# Mechanized Relational Verification of Concurrent Programs with Continuations: Technical Appendix

Amin Timany Lars Birkedal

July 13, 2018

## Contents

# 1 Iris resources

Resources in Iris are described using a kind of partial commutative monoids, and the user of the logic can introduce new monoids.[1] For instance, in the case of finite partial maps, the partiality comes from the fact that disjoint union of finite maps is partial. Undefinedness is treated by means of a validity predicate $\checkmark : \mathcal{M} \to \mathit{iProp}$, which expresses which elements of the monoid $\mathcal{M}$ are valid/defined.

We write $\boxed{a : \mathcal{M}}^{\gamma}$ to assert that a monoid instance named $\gamma$, of type $\mathcal{M}$, has contents $a$. Often, we disregard the type if it is obvious from the context and simply write $\boxed{a}^{\gamma}$. We think of this assertion as a ghost variable $\gamma$ with contents $a$.

$$\begin{array}{lll} \textsc{Ghost-Alloc} & \textsc{Own-Valid} & \textsc{Sharing} \\ \checkmark a \vdash \Rrightarrow \exists \gamma.\, \boxed{a}^{\gamma} & \boxed{a}^{\gamma} \vdash \checkmark(a) & \boxed{a}^{\gamma} * \boxed{b}^{\gamma} \dashv\vdash \boxed{a \cdot b}^{\gamma} \end{array}$$

**Some Useful Monoids** Here, we describe a few monoids which are particularly useful and which we will use in the sequel. We do not give the full definitions of the monoids (those can be found in [Krebbers et al., 2017]), but focus instead on the properties which the elements of the monoids satisfy, shown in Figure 1. These rules stated are only for monoids that we use in this work and not in Iris in its generality. For instance, in the rule Auth-Included, $\subseteq$ is a set relation and is defined for finite set and finite partial function monoids and not in general.

Figure 1 depicts the rules necessary for allocating and updating finite set monoids, $\mathsf{finset}(A)$, and finite partial function monoids, $A \xrightharpoonup{\text{fin}} M$. For $\mathsf{finset}(A)$, the monoid operation $x \cdot y$ is union. The notation $a \mapsto b : A \xrightharpoonup{\text{fin}} B \triangleq \{(a, b)\}$ is a singleton finite partial function.

The constructs $\bullet$ and $\circ$ are constructors of the so-called authoritative monoid $\textsc{Auth}(M)$. We read $\bullet\, a$ as "*full* $a$" and $\circ\, a$ as "*fragment* $a$". We use the authoritative monoid to distribute ownership of fragments of a resource. The intuition is that $\bullet\, a$ is the authoritative knowledge of the full resource, think of it as being kept track of in a central location (see rule Auth-Included). The fragments, $\circ\, a$, can be shared (rule Frag-distributes) while the full part (the central location) should always remain unique (rule Full-Exclusive).

In addition to authoritative monoids, we also use the agreement monoid $\textsc{Ag}(M)$ and exclusive monoid $\textsc{Ex}(M)$. As the name suggests, the operation of the agreement monoid guarantees that $\mathsf{ag}(a) \cdot \mathsf{ag}(b)$ is invalid whenever $a \neq b$ (and otherwise it is idempotent; see rules Agree and Agreement-Valid). From the rule Agree it follows that the ownership of elements of $\textsc{Ag}(M)$ is duplicable.[2]

$$\boxed{\mathsf{ag}(a)}^{\gamma} \dashv\vdash \boxed{\mathsf{ag}(a) \cdot \mathsf{ag}(a)}^{\gamma} \dashv\vdash \boxed{\mathsf{ag}(a)}^{\gamma} * \boxed{\mathsf{ag}(a)}^{\gamma}$$

The operation of the exclusive monoid never results in a valid element (rule Exclusive), enforcing that there can only be one instance of it owned. We can define the resource

[1]Technically these are resource algebras (RAs) which are similar to monoids. In particular, RAs need not necessarily have a unit element. Furthermore, RAs are step-indexed. We ignore these details here as they are not directly relevant to our discussions. For more detail see Jung et al. [2016], Krebbers et al. [2017].

[2]Indeed it can be shown that elements of $\textsc{Ag}(M)$ are also persistent.

