Unifying Refinement and Hoare-Style Reasoning in a Logic for Higher-Order Concurrency (Technical Appendix)

Aaron Turon
MPI-SWS,
Germany
turon@mpi-sws.org

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

Lars Birkedal
Aarhus
University
birkedal@cs.au.dk

April 3, 2013

Note: This online appendix contains a few minor typo corrections found after submission. It also contains a more significant correction to the Flat Combining case study. The original submission contained a last-minute simplification of the protocol that turned out not to be sound. This updated appendix (and the updated paper available at http://www.mpi-sws.org/~turon/caresl/caresl.pdf) contains the original protocol, which has an additional token needed to "freeze" the protocol in certain states.

To accommodate the presentation of the semantic model, the notation in this appendix differs in one small way from that in the paper. Protocols in the appendix have a *nested* component θ giving their transition system and token map, *i.e.,* we have $\pi ::= (\theta, \varphi)$ and $\theta ::= (S, \rightsquigarrow, \mathcal{T})$. This means that the full proof outlines here differ slightly from the corresponding sketches in the paper. The syntax is presented in complete detail below.

Other minor differences: some of the domain and sort names are spelled out in longer form here than in the paper.

Contents

1 The language

1.1 Syntax

Values	v	::=	$()$	Unit value
		|	true	Boolean value
		|	false	Boolean value
		|	(v, v)	Pair value
		|	$\mathsf{inj}_i\ v$	Sum value
		|	$\mathsf{rec}\ f(x).e$	Recursive function
		|	$\Lambda.e$	Type abstraction
		|	n	Heap location
		|	x	Variable
Expressions	e	::=	v	Value
		|	$\mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e$	Conditional
		|	(e, e)	Pair introduction
		|	$\mathsf{prj}_i\ e$	Pair elimination
		|	$\mathsf{inj}_i\ e$	Sum introduction
		|	$\mathsf{case}(e, \mathsf{inj}_1\ x \Rightarrow e, \mathsf{inj}_2\ y \Rightarrow e)$	Sum elimination
		|	$e\ e$	Function application
		|	$e\ \_$	Type application
		|	$\mathsf{new}\ e$	Allocation
		|	$\mathsf{get}\ e$	Dereferencing
		|	$e\ :=\ e$	Assignment
		|	$\mathsf{CAS}(e, e, e)$	Atomic update
		|	newLcl	Thread local allocation
		|	$\mathsf{getLcl}(e)$	Thread local dereference
		|	$\mathsf{setLcl}(e, e)$	Thread local assignment
		|	$\mathsf{fork}\ e$	Thread forking
Comparable types	σ	::=	B	Boolean
		|	$\mathsf{ref}\ \tau$	Reference
		|	$\mathsf{refLcl}\ \tau$	Thread local reference
		|	$\mu\alpha.\sigma$	Recursive comparable type
Types	τ	::=	σ	Comparable type
		|	$\mathbf{1}$	Unit (*i.e.,* nullary tuple)
		|	α	Type variable
		|	$\tau \times \tau$	Product type
		|	$\tau + \tau$	Sum type
		|	$\mu\alpha.\tau$	Recursive type
		|	$\forall\alpha.\tau$	Polymorphic type
		|	$\tau \to \tau$	Function type

Recursive types must be *guarded*: in $\mu\alpha.\tau$, the variable α must appear under some non-μ type constructor.

1.2 Typing

Type context syntax

$$\begin{array}{lrcl} \textsc{Type variable contexts} & \Delta & ::= & \cdot \mid \Delta, \alpha \\ \textsc{Term variable contexts} & \Gamma & ::= & \cdot \mid \Gamma, x : \tau \\ \textsc{Combined contexts} & \Omega & ::= & \Delta; \Gamma \end{array}$$

Well-typed expressions $\boxed{\Delta; \Gamma \vdash e : \tau}$

$$\Omega, x : \tau \vdash x : \tau \qquad \Omega \vdash () : \mathbf{1} \qquad \Omega \vdash \mathsf{true} : \mathbf{B} \qquad \Omega \vdash \mathsf{false} : \mathbf{B} \qquad \frac{\Omega \vdash e : \mathbf{B} \qquad \Omega \vdash e_i : \tau}{\Omega \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

$$\frac{\Omega \vdash e_1 : \tau_1 \qquad \Omega \vdash e_2 : \tau_2}{\Omega \vdash (e_1, e_2) : \tau_1 \times \tau_2} \qquad \frac{\Omega \vdash e : \tau_1 \times \tau_2}{\Omega \vdash \mathsf{prj}_i\ e : \tau_i} \qquad \frac{\Omega \vdash e : \tau_i}{\Omega \vdash \mathsf{inj}_i\ e : \tau_1 + \tau_2}$$

