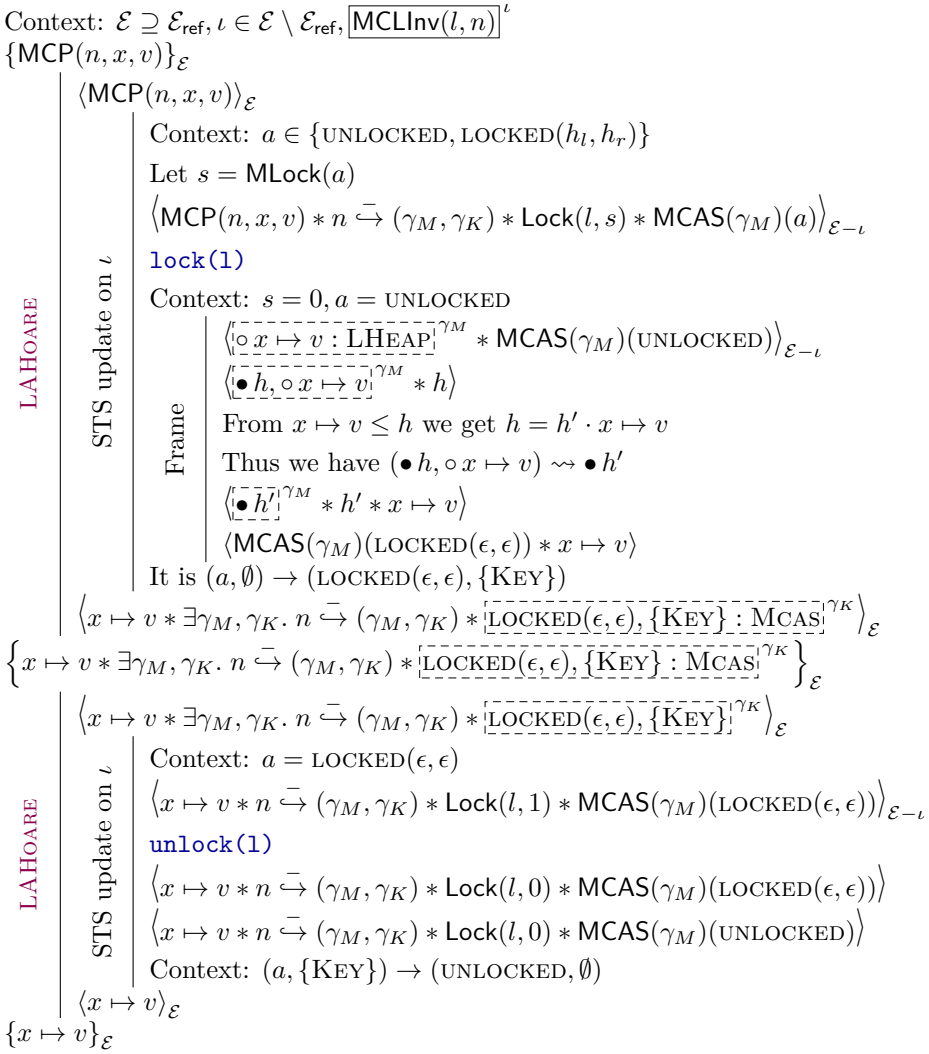


Figure 14: Proof of `unmakeMCP`.