Auth-Included
$\bullet\, a \cdot \circ\, b \vdash b \subseteq a$

Fpfn-valid
$\checkmark(a) \dashv\vdash \forall x \in \mathrm{dom}(a).\ \checkmark(a(x))$

Agreement-Valid
$\checkmark(\mathsf{ag}(a) \cdot \mathsf{ag}(b)) \dashv\vdash a = b$

Exclusive
$\not\checkmark(\mathsf{ex}(a) \cdot b)$

Frag-distributes
$\circ\, a \cdot \circ\, b = \circ\,(a \cdot b)$

Full-Exclusive
$\not\checkmark(\bullet\, a \cdot \bullet\, b)$

Auth-Alloc-Finset
$$\frac{h \cap a = \emptyset}{\boxed{\bullet\, h}^{\gamma} \Rrightarrow\!\!* \boxed{\bullet\,(h \uplus a) \cdot \circ\, a}^{\gamma}}$$

Auth-Alloc-Fpfn
$$\frac{\mathrm{dom}(h) \cap \mathrm{dom}(a) = \emptyset}{\boxed{\bullet\, h}^{\gamma} \Rrightarrow\!\!* \boxed{\bullet\,(h \uplus a) \cdot \circ\, a}^{\gamma}}$$

Agree
$\mathsf{ag}(a) \cdot \mathsf{ag}(a) = \mathsf{ag}(a)$

Fpfn-operation-success
$$(a \cdot b)(x) = \begin{cases} a(x) & \text{if } x \in \mathrm{dom}(a) \wedge x \notin \mathrm{dom}(b) \\ a(x) \cdot b(x) & \text{if } x \in \mathrm{dom}(a) \cap \mathrm{dom}(b) \\ b(x) & \text{if } x \in \mathrm{dom}(b) \wedge x \notin \mathrm{dom}(a) \end{cases}$$

Auth-Update-Fpfn-Excl
$$\boxed{\bullet\,(h \uplus (\ell \mapsto \mathsf{ex}(v_1))) \cdot \circ\, \ell \mapsto \mathsf{ex}(v_1)}^{\gamma} \Rrightarrow\!\!* \boxed{\bullet\,(h \uplus (\ell \mapsto \mathsf{ex}(v_2))) \cdot \circ\, \ell \mapsto \mathsf{ex}(v_2)}^{\gamma}$$

Figure 1: Rules for selected monoid resources in Iris

for keeping track of the physical heap of the programming language, Heap. This is the canonical example of a monoid.

$$\text{Heap} \triangleq \text{Auth}(\mathit{Loc} \xrightharpoonup{\text{fin}} (\text{Ex}(\mathit{Val})))$$

Hence, the points-to proposition of the separation logic can be defined as follows.

$$\ell \mapsto_i v \triangleq \boxed{\circ\,[l \mapsto \mathsf{ex}(v)]}^{\gamma_h}$$

Here, $\gamma_h$ is the *globally fixed* monoid name to keep track of the heap of $\mathsf{F}^{\mu,\mathit{ref}}_{\mathit{conc},\mathit{cc}}$. The full part of this monoid is used in the definition of the weakest precondition to allow weakest preconditions to refer to the physical state of the program [Krebbers et al., 2017]. Notice here that Heap is build from nesting Ex in the finite partial functions monoid, which again is nested in the Auth monoid. Therefore, to allocate and update the Heap monoid, we can use Auth-Alloc-Fpfn and Auth-Update-Fpfn-Excl respectively.

# 2 Details of Mechanized Relational Verification of Concurrent Programs with Continuations

## 2.1 Weakest precondition rules

REC-WP
$$\frac{\triangleright \mathsf{wp}\ K[e[\mathsf{rec}\ f(x) = e, v/f, x]]\ \{\Phi\}}{\mathsf{wp}\ K[(\mathsf{rec}\ f(x) = e)\ v]\ \{\Phi\}}$$

TAPP-WP
$$\frac{\triangleright \mathsf{wp}\ K[e]\ \{\Phi\}}{\mathsf{wp}\ K[(\Lambda\, e)\ \_]\ \{\Phi\}}$$

UNFOLD-WP
$$\frac{\triangleright \mathsf{wp}\ K[v]\ \{\Phi\}}{\mathsf{wp}\ K[\mathsf{unfold}\ (\mathsf{fold}\ v)]\ \{\Phi\}}$$