$$\frac{\Omega \vdash e : \tau_1 + \tau_2 \qquad \Omega, x : \tau_i \vdash e_i : \tau}{\Omega \vdash \mathsf{case}(e, \mathsf{inj}_1\ x \Rightarrow e_1, \mathsf{inj}_2\ x \Rightarrow e_2) : \tau} \qquad \frac{\Omega, f : \tau' \to \tau, x : \tau' \vdash e : \tau}{\Omega \vdash \mathsf{rec}\ f(x).e : \tau' \to \tau} \qquad \frac{\Omega \vdash e : \tau' \to \tau \qquad \Omega \vdash e' : \tau'}{\Omega \vdash e\ e' : \tau}$$

$$\frac{\Omega, \alpha \vdash e : \tau}{\Omega \vdash \Lambda.e : \forall \alpha.\tau} \qquad \frac{\Omega \vdash e : \forall \alpha.\tau}{\Omega \vdash e\ \_ : \tau[\tau'/\alpha]} \qquad \frac{\Omega \vdash e : \tau}{\Omega \vdash \mathsf{new}\ e : \mathsf{ref}\ \tau} \qquad \frac{\Omega \vdash e : \mathsf{ref}\ \tau}{\Omega \vdash \mathsf{get}\ e : \tau} \qquad \frac{\Omega \vdash e : \mathsf{ref}\ \tau \qquad \Omega \vdash e' : \tau}{\Omega \vdash e\ :=\ e' : \mathbf{1}}$$

$$\frac{\Omega \vdash e : \mathsf{ref}\ \sigma \qquad \Omega \vdash e_o : \sigma \qquad \Omega \vdash e_n : \sigma}{\Omega \vdash \mathsf{CAS}(e, e_o, e_n) : \mathbf{B}} \qquad \Omega \vdash \mathsf{newLcl} : \mathsf{refLcl}\ \tau \qquad \frac{\Omega \vdash e : \mathsf{refLcl}\ \tau}{\Omega \vdash \mathsf{getLcl}(e) : \mathbf{1} + \tau}$$

$$\frac{\Omega \vdash e : \mathsf{refLcl}\ \tau \qquad \Omega \vdash e' : \tau}{\Omega \vdash \mathsf{setLcl}(e, e') : \mathbf{1}} \qquad \frac{\Omega \vdash e : \mathbf{1}}{\Omega \vdash \mathsf{fork}\ e : \mathbf{1}} \qquad \frac{\Omega \vdash e : \mu\alpha.\tau}{\Omega \vdash e : \tau[\mu\alpha.\tau/\alpha]} \qquad \frac{\Omega \vdash e : \tau[\mu\alpha.\tau/\alpha]}{\Omega \vdash e : \mu\alpha.\tau}$$

1.3 Operational semantics

Machine syntax

$$\begin{array}{lrcl} \textsc{Thread-local value} & L & \in & \mathbb{N} \xrightarrow{\text{fin}} \text{Value} \\ \textsc{Heap-stored values} & u & ::= & v \mid L \\ \textsc{Heaps} & h & \in & \text{Heap} \triangleq \mathbb{N} \xrightarrow{\text{fin}} \text{HeapVal} \\ \textsc{Thread pools} & \mathcal{E} & \in & \text{ThreadPool} \triangleq \mathbb{N} \xrightarrow{\text{fin}} \text{Expression} \\ \textsc{Machine state} & \varsigma & ::= & h; \mathcal{E} \\ \textsc{Evaluation contexts} & K & ::= & [\,] \mid \mathsf{if}\ K\ \mathsf{then}\ e\ \mathsf{else}\ e \mid K\ e \mid v\ K \mid K\ \_ \\ & & \mid & (K, e) \mid (v, K) \mid \mathsf{prj}_i\ K \mid \mathsf{inj}_i\ K \mid \mathsf{case}(K, \mathsf{inj}_1\ x \Rightarrow e, \mathsf{inj}_2\ x \Rightarrow e) \\ & & \mid & \mathsf{new}\ K \mid \mathsf{get}\ K \mid K\ :=\ e \mid v\ :=\ K \\ & & \mid & \mathsf{CAS}(K, e, e) \mid \mathsf{CAS}(v, K, e) \mid \mathsf{CAS}(v, v, K) \\ & & \mid & \mathsf{getLcl}(K) \mid \mathsf{setLcl}(K, e) \mid \mathsf{setLcl}(v, K) \end{array}$$