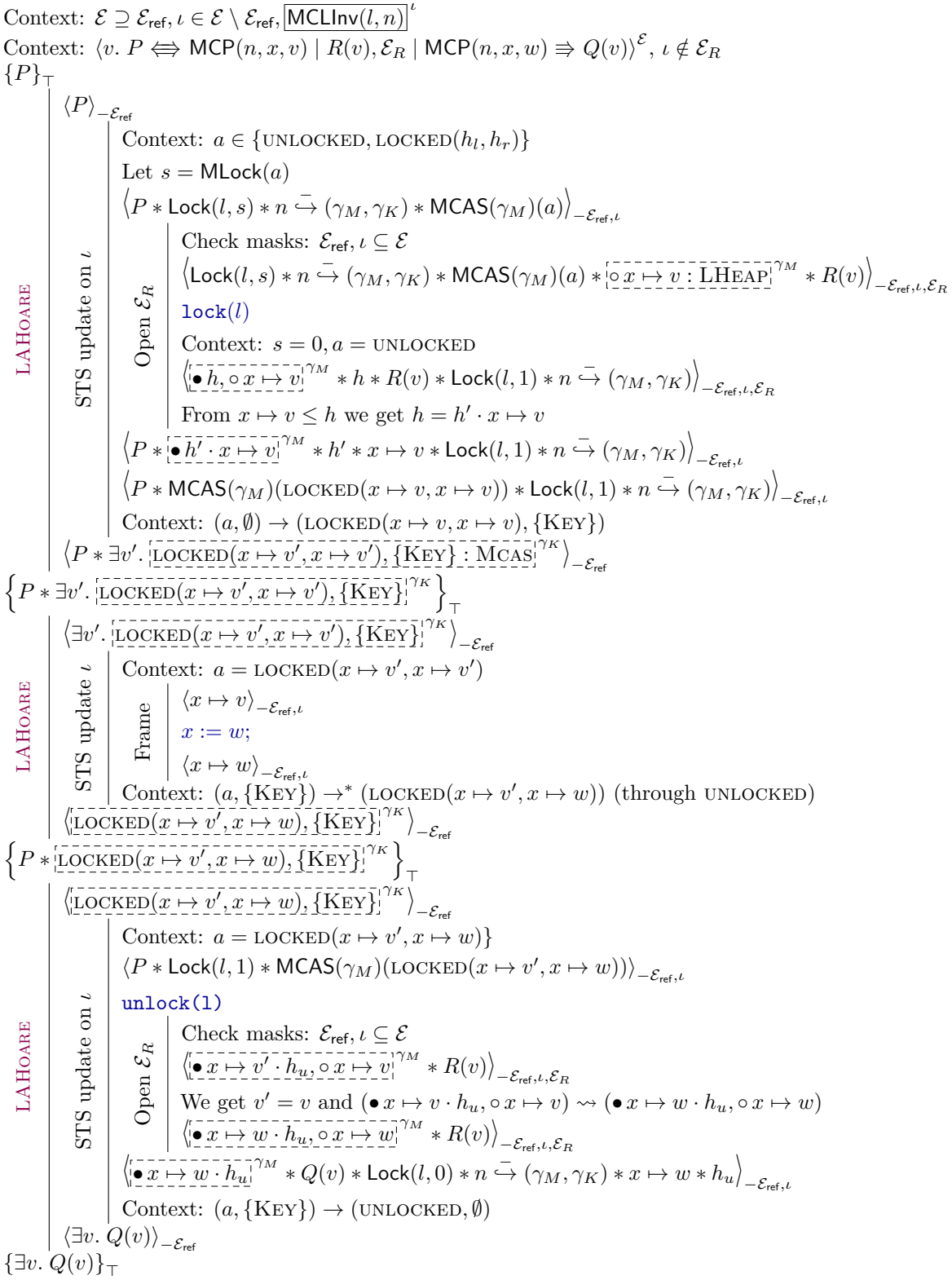


Figure 15: Proof of `write`.

14 Stack with helping

This section shows how to prove the correctness of an elimination stack on top of *logically atomic* references. The code and proof are essentially the same as in iCAP [4].

14.1 Specification

$$
\begin{aligned}
&\exists \mathsf{isStack}, \mathsf{StackCont}, \mathsf{StackPop}.\\
&\quad \forall n, l.\ \mathsf{timeless}(\mathsf{StackCont}(n, l)) \land\\
&\quad \forall n, l, l'.\ \mathsf{StackCont}(n, l) * \mathsf{StackCont}(n, l') \Rightarrow \mathsf{False} \land\\
&\quad \forall n, l, x.\ \mathsf{StackPop}(n, l, x) \Leftrightarrow\\
&\qquad (x = \mathsf{null} \land l = \mathsf{nil} \land \mathsf{StackCont}(n, \mathsf{nil})) \lor (\exists l'.\ l = x :: l' \land \mathsf{StackCont}(n, l')) \land\\
&\quad \forall \mathcal{E}.\ \mathcal{E} \supseteq \mathcal{E}_{\mathsf{ref}} \land \mathsf{infinite}(\mathcal{E} \setminus \mathcal{E}_{\mathsf{ref}}) \Rightarrow \{\mathsf{True}\}\ \texttt{newStack}()\ \{s.\ \exists n.\ \Box \mathsf{isStack}(s, \mathcal{E}, n) * \mathsf{StackCont}(n, \mathsf{nil})\} \land\\
&\quad \forall P, Q, \mathcal{E}, n, s, x.\ \mathsf{isStack}(s, \mathcal{E}, n) \Rightarrow \langle l.\ \mathsf{StackCont}(n, l)\rangle\ \texttt{push}(s, x)\ \langle \mathsf{StackCont}(n, x :: l)\rangle^{\mathcal{E}} \land\\
&\quad \forall P, Q, \mathcal{E}, n, s.\ \mathsf{isStack}(s, \mathcal{E}, n) \Rightarrow \langle l.\ \mathsf{StackCont}(n, l)\rangle\ \texttt{pop}(s)\ \langle x.\ \mathsf{StackPop}(n, l, x)\rangle^{\mathcal{E}}
\end{aligned}
$$