IF-TRUE-WP
$$\frac{\triangleright \mathsf{wp}\ K[e]\ \{\Phi\}}{\mathsf{wp}\ K[\mathsf{if\ true\ then}\ e\ \mathsf{else}\ e']\ \{\Phi\}}$$

IF-FALSE-WP
$$\frac{\triangleright \mathsf{wp}\ K[e']\ \{\Phi\}}{\mathsf{wp}\ K[\mathsf{if\ false\ then}\ e\ \mathsf{else}\ e']\ \{\Phi\}}$$

FST-WP
$$\frac{\triangleright \mathsf{wp}\ K[v]\ \{\Phi\}}{\mathsf{wp}\ K[\pi_1\ (v, w)]\ \{\Phi\}}$$

SND-WP
$$\frac{\triangleright \mathsf{wp}\ K[w]\ \{\Phi\}}{\mathsf{wp}\ K[\pi_2\ (v, w)]\ \{\Phi\}}$$

MATCH-INJ1-WP
$$\frac{\triangleright \mathsf{wp}\ K[e_1[v/x]]\ \{\Phi\}}{\mathsf{wp}\ K[\mathsf{match\ inj}_1\ v\ \mathsf{with\ inj}_1\ x \Rightarrow e_1 \mid \mathsf{inj}_2\ x \Rightarrow e_2\ \mathsf{end}]\ \{\Phi\}}$$

MATCH-INJ2-WP
$$\frac{\triangleright \mathsf{wp}\ K[e_2[v/x]]\ \{\Phi\}}{\mathsf{wp}\ K[\mathsf{match\ inj}_2\ v\ \mathsf{with\ inj}_1\ x \Rightarrow e_1 \mid \mathsf{inj}_2\ x \Rightarrow e_2\ \mathsf{end}]\ \{\Phi\}}$$

ALLOC-WP
$$\frac{\forall \ell.\ \ell \mapsto_i v \twoheadrightarrow \mathsf{wp}\ K[\ell]\ \{\Phi\}}{\mathsf{wp}\ K[\mathsf{ref}(v)]\ \{\Phi\}}$$

LOAD-WP
$$\frac{\ell \mapsto_i v \twoheadrightarrow \mathsf{wp}\ K[v]\ \{\Phi\} \qquad \triangleright \ell \mapsto_i v}{\mathsf{wp}\ K[!\,\ell]\ \{\Phi\}}$$

STORE-WP
$$\frac{\ell \mapsto_i w \twoheadrightarrow \triangleright \mathsf{wp}\ K[()]\ \{\Phi\} \qquad \triangleright \ell \mapsto_i v}{\mathsf{wp}\ K[\ell \leftarrow w]\ \{\Phi\}}$$

CAS-SUC-WP
$$\frac{\ell \mapsto_i w \twoheadrightarrow \triangleright \mathsf{wp}\ K[\mathsf{true}]\ \{\Phi\} \qquad \triangleright \ell \mapsto_i v}{\mathsf{wp}\ K[\mathsf{cas}(\ell, v, w)]\ \{\Phi\}}$$

CAS-FAIL-WP
$$\frac{\ell \mapsto_i v' \twoheadrightarrow \triangleright \mathsf{wp}\ K[\mathsf{false}]\ \{\Phi\} \qquad \triangleright \ell \mapsto_i v' \qquad v \neq v'}{\mathsf{wp}\ K[\mathsf{cas}(\ell, v, w)]\ \{\Phi\}}$$

FORK-WP
$$\frac{\triangleright \mathsf{wp}\ K[()]\ \{\Phi\} \qquad \mathsf{wp}\ e\ \{\_.\ \top\}}{\mathsf{wp}\ K[\mathsf{fork}\ \{e\}]\ \{\Phi\}}$$

CALLCC-WP
$$\frac{\triangleright \mathsf{wp}\ K[e[\mathsf{cont}(K)/x]]\ \{\Phi\}}{\mathsf{wp}\ K[\mathsf{call/cc}\ (x.\, e)]\ \{\Phi\}}$$

THROW-WP
$$\frac{\triangleright \mathsf{wp}\ K'[v]\ \{\Phi\}}{\mathsf{wp}\ K[\mathsf{throw}\ v\ \mathsf{to\ cont}(K')]\ \{\Phi\}}$$