Thread-local ref lifting

$$\lfloor L \rfloor(i) \triangleq \begin{cases} \mathsf{inj}_1\ () & i \notin \mathrm{dom}(L) \\ \mathsf{inj}_2\ L(i) & \text{otherwise} \end{cases}$$

Pure reduction $\boxed{e \overset{\text{pure}}{\to} e'}$

$$
\begin{aligned}
\textbf{if true then } e_1 \textbf{ else } e_2 &\overset{\text{pure}}{\to} e_1 \\
\textbf{if false then } e_1 \textbf{ else } e_2 &\overset{\text{pure}}{\to} e_1 \\
\textbf{prj}_i\ (v_1, v_2) &\overset{\text{pure}}{\to} v_i \\
\textbf{case}(\textbf{inj}_i\ v, \textbf{inj}_1\ x \Rightarrow e_1, \textbf{inj}_2\ x \Rightarrow e_2) &\overset{\text{pure}}{\to} e_i[v/x] \\
\textbf{rec}\ f(x).e\ v &\overset{\text{pure}}{\to} e[\textbf{rec}\ f(x).e/f, v/x] \\
\Lambda.e\ \_ &\overset{\text{pure}}{\to} e
\end{aligned}
$$

Per-thread reduction $\boxed{h; e \overset{i}{\to} h'; e'}$

$$
\begin{aligned}
h; e &\overset{i}{\to} h; e' && \text{when } e \overset{\text{pure}}{\to} e' \\
h; \textbf{new}\ v &\overset{i}{\to} h \uplus [\ell \mapsto v]; \ell \\
h; \textbf{get}\ \ell &\overset{i}{\to} h; v && \text{when } h(\ell) = v \\
h \uplus [\ell \mapsto \_]; \ell\ :=\ v &\overset{i}{\to} h \uplus [\ell \mapsto v]; () \\
h \uplus [\ell \mapsto v_o]; \textbf{CAS}(\ell, v_o, v_n) &\overset{i}{\to} h \uplus [\ell \mapsto v_n]; \textbf{true} \\
h; \textbf{CAS}(\ell, v_o, v_n) &\overset{i}{\to} h; \textbf{false} && \text{when } h(\ell) \neq v_o \\
h; \textbf{newLcl} &\overset{i}{\to} h \uplus [\ell \mapsto \emptyset]; \ell \\
h; \textbf{getLcl}(\ell) &\overset{i}{\to} h; v && \text{when } h(\ell) = L,\ \lfloor L \rfloor(i) = v \\
h \uplus [\ell \mapsto L]; \textbf{setLcl}(\ell, v) &\overset{i}{\to} h \uplus [\ell \mapsto L[i := v]]; ()
\end{aligned}
$$

General reduction $\boxed{h; \mathcal{E} \to h'; \mathcal{E}'}$

$$
\frac{h; e \overset{i}{\to} h'; e'}{h; \mathcal{E} \uplus [i \mapsto K[e]] \to h'; \mathcal{E} \uplus [i \mapsto K[e']]}
\qquad
h; \mathcal{E} \uplus [i \mapsto K[\textbf{fork}\ e]] \to h; \mathcal{E} \uplus [i \mapsto K[()]] \uplus [j \mapsto e]
$$

1.4 Refinement

If $\Omega \vdash e_{\text{I}} : \tau$ and $\Omega \vdash e_{\text{S}} : \tau$, we say e_{I} *contextually refines* e_{S}, written $\Omega \models e_{\text{I}} \preceq e_{\text{S}} : \tau$, if:

$$
\begin{aligned}
&\text{for every } i, j \text{ and } C : (\Omega, \tau) \rightsquigarrow (\emptyset, \textbf{N}) \text{ we have} \\
&\quad \forall n. \forall \mathcal{E}_{\text{I}}.\quad \emptyset; [i \mapsto C[e_{\text{I}}]] \ \to^*\ h_{\text{I}}; [i \mapsto n] \uplus \mathcal{E}_{\text{I}} \\
&\implies \exists \mathcal{E}_{\text{S}}.\quad \emptyset; [j \mapsto C[e_{\text{S}}]] \to^* h_{\text{S}}; [j \mapsto n] \uplus \mathcal{E}_{\text{S}}
\end{aligned}
$$