14.2 Code

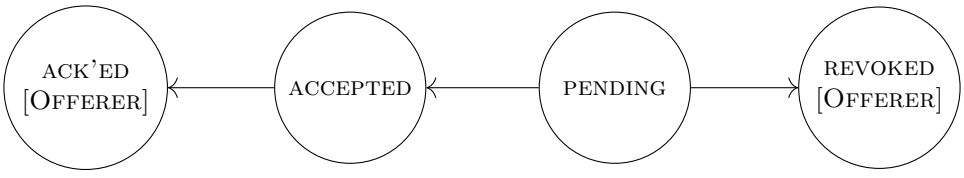
$$\texttt{newStack} := \lambda\_.\ \mathsf{ref}\ (\mathsf{head} \mapsto \mathsf{null}, \mathsf{offer} \mapsto \mathsf{null})$$

```
push := rec loop(s, x).
        let h_n = ref (next ↦ null, value ↦ x) in
        let h_o = !s.head in
        h_n.next := h_o;
        let b = cas(s.head, h_o, h_n) in
        if b then () else
        let o = ref (state ↦ 0, value ↦ x) in
        s.offer := o;
        s.offer := null;
        let b = cas(o.state, 0, 2) in
        if b then loop(s, x) else skip
```

```
pop := rec loop(s).
       let h_o = !s.head in
       if h_o == null then null else
       let h_n = !h_o.next in
       let b = cas(s.head, h_o, h_n) in
       if b then !h_o.value else
       let o = !s.offer in
       skip;
       if o ≠ null then
         let b = cas(o.state, 0, 1) in
         if b then !o.value else loop(s)
       else loop(s)
```

14.3 Predicate definitions, invariants

The implementation of the stack will use helping, and the complex part of defining the invariants is defining the protocol used for helping. That protocol has 4 possible states: *pending* (the offer for being helped has just been created), *revoked* (the offer has been revoked), *accepted* (the offer has been accepted), *ack'ed* (accepting the offer has been acknowledged by the offerer). The transitions to revoke or acknowledge an offer must only be made by the offerer, hence their target states require a token that the offerer will hold. The protocol (an STS with tokens) now looks as follows:



This defines the STS $\mathcal{S}$. Let $\textsc{Offer} \triangleq \textsc{Sts}_{\mathcal{S}}$ be the monoid describing the STS.

The entire protocol is parameterized over the client's P and Q as well as the location of the offer.

$$\mathsf{Offer}(o, P, Q)(\textsc{pending}) \triangleq o.\mathsf{state} \mapsto 0 * P$$
$$\mathsf{Offer}(o, P, Q)(\textsc{revoked}) \triangleq o.\mathsf{state} \mapsto 2$$
$$\mathsf{Offer}(o, P, Q)(\textsc{accepted}) \triangleq o.\mathsf{state} \mapsto 1 * Q$$
$$\mathsf{Offer}(o, P, Q)(\textsc{ack'ed}) \triangleq o.\mathsf{state} \mapsto 1$$

This gives rise to an invariant $\mathsf{OfferInv}(o, \gamma_o, P, Q) \triangleq \mathsf{STSInv}(\mathcal{S}, \mathsf{Offer}(o, P, Q), \gamma_o)$.

We also introduce two predicates to assert existence and ownership of an offer, respectively:

$$\mathsf{isOffer}(n, \mathcal{E}, \mathcal{E}_o, o) \triangleq \exists P, Q, x, \iota_o, \gamma_o.\ \iota_o \in \mathcal{E}_o * \boxed{\mathsf{OfferInv}(o, \gamma_o, P, Q)}^{\iota_o} * o.\mathsf{value} \overset{-}{\mapsto} x * \langle l.\ P \Longleftrightarrow \mathsf{StackCont}(n, l) \mid \mathsf{StackCont}(n, x :: l) \Rrightarrow Q \rangle^{\mathcal{E}}$$

$$\mathsf{myOffer}(\mathcal{E}_o, o, P, Q, x) \triangleq \exists \iota_o, \gamma_o.\ \iota_o \in \mathcal{E}_o * \boxed{\mathsf{OfferInv}(o, \gamma_o, P, Q)}^{\iota_o} * \boxed{\textsc{pending}, \{\textsc{Offerer}\} : \textsc{Offer}}^{\gamma_o} * o.\mathsf{value} \overset{-}{\mapsto} x$$

Note that $\mathsf{isOffer}$ is duplicable.