## 2.2 Rules for execution on the specification side

$$\textsc{Rec-Step}\ \frac{j \Mapsto K[(\mathsf{rec}\, f(x) = e)\ v]}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K[e[\mathsf{rec}\, f(x) = e, v/f, x]]}$$

$$\textsc{TApp-Step}\ \frac{j \Mapsto K[(\Lambda\, e)\ \_]}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K[e]}$$

$$\textsc{Unfold-Step}\ \frac{j \Mapsto K[\mathsf{unfold}\ (\mathsf{fold}\, v)]}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K[v]}$$

$$\textsc{If-True-Step}\ \frac{j \Mapsto K[\mathsf{if\ true\ then}\ e\ \mathsf{else}\ e']}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K[e]}$$

$$\textsc{If-False-Step}\ \frac{j \Mapsto K[\mathsf{if\ false\ then}\ e\ \mathsf{else}\ e']}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K[e']}$$

$$\textsc{Fst-Step}\ \frac{j \Mapsto K[\pi_1\,(v, w)]}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K[v]}$$

$$\textsc{Snd-Step}\ \frac{j \Mapsto K[\pi_2\,(v, w)]}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K[w]}$$

$$\textsc{Match-Inj1-Step}\ \frac{j \Mapsto K[\mathsf{match\ inj}_1\, v\ \mathsf{with\ inj}_1\, x \Rightarrow e_1 \mid \mathsf{inj}_2\, x \Rightarrow e_2\ \mathsf{end}]}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K[e_1[v/x]]}$$

$$\textsc{Match-Inj2-Step}\ \frac{j \Mapsto K[\mathsf{match\ inj}_2\, v\ \mathsf{with\ inj}_1\, x \Rightarrow e_1 \mid \mathsf{inj}_2\, x \Rightarrow e_2\ \mathsf{end}]}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K[e_2[v/x]]}$$

$$\textsc{Alloc-Step}\ \frac{j \Mapsto K[\mathsf{ref}(v)]}{\mathrel{|\!\!\Rrightarrow} \exists \ell.\ \ell \mapsto_s v * j \Mapsto K[\ell]}$$

$$\textsc{Load-Step}\ \frac{\ell \mapsto_s v \qquad j \Mapsto K[!\,\ell]}{\mathrel{|\!\!\Rrightarrow} \exists \ell.\ \ell \mapsto_s v * j \Mapsto K[v]}$$

$$\textsc{Store-Step}\ \frac{\ell \mapsto_s v \qquad j \Mapsto K[\ell \leftarrow w]}{\mathrel{|\!\!\Rrightarrow} \ell \mapsto_s w * j \Mapsto K[()]}$$

$$\textsc{CAS-suc-Step}\ \frac{\ell \mapsto_s v \qquad j \Mapsto K[\mathsf{cas}(\ell, v, w)]}{\mathrel{|\!\!\Rrightarrow} \ell \mapsto_s w * j \Mapsto K[\mathsf{true}]}$$

$$\textsc{CAS-fail-Step}\ \frac{\ell \mapsto_s v' \qquad j \Mapsto K[\mathsf{cas}(\ell, v, w)] \qquad v \neq v'}{\mathrel{|\!\!\Rrightarrow} \ell \mapsto_s v' * j \Mapsto K[\mathsf{false}]}$$

$$\textsc{Fork-Step}\ \frac{j \Mapsto K[\mathsf{fork}\ \{e\}]}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K[()] * \exists j'.\ j' \Mapsto e}$$

$$\textsc{Callcc-Step}\ \frac{j \Mapsto K[\mathsf{call/cc}\,(x.\, e)]}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K[e[\mathsf{cont}(K)/x]]}$$

$$\textsc{Throw-Step}\ \frac{j \Mapsto K[\mathsf{throw}\ v\ \mathsf{to}\ K']}{\mathrel{|\!\!\Rrightarrow} j \Mapsto K'[v]}$$