1.5 Derived forms

$$
\begin{aligned}
\lambda x.e &\triangleq \textbf{rec}\ \_(x).e \\
\textbf{let}\ x = e\ \textbf{in}\ e' &\triangleq (\lambda \_.e')\ e \\
e; e' &\triangleq \textbf{let}\ \_ = e\ \textbf{in}\ e' \\
\\
\tau_? &\triangleq \mathbf{1} + \tau \\
\mathsf{none} &\triangleq \textbf{inj}_1\ () \\
\mathsf{some}(e) &\triangleq \textbf{inj}_2\ e
\end{aligned}
$$

2 Example code

2.1 Pure lists

$$
\begin{array}{rcl}
\mathsf{list}(\alpha) & \triangleq & \mu\beta.\ \mathbf{1} + (\alpha \times \beta) \\
\mathsf{foreach} & : & \forall\alpha.\ (\alpha \to \mathbf{1}) \to \mathsf{list}(\alpha) \to \mathbf{1} \\
\mathsf{foreach} & \triangleq & \Lambda[\alpha].\ \lambda f : [\alpha \to \mathbf{1}].\ \mathbf{rec}\ \mathsf{loop}(l).\ \mathbf{case}\ l\ \mathbf{of}\ \mathsf{none} \Rightarrow () \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mid\ \mathsf{some}(x, n) \Rightarrow f(x);\ \mathsf{loop}(n)
\end{array}
$$

2.2 Locking

$$
\begin{array}{rcl}
\mathsf{tryAcq} & \triangleq & \lambda x.\ \mathbf{CAS}(x, \mathbf{false}, \mathbf{true}) \\
\mathsf{acq} & \triangleq & \mathbf{rec}\ \mathsf{loop}(x).\ \mathbf{if}\ \mathsf{tryAcq}(x)\ \mathbf{then}\ ()\ \mathbf{else}\ \mathsf{loop}(x) \\
\mathsf{rel} & \triangleq & \lambda x.\ x\ :=\ \mathbf{false} \\
\mathsf{withLock}(f, \mathsf{lock}) & \triangleq & \lambda x.\ \mathsf{acq}(\mathsf{lock}); \mathbf{let}\ r = f(x)\ \mathbf{in}\ \mathsf{rel}(\mathsf{lock});\ r \\
\mathsf{mkSync} & \triangleq & \lambda().\mathbf{let}\ \mathsf{lock} = \mathbf{new}\ \mathbf{false}\ \mathbf{in}\ \Lambda[\alpha].\Lambda[\beta].\lambda f : [\alpha \to \beta].\mathsf{withLock}(f, \mathsf{lock})
\end{array}
$$

2.3 Stack (used in flat combining)

2.3.1 Specification

$\mathsf{stack}_{\mathrm{S}} \;:\; \forall\alpha.\ (\alpha \to \mathbf{1}) \times (\mathbf{1} \to \alpha_?) \times ((\alpha \to \mathbf{1}) \to \mathbf{1})$
$\mathsf{stack}_{\mathrm{S}} \triangleq \Lambda[\alpha].$
  let hd $= \mathbf{new}\ (\mathsf{none})$
  let sync $= \mathsf{mkSync}()$
  let push $= \lambda x.\ \mathsf{hd} := \mathsf{some}(x, \mathsf{get}(\mathsf{hd}))$
  let pop $= \lambda().$ **case** get(hd) **of** none $\Rightarrow$ none
    $\mid$ some$(x, n) \Rightarrow$ hd $:= n;$ some(x)
  let snap $= \mathsf{sync}\ [\mathbf{1}]\ [\mathsf{list}(\alpha)]\ (\lambda().\mathsf{get}(\mathsf{hd}))$
  let iter $= \lambda f.\ \mathsf{foreach}\ \alpha\ f\ (\mathsf{snap}())$
  in (sync $[\alpha]\ [\mathbf{1}]$ push, sync $[\mathbf{1}]\ [\alpha_?]$ pop, iter)

2.3.2 Implementation

Adaptation of Treiber's stack.