We need a recursive duplicable predicate to describe the structure of a linked list.

$$\mathsf{isList}(h, \mathsf{nil}) \triangleq h = \mathsf{null}$$
$$\mathsf{isList}(h, v :: l) \triangleq \exists h'.\ h.\mathsf{value} \overset{-}{\mapsto} v * x.\mathsf{next} \overset{-}{\mapsto} h' * \triangleright \mathsf{isList}(h', l)$$

Finally, we can give the main stack invariant, and the interpretation of the abstract predicates. To manage the stack state, we introduce a monoid $\textsc{Stack} \triangleq \textsc{Ex}(\text{list val})$.

$$\mathsf{StackInv}(\gamma_s, \mathcal{E}, \mathcal{E}_o, s) \triangleq \exists h, o, l.\ s.\mathsf{head} \mapsto h * \mathsf{isList}(h, l) * \boxed{\bullet\, l : \textsc{Auth}(\textsc{Stack})}^{\gamma_s} * s.\mathsf{offer} \mapsto o * (o = \mathsf{null} \vee \mathsf{isOffer}(\gamma_s, \mathcal{E}, \mathcal{E}_o, o))$$

$$\mathsf{isStack}(s, \mathcal{E}, n) \triangleq \mathcal{E}_{\mathsf{ref}} \subseteq \mathcal{E} \wedge \exists \iota_s, \mathcal{E}_o.\ \iota_s \in \mathcal{E} \setminus (\mathcal{E}_o \uplus \mathcal{E}_{\mathsf{ref}}) \wedge \mathcal{E}_o \subseteq \mathcal{E} \wedge \mathsf{infinite}(\mathcal{E}_o) \wedge \boxed{\mathsf{StackInv}(n, \mathcal{E}, \mathcal{E}_o, s)}^{\iota_s}$$

$$\mathsf{StackCont}(n, l) \triangleq \boxed{\circ\, l : \textsc{Auth}(\textsc{Stack})}^{n}$$

14.4 Proof of newStack

Context: $\mathcal{E} \supseteq \mathcal{E}_{\mathsf{ref}}$, $\mathsf{infinite}(\mathcal{E} \setminus \mathcal{E}_{\mathsf{ref}})$
Have: $\exists \mathcal{E}_o, \mathcal{E}_s.\ \mathcal{E} \setminus \mathcal{E}_{\mathsf{ref}} = \mathcal{E}_o \uplus \mathcal{E}_s \wedge \mathsf{infinite}(\mathcal{E}_o) \wedge \mathsf{infinite}(\mathcal{E}_s)$
$\{\mathsf{True}\}_{\top}$
$\mathsf{ref}\ (\mathsf{head} \mapsto \mathsf{null}, \mathsf{offer} \mapsto \mathsf{null})$
$\{s.\ s.\mathsf{head} \mapsto \mathsf{null} * s.\mathsf{offer} \mapsto \mathsf{null}\}_{\top}$
$\left\{s.\ s.\mathsf{head} \mapsto \mathsf{null} * s.\mathsf{offer} \mapsto \mathsf{null} * \exists n.\ \boxed{(\bullet\, \mathsf{nil}, \circ\, \mathsf{nil}) : \textsc{Auth}(\textsc{Stack})}^{n}\right\}_{\top}$ by NewGhost
$\{s.\ \mathsf{StackInv}(n, \mathcal{E}, \mathcal{E}_o, s) * \mathsf{StackCont}(n, \mathsf{nil})\}_{\top}$
$\{s.\ \Box \mathsf{isStack}(s, \mathcal{E}, n) * \mathsf{StackCont}(n, \mathsf{nil})\}_{\top}$ by NewInv from $\mathcal{E}_s$

14.5 Proof of push

Context: $\mathcal{E}_{\mathsf{ref}} \subseteq \mathcal{E}, \iota_s \in \mathcal{E} \setminus (\mathcal{E}_o \uplus \mathcal{E}_{\mathsf{ref}}), \mathcal{E}_o \subseteq \mathcal{E}, \mathsf{infinite}(\mathcal{E}_o), \boxed{\mathsf{StackInv}(\gamma_s, \mathcal{E}, \iota_s, s)}^{\iota_s}$
Context: $\langle l.\ P \Longleftrightarrow \mathsf{StackCont}(\gamma_s, l) \mid R(l), \mathcal{E}_R \mid \mathsf{StackCont}(\gamma_s, x :: l) \Rrightarrow Q \rangle^{\mathcal{E}}$
Context: $\forall s', x'.\ \{\triangleright P\}\ \mathit{loop}(s', x')\ \{Q\}$

$\{P\}_\top$

let $h_n = \mathsf{ref}\,(\mathsf{next} \mapsto \mathsf{null}, \mathsf{value} \mapsto x)$ **in**