## 2.3 Context-local Weakest precondition rules

$$\textsc{Rec-CLWP}\ \frac{\triangleright \mathsf{clwp}\ e[\mathsf{rec}\, f(x) = e, v/f, x]\ \{\Phi\}}{\mathsf{clwp}\ (\mathsf{rec}\, f(x) = e)\ v\ \{\Phi\}}$$

$$\textsc{TApp-CLWP}\ \frac{\triangleright \mathsf{clwp}\ e\ \{\Phi\}}{\mathsf{clwp}\ (\Lambda\, e)\ \_\ \{\Phi\}}$$

$$\textsc{Unfold-CLWP}\ \frac{\triangleright \mathsf{clwp}\ v\ \{\Phi\}}{\mathsf{clwp}\ \mathsf{unfold}\ (\mathsf{fold}\, v)\ \{\Phi\}}$$

$$\textsc{If-True-CLWP}\ \frac{\triangleright \mathsf{clwp}\ e\ \{\Phi\}}{\mathsf{clwp}\ K[\mathsf{if\ true\ then}\ e\ \mathsf{else}\ e']\ \{\Phi\}}$$

$$\textsc{If-False-CLWP}\ \frac{\triangleright \mathsf{clwp}\ e'\ \{\Phi\}}{\mathsf{clwp}\ \mathsf{if\ false\ then}\ e\ \mathsf{else}\ e'\ \{\Phi\}}$$

$$\textsc{Fst-CLWP}\ \frac{\triangleright \mathsf{clwp}\ v\ \{\Phi\}}{\mathsf{clwp}\ \pi_1\,(v, w)\ \{\Phi\}}$$

$$\textsc{Snd-CLWP}\ \frac{\triangleright \mathsf{clwp}\ w\ \{\Phi\}}{\mathsf{clwp}\ \pi_2\,(v, w)\ \{\Phi\}}$$

$$\textsc{Match-Inj1-CLWP}\ \frac{\triangleright \mathsf{clwp}\ e_1[v/x]\ \{\Phi\}}{\mathsf{clwp}\ \mathsf{match\ inj}_1\, v\ \mathsf{with\ inj}_1\, x \Rightarrow e_1 \mid \mathsf{inj}_2\, x \Rightarrow e_2\ \mathsf{end}\ \{\Phi\}}$$

$$\textsc{match-inj2-clwp}\quad \frac{\triangleright\,\mathsf{clwp}\; e_2[v/x]\;\{\Phi\}}{\mathsf{clwp}\;\mathsf{match}\;\mathsf{inj}_2\,v\;\mathsf{with}\;\mathsf{inj}_1\,x \Rightarrow e_1 \mid \mathsf{inj}_2\,x \Rightarrow e_2\;\mathsf{end}\;\{\Phi\}}$$

$$\textsc{alloc-clwp}\quad \mathsf{clwp}\;K[\mathsf{ref}(v)]\;\{v.\;\exists \ell.\; v = \ell * \ell \mapsto_i v\}$$

$$\textsc{load-clwp}\quad \frac{\triangleright\,\ell \mapsto_i v}{\mathsf{clwp}\;!\,\ell\;\{w.\; w = v * \ell \mapsto_i v\}}$$

$$\textsc{store-clwp}\quad \frac{\triangleright\,\ell \mapsto_i v}{\mathsf{clwp}\;\ell \leftarrow w\;\{v'.\; v' = () * \ell \mapsto_i w\}}$$

$$\textsc{cas-suc-wp}\quad \frac{\triangleright\,\ell \mapsto_i v}{\mathsf{clwp}\;\mathsf{cas}(\ell, v, w)\;\{v'.\; v\;\; = \mathsf{true} * \ell \mapsto_i w\}}$$

$$\textsc{cas-fail-clwp}\quad \frac{\triangleright\,\ell \mapsto_i v' \qquad v \neq v'}{\mathsf{clwp}\;\mathsf{cas}(\ell, v, w)\;\{v''.\; v'' = \mathsf{false} * \ell \mapsto_i v'\}}$$

$$\textsc{fork-clwp}\quad \frac{\triangleright\,\mathsf{clwp}\; e\;\{\_.\;\top\}}{\mathsf{clwp}\;\mathsf{fork}\;\{e\}\;\{v.\; v = ()\}}$$

## 2.4 Logical Relations

Observational refinement ($\mathcal{O} : \mathit{Expr} \times \mathit{Expr} \to \mathit{iProp}$):

$$\mathcal{O}(e, e') \triangleq \forall j.\; j \Mapsto e' \twoheadrightarrow \mathsf{wp}\; e\;\{\exists w.\; j \Mapsto w\}$$