$\mathsf{clist}(\alpha) \triangleq \mu\beta.\ \mathbf{ref}\ \mathbf{1} + (\alpha \times \beta)$ (*Comparable* list: allows CAS)
$\mathsf{nil} \triangleq \mathbf{new}\ \mathsf{none}$
$\mathsf{cons}\ e \triangleq \mathbf{new}\ \mathsf{some}(e)$

$\mathsf{stack}_{\mathrm{I}}\colon \forall\alpha.\ (\alpha \to \mathbf{1}) \times (\mathbf{1} \to \alpha_?) \times ((\alpha \to \mathbf{1}) \to \mathbf{1})$
$\mathsf{stack}_{\mathrm{I}} \triangleq \Lambda[\alpha].$
  let hd $= \mathbf{new}$ nil
  let push $=$ **rec** try(x).
    let $c = \mathbf{get}$ hd
    let $n = \mathsf{cons}(x, c)$
    in if **CAS**(hd, c, n) **then** $()$ **else** try(x),
  let pop $=$ **rec** try$()$.
    let $c = \mathbf{get}$ hd
    in case get c **of**
      none $\Rightarrow$ none
    $\mid$ some$(d, n) \Rightarrow$ **if** **CAS**(hd, c, n) **then** some(d) **else** try()
  let iter $= \lambda f.$
    let rec loop$(c) =$ **case get** c **of**
      none $\Rightarrow$ $()$
    $\mid$ some$(d, n) \Rightarrow f(d); \mathsf{loop}(n)$
    in loop(**get** hd)
  in (push, pop, iter)

2.4 Universal construction

2.4.1 Specification

$\mathsf{flat}_{\mathrm{S}}\ :\ \ \forall\alpha.\,\forall\beta.\,(\alpha\to\beta)\to(\alpha\to\beta)$
$\mathsf{flat}_{\mathrm{S}}\ \triangleq\ \ \mathsf{mkSync}()$

2.4.2 Flat combining implementation

$\mathsf{flat}_{\mathrm{I}}\ :\ \ \forall\alpha.\,\forall\beta.\,(\alpha\to\beta)\to(\alpha\to\beta)$
$\mathsf{flat}_{\mathrm{I}}\ \triangleq\ \ \Lambda[\alpha].\,\Lambda[\beta].\,\lambda f : [\alpha\to\beta].$

```
  let lock : [ref B] = new (false)
  let localOps : [refLcl op] = newLcl    (where op ≜ ref α + β)
  let (add, _, iter) = stack_I [op]
  let doOp : [op → 1] = λo.
    case get(o) of inj_1 req ⇒ o := inj_2 f(req)
                 | inj_2 res ⇒ ()
  let install : [α → op] = λ req.
    case getLcl(localOps)
      of none      ⇒ let o = new (inj_1 req)
                     in setLcl(localOps, o);
                        add(o);
                        o
       | some(o)  ⇒ o := inj_1 req; o
  in λx. let o = install(x)
         let rec loop() = case get(o)
           of inj_1 _   ⇒ if not(get lock) and tryAcq(lock) then
                            iter doOp;
                            rel(lock);
                            loop()
                          else loop()
            | inj_2 res ⇒ res
         in loop()
```

2.4.3 Counterexample to refinement

```
let r = newLcl
let f = flat [1] [1_?] (λ(). getLcl(r))
in fork f();
   setLcl(r, ());
   f()
```

With $\mathsf{flat}_{\mathrm{S}}$, always returns some(). With $\mathsf{flat}_{\mathrm{I}}$, can also return none.