$\{P * h_n.\mathsf{next} \mapsto \mathsf{null} * h_n.\mathsf{value} \mapsto x\}_\top$

LAHOARE
- $\langle \mathsf{True} \rangle_{-\mathcal{E}_{\mathsf{ref}}}$
- Open ι_s
  - $\langle \triangleright \mathsf{StackInv}(\gamma_s, \mathcal{E}, \iota_s, s) \rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$
  - **let** $h_o = !s.\mathsf{head}$ **in**
  - $\langle \triangleright \mathsf{StackInv}(\gamma_s, \mathcal{E}, \iota_s, s) \rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$

$\{P * h_n.\mathsf{next} \mapsto \mathsf{null} * h_n.\mathsf{value} \mapsto x\}_\top$

$h_n.\mathsf{next} := h_o;$ using LAHOARE

$\{P * h_n.\mathsf{next} \mapsto h_o * h_n.\mathsf{value} \mapsto x\}_\top$

LAHOARE, Open Invariant ι_s
- Frame $\exists o.\ s.\mathsf{offer} \mapsto o * (o = \mathsf{null} \lor \triangleright \mathsf{isOffer}(\gamma_s, \mathcal{E}, \mathcal{E}_o, o))$
- $\left\langle P * h_n.\mathsf{next} \mapsto h_o * h_n.\mathsf{value} \mapsto x * s.\mathsf{head} \mapsto h * \mathsf{isList}(h, l) * \boxed{\bullet\, l}^{\gamma_s} \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$
- **let** $b = \mathsf{cas}(s.\mathsf{head}, h_o, h_n)$ **in**
- Left
  - Context: $b = \mathsf{true}, h = h_o$
  - $\langle P * h_n.\mathsf{next} \mapsto h_o * h_n.\mathsf{value} \mapsto x * s.\mathsf{head} \mapsto h_n * \mathsf{isList}(h_o, l) \rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$
  - $\left\langle P * s.\mathsf{head} \mapsto h_n * \mathsf{isList}(h_n, x :: l) * \boxed{\bullet\, l}^{\gamma_s} \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$
  - Open $\mathcal{E}_R$
    - Check masks: $\mathcal{E}_{\mathsf{ref}}, \iota_s \subseteq \mathcal{E}$
    - $\left\langle \boxed{\bullet\, l, \circ\, l}^{\gamma_s} * R(l) \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s, \mathcal{E}_R}$
    - We have $\bullet\, l, \circ\, l \rightsquigarrow \bullet\, x :: l, \circ\, x :: l$
    - $\left\langle \boxed{\bullet\, x :: l, \circ\, x :: l}^{\gamma_s} * R(l) \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s, \mathcal{E}_R}$
  - $\left\langle \boxed{\bullet\, x :: l}^{\gamma_s} * Q * s.\mathsf{head} \mapsto h_n * \mathsf{isList}(h_n, x :: l) \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$
  - Establish StackInv with $h \triangleq h_n, l \triangleq x :: l$

$\{b = \mathsf{true} * Q \lor b = \mathsf{false} * P\}_\top$

$\{b = \mathsf{true} * Q \lor b = \mathsf{false} * P\}_\top$

if b **then**

$\{b = \mathsf{true} * Q\}_\top$

$()$

$\{Q\}_\top$

else

$\{P\}_\top$

let $o = \mathsf{ref}\ (\mathsf{state} \mapsto 0, \mathsf{value} \mapsto x)$ **in**

$\{P * o.\mathsf{state} \mapsto 0 * o.\mathsf{value} \mapsto x\}_\top$

New monoid γ_o, new invariant $\iota_o \in \mathcal{E}_o$

$\left\{o.\mathsf{value} \mapsto x * \boxed{\mathsf{OfferInv}(\gamma_s, \mathcal{E}, o, \gamma_o, P, Q, x)}^{\iota_o} * \boxed{\textsc{pending}, \{\textsc{Offerer}\}}^{\gamma_o}\right\}_\top$

Context: $\boxed{\mathsf{OfferInv}(\gamma_s, \mathcal{E}, o, \gamma_o, P, Q, x)}^{\iota_o}$

$\left\{o.\mathsf{value} \mapsto x * \boxed{\textsc{pending}, \{\textsc{Offerer}\} : \textsc{Offer}}^{\gamma_o}\right\}_\top$

LaHoare

$\left\langle o.\mathsf{value} \mapsto x * \boxed{\textsc{pending}, \{\textsc{Offerer}\} : \textsc{Offer}}^{\gamma_o}\right\rangle_{-\mathcal{E}_{\mathsf{ref}}}$

Open ι_s

$\langle \mathsf{isOffer}(\gamma_s, \mathcal{E}, \mathcal{E}_o, o) * \triangleright \mathsf{StackInv}(\gamma_s, \mathcal{E}, \iota_s, s)\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$