Value interpretation of types ($[\![\Xi \vdash \tau]\!]_\Delta : \mathit{Val} \times \mathit{Val} \to \mathit{iProp}$ for $\Delta : \mathit{Var} \to (\mathit{Val} \times \mathit{Val}) \to \mathit{iProp}$):

$$
\begin{aligned}
[\![\Xi \vdash \alpha]\!]_\Delta &\triangleq \Delta(\alpha) \\
[\![\Xi \vdash 1]\!]_\Delta(v, v') &\triangleq v = v' = () \\
[\![\Xi \vdash \mathbb{B}]\!]_\Delta(v, v') &\triangleq v = v' = \mathsf{true} \lor v = v' = \mathsf{false} \\
[\![\Xi \vdash \mathbb{N}]\!]_\Delta(v, v') &\triangleq \exists n.\ v = v' = n \\
[\![\Xi \vdash \tau_1 + \tau_2]\!]_\Delta(v, v') &\triangleq \bigvee_{i \in \{1,2\}} \left( \begin{array}{c} \exists w, w'.\ v = \mathsf{inj}_i\, w \land v' = \mathsf{inj}_i\, w' \land \\ [\![\Xi \vdash \tau_i]\!]_\Delta(w, w') \end{array} \right) \\
[\![\Xi \vdash \tau \times \tau']\!]_\Delta(v) &\triangleq \exists w_1, w_2, w_1', w2'.\ v = (w_1, w_2) \land v' = (w_1', w_2') \land \\
&\qquad [\![\Xi \vdash \tau]\!]_\Delta(w_1, w_1') * [\![\Xi \vdash \tau']\!]_\Delta(w_2, w_2') \\
[\![\Xi \vdash \mu\alpha.\ \tau]\!]_\Delta(v, v') &\triangleq \mu f : \mathit{Val} \times \mathit{Val} \to \mathit{iProp}.\ \exists w, w'.\ v = \mathsf{fold}\, w \land \\
&\qquad v' = \mathsf{fold}\, w' \land \triangleright [\![\alpha, \Xi \vdash \tau]\!]_{\Delta, \alpha \mapsto f}(w, w') \\
[\![\Xi \vdash \tau \to \tau']\!]_\Delta(v, v') &\triangleq \square \big(\forall w, w'.\ [\![\Xi \vdash \tau]\!]_\Delta(w, w') \Rightarrow \mathcal{E}[\![\Xi \vdash \tau]\!]_\Delta(v\ w, v'\ w')\big) \\
[\![\Xi \vdash \forall \alpha.\ \tau]\!]_\Delta(v, v') &\triangleq \square \big(\forall f : \mathit{Val} \times \mathit{Val} \to \mathit{iProp}. \\
&\qquad \mathsf{persistent}(f) \Rightarrow \mathcal{E}[\![\alpha, \Xi \vdash \tau]\!]_{\Delta, \alpha \mapsto f}(v\ \_, v'\ \_)\big) \\
[\![\Xi \vdash \mathsf{ref}(\tau)]\!]_\Delta(v, v') &\triangleq \exists \ell, \ell'.\ v = \ell \land v' = \ell' \land \\
&\qquad \boxed{\exists w, w'.\ \ell \mapsto_i w * \ell' \mapsto_s w' * [\![\Xi \vdash \tau]\!]_\Delta(w, w')}^{\mathcal{N}.\ell.\ell'} \\
[\![\Xi \vdash \mathsf{cont}(\tau)]\!]_\Delta(v, v) &\triangleq \exists K, K'.\ v = \mathsf{cont}(K) \land v' = \mathsf{cont}(K') \land \\
&\qquad \mathcal{K}[\![\Xi \vdash \tau]\!]_\Delta(K, K')
\end{aligned}
$$

Evaluation context interpretation of types ($\mathcal{K}[\![\Xi \vdash \tau]\!]_\Delta : \mathit{Ectx} \times \mathit{Ectx} \to \mathit{iProp}$ for $\Delta : \mathit{Var} \to (\mathit{Val} \times \mathit{Val}) \to \mathit{iProp}$):

$$\mathcal{K}[\![\Xi \vdash \tau]\!]_\Delta(K, K') \triangleq \forall v, v'.\ [\![\Xi \vdash \tau]\!]_\Delta(v, v') \Rightarrow \mathcal{O}(K[v], K'[v'])$$