3 Logic: syntax

3.1 Grammar

Impl/Spec	is	::	$\mathrm{I} \mid \mathrm{S}$	
Terms	M, N	::=	$X \mid n \mid e \mid K \mid M[M] \mid (M, M) \mid M \uplus M \mid \lfloor M \rfloor(x) \mid M[x := M] \mid \cdots$	
Sorts	Σ	::=	$\mathbf{1} \mid \mathrm{Nat} \mid \mathrm{Val} \mid \mathrm{Exp} \mid \mathrm{AExp} \mid \mathrm{EvalCtx} \mid \mathrm{ThreadPool}$	
		$\mid$	$\mathrm{TokenSet} \mid \mathrm{State} \mid \mathrm{LocalStorage} \mid \Sigma \times \Sigma$	
Absolute props.	A	::=	$M = M$	Equality
		$\mid$	$\mathrm{TokPure}(P)$	Token-pure proposition
		$\mid$	$M \overset{\text{pure}}{\to} M$	Effect-free step
		$\mid$	$P \to_{\mathrm{S}} P$	Spec step
		$\mid$	$M \overset{\text{rely}}{\sqsubseteq}_{\pi} M$	Rely move
		$\mid$	$M \overset{\text{guar}}{\sqsubseteq}_{\pi} M$	Guarantee move
Propositions	P, Q, R	::=	A	Absolute proposition
		$\mid$	$M \hookrightarrow_{\mathrm{I}} M$	Singleton implementation heap
		$\mid$	$P * P$	Separating conjunction
		$\mid$	$P \Rightarrow P$	Implication
		$\mid$	$P \wedge P$	Conjunction
		$\mid$	$P \vee P$	Disjunction
		$\mid$	$\exists X \in \Sigma. P$	Existential quantification (first order)
		$\mid$	$\forall X \in \Sigma. P$	Universal quantification (first order)
		$\mid$	$\varphi(M)$	Predicate elimination (also written $M \in \varphi$)
		$\mid$	$\exists p \in \mathbb{P}(\Sigma). P$	Existential quantification (second order)
		$\mid$	$\forall p \in \mathbb{P}(\Sigma). P$	Universal quantification (second order)
		$\mid$	$\Box P$	Always modality
		$\mid$	$\triangleright P$	Later modality
		$\mid$	$\boxed{M}^{M}_{\pi}$	Island assertion
		$\mid$	$\mathit{Tid}(M)$	Thread ID token
		$\mid$	$\{P\}\ M \Mapsto M\ \{\varphi\}$	Concurrent Hoare triple
		$\mid$	$(\!\vert P \vert\!)\ M \Mapsto_{\mathrm{IS}} M\ (\!\vert \varphi \vert\!)$	Atomic Hoare triple
		$\mid$	$M \hookrightarrow_{\mathrm{S}} M$	Singleton specification heap
		$\mid$	$M \Mapsto_{\mathrm{S}} M$	Singleton specification thread
		$\mid$	$M \bowtie M$	Relatedness between thread IDs
Predicates	φ, ψ	::=	p	Predicate variable (also q, r)
		$\mid$	$(X \in \Sigma). P$	Predicate introduction
		$\mid$	$\mu p \in \mathbb{P}(\Sigma). \varphi$	Recursive predicate
STS	θ	::=	$(S, \rightsquigarrow, \mathcal{T})$	
			where $S \subseteq \mathrm{State}$,	States
			$\rightsquigarrow \subseteq S \times S$,	Transition relation
			$\mathcal{T} \in S \to \wp(\mathrm{Token})$	Free tokens
Protocols	π	::=	(θ, φ) $\quad \varphi$ token-pure	

Recursive predicates must be *guarded*: in $\mu\alpha \in \mathbb{P}(\Sigma). P$, the variable α must appear under the later $\triangleright$ modality.

3.2 Sort system (selected rules)

Variable contexts

$$\mathcal{X} \quad ::= \quad \cdot \mid \mathcal{X}, X : \Sigma \mid \mathcal{X}, p : \mathbb{P}(\Sigma)$$

Well-sorted terms $\boxed{\mathcal{X} \vdash M : \Sigma}$

$$\frac{}{\mathcal{X}, X : \Sigma \vdash X : \Sigma} \qquad \frac{}{\mathcal{X} \vdash n : \mathrm{Nat}} \qquad \frac{\mathcal{X} \vdash M_1 : \Sigma_1 \qquad \mathcal{X} \vdash M_2 : \Sigma_2}{\mathcal{X} \vdash (M_1, M_2) : \Sigma_1 \times \Sigma_2}$$

$$\frac{\mathcal{X} \vdash M : \mathrm{EvalCtx} \qquad \mathcal{X} \vdash N : \mathrm{EvalCtx}}{\mathcal{X} \vdash M[N] : \mathrm{EvalCtx}} \qquad \frac{\mathcal{X} \vdash M : \mathrm{EvalCtx} \qquad \mathcal{X} \vdash N : \mathrm{Exp}}{\mathcal{X} \vdash M[N] : \mathrm{Exp}} \qquad \frac{\mathcal{X} \vdash M : \mathrm{Val}}{\mathcal{X} \vdash \mathsf{new}\ M : \mathrm{AExp}}$$

$$\frac{\mathcal{X} \vdash M : \mathrm{Val}}{\mathcal{X} \vdash \mathsf{get}\ M : \mathrm{AExp}} \qquad \frac{\mathcal{X} \vdash M : \mathrm{Val} \qquad \mathcal{X} \vdash N : \mathrm{Val}}{\mathcal{X} \vdash M\ :=\ N : \mathrm{AExp}} \qquad \frac{\mathcal{X} \vdash M : \mathrm{Val} \qquad \mathcal{X} \vdash N_o : \mathrm{Val} \qquad \mathcal{X} \vdash N_n : \mathrm{Val}}{\mathcal{X} \vdash \mathsf{CAS}(M, N_o, N_n) : \mathrm{AExp}}$$