$s.\mathsf{offer} := o;$

$\left\langle s.\mathsf{head} \mapsto h * \mathsf{isList}(h, l) * \boxed{\bullet\ l}^{\gamma_s} * s.\mathsf{offer} \mapsto o * \mathsf{isOffer}(\gamma_s, \mathcal{E}, \mathcal{E}_o, o)\right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$

$\{\mathsf{myOffer}(\gamma_s, \mathcal{E}, \mathcal{E}_o, o, P, Q, x)\}_\top$

LaHoare

$\langle \mathsf{True}\rangle_{-\mathcal{E}_{\mathsf{ref}}}$

Open ι_s

$\langle \triangleright \mathsf{StackInv}(\gamma_s, \mathcal{E}, \iota_s, s)\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$

$s.\mathsf{offer} := \mathsf{null};$

$\langle \triangleright \mathsf{StackInv}(\gamma_s, \mathcal{E}, \iota_s, s)\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$

$\{\mathsf{myOffer}(\gamma_s, \mathcal{E}, \mathcal{E}_o, o, P, Q, x)\}_\top$

LaHoare

$\left\langle \boxed{\textsc{pending}, \{\textsc{Offerer}\}}^{\gamma_o}\right\rangle_{-\mathcal{E}_{\mathsf{ref}}}$

STS update ι_o

Context: $a \in \{\textsc{pending}, \textsc{accepted}\}, T = \{\textsc{Offerer}\}$

$\langle \triangleright \mathsf{Offer}(\gamma_s, \mathcal{E}, o, P, Q, x)(a)\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o}$

let $b = \mathsf{cas}(o.\mathsf{state}, 0, 2)$ **in**

Left

Context: $b = \mathsf{true}, a = \textsc{pending}$

$\langle \triangleright P * \mathsf{Offer}(\gamma_s, \mathcal{E}, o, P, Q, x)(\textsc{revoked})\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o}$

We have $(\textsc{pending}, \{\textsc{Offerer}\}) \rightarrow (\textsc{revoked}, \emptyset)$

Right

Context: $b = \mathsf{false}, a = \textsc{accepted}$

$\langle \triangleright Q * \mathsf{Offer}(\gamma_s, \mathcal{E}, o, P, Q, x)(\textsc{ack'ed})\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o}$

We have $(\textsc{accepted}, \{\textsc{Offerer}\}) \rightarrow (\textsc{ack'ed}, \emptyset)$

$\{b = \mathsf{true} * P \lor b = \mathsf{false} * \triangleright Q\}_\top$

if b **then**

$\{b = \mathsf{true} * P\}_\top$

$loop(s, x)$

$\{Q\}_\top$

else

$\{b = \mathsf{false} * \triangleright Q\}_\top$

skip

$\{Q\}_\top$

$\{Q\}_\top$

14.6 Proof of pop

Context: $\mathcal{E}_{\mathsf{ref}} \subseteq \mathcal{E}, \iota_s \in \mathcal{E} \setminus (\mathcal{E}_o \uplus \mathcal{E}_{\mathsf{ref}}), \mathcal{E}_o \subseteq \mathcal{E}, \mathsf{infinite}(\mathcal{E}_o), \boxed{\mathsf{StackInv}(\gamma_s, \mathcal{E}, \iota_s, s)}^{\iota_s}$
Context: $\langle l.\ P \Longleftrightarrow \mathsf{StackCont}(\gamma_s, l) \mid R(l), \mathcal{E}_R \mid x.\ \mathsf{StackPop}(\gamma_s, l, x) \Rrightarrow Q(x) \rangle^{\mathcal{E}}$
Context: $\forall s'.\ \{\triangleright P\}\ \mathit{loop}(s')\ \{x.\ Q(x)\}$
$\{P\}_\top$

LAHoare, Open Invariant ι_s:
- $\left\langle P * s.\mathsf{head} \mapsto h * \mathsf{isList}(h, l) * \boxed{\bullet\, l : \textsc{Auth}(\textsc{Stack})}^{\gamma_s} \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$
- **let** $h_o = !s.\mathsf{head}$ **in**
- Case distinction: $h_o = \mathsf{null} \vee h_o \neq \mathsf{null}$

  Left:
  - Context $h = \mathsf{null}, l = \mathsf{nil}$
  - $\left\langle P * \boxed{\bullet\, \mathsf{nil}}^{\gamma_s} \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$