Expression interpretation of types ($\mathcal{E}[\![\Xi \vdash \tau]\!]_\Delta : \mathit{Expr} \times \mathit{Expr} \to \mathit{iProp}$ for $\Delta : \mathit{Var} \to (\mathit{Val} \times \mathit{Val}) \to \mathit{iProp}$):

$$\mathcal{E}[\![\Xi \vdash \tau]\!]_\Delta(e, e') \triangleq \forall K, K'.\ \mathcal{K}[\![\Xi \vdash \tau]\!]_\Delta(K, K') \Rightarrow \mathcal{O}(K[e], K'[e'])$$

Logical relatedness ($\Xi \mid \Gamma \vDash e \leq_{\log} e' : \tau : \mathit{iProp}$):

$$
\begin{aligned}
\Xi \mid \Gamma \vDash e \leq_{\log} e' : \tau \triangleq \forall \Delta, \vec{v}, \vec{v'}.\ &\left( \mathop{\Large *}_{x_i : \tau_i} [\![\Xi \vdash \tau_i]\!]_\Delta(v_i, v_i') \right) \Rightarrow \\
&\mathcal{E}[\![\Xi \vdash \tau]\!]_\Delta(e[\vec{v}/\vec{x}], e'[\vec{v'}/\vec{x}])
\end{aligned}
$$

assuming $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$

## 2.5 Inadmissibility of the Bind Rule

Consider the derivation given the following derivation

$$\begin{array}{cl}
\mathsf{false} = \mathsf{false} & \\ \hline
\mathsf{wp}\ \mathsf{false}\ \{v.\ v = \mathsf{false}\} & \\ \hline
\mathsf{wp}\ \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ \mathsf{true}\ \{v.\ v = \mathsf{false}\} & \\ \hline
\mathsf{wp}\ \mathsf{true}\ \{w.\ \mathsf{wp}\ \mathsf{if}\ w\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ \mathsf{true}\ \{v.\ v = \mathsf{false}\}\} & \\ \hline
\mathsf{wp}\ \mathsf{throw}\ \mathsf{true}\ \mathsf{to}\ -\ \{w.\ \mathsf{wp}\ \mathsf{if}\ w\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ \mathsf{true}\ \{v.\ v = \mathsf{false}\}\} & \\ \hline
\mathsf{wp}\ \mathsf{if}\ (\mathsf{throw}\ \mathsf{true}\ \mathsf{to}\ -)\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ \mathsf{true}\ \{v.\ v = \mathsf{false}\} & \textsc{inadmissible-bind} \\ \hline
\mathsf{wp}\ \mathsf{call/cc}\ (x.\ \mathsf{if}\ (\mathsf{throw}\ \mathsf{true}\ \mathsf{to}\ x)\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ \mathsf{true})\ \{v.\ v = \mathsf{false}\} &
\end{array}$$

Here, we omit steps corresponding to eliminations of $\triangleright$. Note that we can easily show that

$$\mathsf{call/cc}\ (x.\ \mathsf{if}\ (\mathsf{throw}\ \mathsf{true}\ \mathsf{to}\ x)\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ \mathsf{true})$$

reduces to the value $\mathsf{true}$ which falsifies the derivation above.

### 2.5.1 The resource for one-shot bit

For details of representing resources with monoids in Iris see Section 1. The predicates for representing the one-shot bits, $\mathit{OneShotBits}(M)$ and $\mathit{isOneShotBit}(b)$, are defined as follows:

$$\begin{aligned}
\textsc{OneShotBit} &\triangleq \textsc{Auth}(\mathsf{finset}(\ell)) \\
\mathit{OneShotBits}(M) &\triangleq \boxed{\bullet\, M}^{\gamma_{os}} \\
\mathit{isOneShotBit}(b) &\triangleq \boxed{\circ\ \{b\}}^{\gamma_{os}}
\end{aligned}$$

It is easy to see, based on the rules for monoids in Section 1, that $\mathit{isOneShotBit}(b)$ is persistent and that:

$$\mathit{OneShotBits}(M) * \mathit{isOneShotBit}(b) \vdash \checkmark(\bullet\, M \cdot \circ\ \{b\}) \vdash \{b\} \subseteq M \vdash b \in M$$

## References

R. Jung, R. Krebbers, L. Birkedal, and D. Dreyer. Higher-order ghost state. In *ICFP*, pages 256–269, 2016.

R. Krebbers, R. Jung, A. Bizjak, J.-H. Jourdan, D. Dreyer, and L. Birkedal. The essence of higher-order concurrent separation logic. In *European Symposium on Programming (ESOP)*, April 2017.