$$\mathcal{X} \vdash \mathsf{newLcl} : \mathrm{AExp} \qquad \frac{\mathcal{X} \vdash M : \mathrm{Val}}{\mathcal{X} \vdash \mathsf{getLcl}(M) : \mathrm{AExp}} \qquad \frac{\mathcal{X} \vdash M : \mathrm{Val} \qquad \mathcal{X} \vdash N : \mathrm{Val}}{\mathcal{X} \vdash \mathsf{setLcl}(M, N) : \mathrm{AExp}} \qquad \mathcal{X} \vdash \emptyset : \mathrm{LocalStorage}$$

$$\frac{\mathcal{X} \vdash M : \mathrm{Nat} \qquad \mathcal{X} \vdash N : \mathrm{Val}}{\mathcal{X} \vdash [M \mapsto N] : \mathrm{LocalStorage}} \qquad \frac{\mathcal{X} \vdash M : \mathrm{LocalStorage} \qquad \mathcal{X} \vdash N : \mathrm{LocalStorage}}{\mathcal{X} \vdash M \uplus N : \mathrm{LocalStorage}}$$

$$\frac{\mathcal{X} \vdash M : \mathrm{LocalStorage} \qquad \mathcal{X} \vdash N : \mathrm{Nat}}{\mathcal{X} \vdash \lfloor M \rfloor(N) : \mathrm{Val}} \qquad \frac{\mathcal{X} \vdash M : \mathrm{TokenSet} \qquad \mathcal{X} \vdash N : \mathrm{TokenSet}}{\mathcal{X} \vdash M \uplus N : \mathrm{TokenSet}}$$

Well-sorted propositions $\boxed{\mathcal{X} \vdash P : \mathbb{B}}$

$$\frac{\mathcal{X} \vdash M : \Sigma \qquad \mathcal{X} \vdash N : \Sigma}{\mathcal{X} \vdash M = N : \mathbb{B}} \qquad \frac{\mathcal{X} \vdash \varphi : \mathbb{P}(\Sigma) \qquad \mathcal{X} \vdash M : \Sigma}{\mathcal{X} \vdash \varphi(M) : \mathbb{B}} \qquad \frac{\mathcal{X} \vdash P : \mathbb{B} \qquad \mathcal{X} \vdash Q : \mathbb{B}}{\mathcal{X} \vdash P \vee Q : \mathbb{B}}$$

$$\frac{\mathcal{X}, p : \mathbb{P}(\Sigma) \vdash P : \mathbb{B}}{\mathcal{X} \vdash \exists p \in \mathbb{P}(\Sigma). P : \mathbb{B}} \qquad \frac{\mathcal{X} \vdash M : \mathrm{Nat} \qquad \mathcal{X} \vdash N : \mathrm{Exp}}{\mathcal{X} \vdash M \mapsto_{\mathrm{S}} N : \mathbb{B}} \qquad \frac{\mathcal{X} \vdash M : \mathrm{State} \times \mathrm{TokenSet} \qquad \mathcal{X} \vdash N : \mathrm{Nat}}{\mathcal{X} \vdash \boxed{M}_{\pi}^{N} : \mathbb{B}}$$

$$\frac{\mathcal{X} \vdash P : \mathbb{B} \qquad \mathcal{X} \vdash M : \mathrm{Nat} \qquad \mathcal{X} \vdash N : \mathrm{Exp} \qquad \mathcal{X} \vdash \varphi : \mathbb{P}(\mathrm{Val})}{\mathcal{X} \vdash \{P\}\ M \mapsto N\ \{\varphi\} : \mathbb{B}}$$

$$\frac{\mathcal{X} \vdash P : \mathbb{B} \qquad \mathcal{X} \vdash M : \mathrm{Nat} \qquad \mathcal{X} \vdash N : \mathrm{AExp} \qquad \mathcal{X} \vdash \varphi : \mathbb{P}(\mathrm{Val})}{\mathcal{X} \vdash (\!|P|\!)\ M \mapsto_{\mathrm{IS}} N\ (\!|\varphi|\!) : \mathbb{B}}$$

Well-sorted predicates $\boxed{\mathcal{X} \vdash \varphi : \mathbb{P}(\Sigma)}$

$$\frac{}{\mathcal{X}, p : \mathbb{P}(\Sigma) \vdash p : \mathbb{P}(\Sigma)} \qquad \frac{\mathcal{X}, X : \Sigma \vdash P : \mathbb{B}}{\mathcal{X} \vdash X \in \Sigma.\ P : \mathbb{P}(\Sigma)} \qquad \frac{\mathcal{X}, p : \mathbb{P}(\Sigma) \vdash \varphi : \mathbb{P}(\Sigma)}{\mathcal{X} \vdash \mu p \in \mathbb{P}(\Sigma).\ \varphi : \mathbb{P}(\Sigma)}$$