    Open $\mathcal{E}_R$:
    - Check masks: $\mathcal{E}_{\mathsf{ref}}, \iota_s \subseteq \mathcal{E}$
    - $\left\langle \boxed{\bullet\, \mathsf{nil}, \circ\, \mathsf{nil}}^{\gamma_s} * R(\mathsf{nil}) \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s, \mathcal{E}_R}$
    - $\left\langle \boxed{\bullet\, \mathsf{nil}}^{\gamma_s} * \mathsf{StackPop}(n, \mathsf{nil}, \mathsf{null}) * R(\mathsf{nil}) \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s, \mathcal{E}_R}$
  - $\left\langle Q(\mathsf{null}) * \boxed{\bullet\, \mathsf{nil}}^{\gamma_s} \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$

  Right:
  - $\left\langle P * \mathsf{isList}(h_o, l) * s.\mathsf{head} \mapsto h * \mathsf{isList}(h, l) * \boxed{\bullet\, l}^{\gamma_s} \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$

$\{h_o = \mathsf{null} * Q(\mathsf{null}) \vee h_o \neq \mathsf{null} * P * \exists l.\ \mathsf{isList}(h_o, l')\}_\top$
if $h_o == \mathsf{null}$ **then**
- $\{h_o = \mathsf{null} * Q(\mathsf{null})\}_\top$
- null
- $\{x.\ Q(x)\}_\top$

else
- $\{h_o \neq \mathsf{null} * P * \exists l'.\ \mathsf{isList}(h_o, l')\}_\top$
- $\left\{P * \exists x, p, l'.\ h_o.\mathsf{value} \overset{-}{\mapsto} x * h_o.\mathsf{next} \overset{-}{\mapsto} p * \mathsf{isList}(p, l')\right\}_\top$
- **let** $h_n = !h_o.\mathsf{next}$ **in** using LAHoare
- $\left\{P * h_o.\mathsf{next} \overset{-}{\mapsto} h_n\right\}_\top$

  LAHoare, Open Invariant ι_s:
  - $\left\langle P * h_o.\mathsf{next} \overset{-}{\mapsto} h_n * s.\mathsf{head} \mapsto h * \mathsf{isList}(h, l) * \boxed{\bullet\, l}^{\gamma_s} \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$
  - **let** $b = \mathsf{cas}(s.\mathsf{head}, h_o, h_n)$ **in**

    Left:
    - Context $b = \mathsf{true}, h = h_o, l = x :: l'$
    - $\left\langle P * \boxed{\bullet\, l}^{\gamma_s} \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$

      Open $\mathcal{E}_R$:
      - Check masks: $\mathcal{E}_{\mathsf{ref}}, \iota_s \subseteq \mathcal{E}$
      - $\left\langle \boxed{\bullet\, l, \circ\, l}^{\gamma_s} * R(l) \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s, \mathcal{E}_R}$
      - We have $(\bullet\, l, \circ\, l) \rightsquigarrow (\bullet\, l', \circ\, l')$
      - $\left\langle \boxed{\bullet\, l'}^{\gamma_s} * \mathsf{StackPop}(\gamma_s, l, x) * R(l) \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s, \mathcal{E}_R}$
    - $\left\langle Q(x) * \boxed{\bullet\, l'}^{\gamma_s} \right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$
  - $\left\{b = \mathsf{true} * h_o.\mathsf{value} \overset{-}{\mapsto} x * Q(x) \vee b = \mathsf{false} * P\right\}_\top$
  - **if** b **then**
    - $\left\{b = \mathsf{true} * h_o.\mathsf{value} \overset{-}{\mapsto} x * Q(x)\right\}_\top$
    - $!h_o.\mathsf{value}$ using LAHoare
    - $\{x.\ Q(x)\}_\top$

else

$\{b = \mathsf{false} * P\}_\top$

LAHOARE | Open ι_s | $\langle \triangleright \mathsf{StackInv}(\gamma_s, \mathcal{E}, \iota_s, s)\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$

let $o = !s.\mathsf{offer}$ **in**

$\langle \triangleright \mathsf{StackInv}(\gamma_s, \mathcal{E}, \iota_s, s) * (o = \mathsf{null} \vee \triangleright \mathsf{isOffer}(\gamma_s, \mathcal{E}, \mathcal{E}_o, o))\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_s}$

$\{P * (o = \mathsf{null} \vee \triangleright \mathsf{isOffer}(\gamma_s, \mathcal{E}, \mathcal{E}_o, o))\}_\top$

skip;

$\{P * (o = \mathsf{null} \vee \mathsf{isOffer}(\gamma_s, \mathcal{E}, \mathcal{E}_o, o))\}_\top$

if $o \neq \mathsf{null}$ **then**

Context $\iota_o \in \mathcal{E}_o, \boxed{\mathsf{OfferInv}(n, \mathcal{E}, o, \gamma_o, P, Q, x)}^{\iota_o}$

Context: $\langle l.\ P \Longleftrightarrow \mathsf{StackCont}(n, l) \mid \mathsf{StackCont}(n, x :: l) \Rrightarrow Q\rangle^{\mathcal{E}}$

$\left\{P * o.\mathsf{value} \xmapsto{-} x\right\}_\top$

LAHOARE, STS update ι_o | Left

Context: $a \in \{\text{PENDING}, \text{REVOKED}, \text{ACCEPTED}, \text{ACK'ED}\}, T = \emptyset$

$\langle \triangleright \mathsf{Offer}(\gamma_s, \mathcal{E}, o, P_o, Q_o, x_o)(a)\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o}$

let $b = \mathsf{cas}(o.\mathsf{state}, 0, 1)$ **in**

Context $b = \mathsf{true}, a = \text{PENDING}$

$\langle P * P_o * o.\mathsf{state} \mapsto 1\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o}$

Open ι_s | $\left\langle P * P_o * \boxed{\bullet\, l}^{\gamma_s}\right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o, \iota_s}$

Open $\mathcal{E}_{R_o}$ | Check masks: $\mathcal{E}_{\mathsf{ref}}, \iota_o, \iota_s \subseteq \mathcal{E}$

$\left\langle P * \boxed{\bullet\, l, \circ\, l}^{\gamma_s} * R_o(l)\right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o, \iota_s, \mathcal{E}_{R_o}}$

We have $(\bullet\, l, \circ\, l) \rightsquigarrow (\bullet\, x_o :: l, \circ\, x_o :: l)$

$\left\langle P * \boxed{\bullet\, x_o :: l, \circ\, x_o :: l}^{\gamma_s} * R_o(l)\right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o, \iota_s, \mathcal{E}_{R_o}}$

$\left\langle P * \boxed{\bullet\, x_o :: l}^{\gamma_s} * Q_o\right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o, \iota_s}$

Open $\mathcal{E}_R$ | Check masks: $\mathcal{E}_{\mathsf{ref}}, \iota_o, \iota_s \subseteq \mathcal{E}$

$\left\langle \boxed{\bullet\, x_o :: l, \circ\, x_o :: l}^{\gamma_s} * R(x_o :: l) * Q_o\right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o, \iota_s, \mathcal{E}_R}$

We have $(\bullet\, x_o :: l, \circ\, x_o :: l) \rightsquigarrow (\bullet\, l, \circ\, l)$

$\left\langle \boxed{\bullet\, l}^{\gamma_s} * \mathsf{StackPop}(\gamma_S, x_o :: l, x_o) * R(x_o :: l) * Q_o\right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o, \iota_s, \mathcal{E}_R}$

$\left\langle \boxed{\bullet\, l}^{\gamma_s} * Q(x_o) * Q_o\right\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o, \iota_s}$

$\langle \mathsf{Offer}(\gamma_s, \mathcal{E}, o, P_o, Q_o, x_o)(\text{ACCEPTED}) * Q(x_o)\rangle_{-\mathcal{E}_{\mathsf{ref}}, \iota_o}$

We have $(\text{PENDING}, \emptyset) \to (\text{ACCEPTED}, \emptyset)$

$\left\{(b = \mathsf{true} * Q(x_o) \vee b = \mathsf{false} * P) * o.\mathsf{value} \xmapsto{-} x_o\right\}_\top$

if b **then**

$\left\{b = \mathsf{true} * Q(x_o) * o.\mathsf{value} \xmapsto{-} x_o\right\}_\top$

$!o.\mathsf{value}$ using LAHOARE

$\{x.\ Q(x)\}_\top$

else

$\{b = \mathsf{false} * P\}_\top$

$loop(s)$

$\{x.\ Q(x)\}_\top$

else

$\{b = \mathsf{false} * P\}_\top$

$loop(s)$

$\{x.\ Q(x)\}_\top$

$\{x.\ Q(x)\}_\top$

References

[1] P. America and J. Rutten. Solving reflexive domain equations in a category of complete metric spaces. *J. Comput. Syst. Sci.*, 39(3):343–375, 1989.

[2] P. da Rocha Pinto, T. Dinsdale-Young, and P. Gardner. TaDA: A logic for time and data abstraction. In *ECOOP*, 2014.

[3] N. R. Krishnaswami, A. Turon, D. Dreyer, and D. Garg. Superficially substructural types. In *ICFP*, 2012.

[4] K. Svendsen and L. Birkedal. Impredicative concurrent abstract predicates. In *ESOP*, 2014.

[5] A. Turon, D. Dreyer, and L. Birkedal. Unifying refinement and Hoare-style reasoning in a logic for higher-order concurrency. In *ICFP*, 2013.