3.3 Conventions

Throughout, we use additional metavariables to denote terms (or variables) of specific sorts, often leaving off sort annotations:

$$\begin{array}{rcl} x, y, z, v & : & \text{Val} \\ e & : & \text{Exp} \\ a & : & \text{AExp} \\ i, j, k & : & \text{Nat} \\ s & : & \text{State} \\ T & : & \text{TokenSet} \\ b & : & \text{State} \times \text{TokenSet} \\ K, \kappa & : & \text{EvalCtx} \\ L & : & \text{LocalStorage} \end{array}$$

4 Logic: semantics

4.1 Semantic structures

$$\begin{array}{lcl} \text{State} & \ni & s \\ \wp(\text{Token}) & \ni & T \\ \text{Resource} & \triangleq & \{\ \eta = (h, \varsigma, I) \ \mid\ I \in \wp(\mathbb{N})\ \} \\ \text{Island}_n & \triangleq & \left\{\ \iota = (\theta, \Phi, (s, T)) \ \middle|\ \Phi \in [\![\mathbb{P}(\text{State})]\!]_n,\ T \# \theta.\mathcal{T}(s)\ \right\} \\ \text{ThreadMap} & \triangleq & \{\ B \subseteq \mathbb{N} \times \mathbb{N} \ \mid\ B \text{ a partial bijection}\ \} \\ \text{World}_n & \triangleq & \left\{\ W = (k, \omega, B) \ \middle|\ k < n,\quad \omega \in \mathbb{N} \xrightarrow{\text{fin}} \text{Island}_k\ \right\} \end{array}$$

$$\begin{array}{lcl} [\![\mathbf{1}]\!] & \triangleq & \{()\} \\ [\![\text{Nat}]\!] & \triangleq & \mathbb{N} \\ [\![\text{Val}]\!] & \triangleq & \text{Val} \\ [\![\text{Exp}]\!] & \triangleq & \text{Exp} \\ [\![\text{AExp}]\!] & \triangleq & \text{AExp} \\ [\![\text{EvalCtx}]\!] & \triangleq & \text{EvalCtx} \\ [\![\text{ThreadPool}]\!] & \triangleq & \text{ThreadPool} \\ [\![\text{State}]\!] & \triangleq & \text{State} \\ [\![\text{TokenSet}]\!] & \triangleq & \wp_{\text{fin}}(\text{Token}) \\ [\![\text{LocalStorage}]\!] & \triangleq & \text{LocalStorage} \\ [\![\Sigma_1 \times \Sigma_2]\!] & \triangleq & [\![\Sigma_1]\!] \times [\![\Sigma_2]\!] \\ [\![\mathbb{B}]\!]_n & \triangleq & \text{World}_n \times \text{Resource} \xrightarrow{\text{mon}} \mathbb{B} \\ [\![\mathbb{P}(\Sigma)]\!]_n & \triangleq & \text{World}_n \times \text{Resource} \xrightarrow{\text{mon}} \wp([\![\Sigma]\!]) \end{array}$$

The function space $\xrightarrow{\text{mon}}$ includes only *monotonic* functions from $\text{World}_n \times \text{Resource}$ to an ordered codomain, *i.e.,* f is monotonic iff

$$\forall W' \geq W, \eta' \geq \eta.\ f(W', \eta') \geq f(W, \eta)$$

where $\geq$ on worlds and resources is defined in section 4.4 below.

4.2 Island and world operations

$$
\begin{array}{lcl}
|(\theta, \Phi, s, T)| & \triangleq & (\theta, \Phi, s, \emptyset) \\
|(k, \omega, B)| & \triangleq & (k, \lambda i.|\omega(i)|, B) \\
\\
\lfloor(\theta, \Phi, s, T)\rfloor_k & \triangleq & (\theta, \Phi \restriction \mathrm{World}_k \times \mathrm{Resource}, s, T) \\
\triangleright(k+1, \omega, B) & \triangleq & (k,\ \lambda i.\lfloor\omega(i)\rfloor_k,\ B) \\
\\
\mathrm{interp}(\theta, \Phi, s, T)(W) & \triangleq & \{\ \eta \ \mid\ s \in \Phi(W, \eta)\ \}
\end{array}
$$

4.3 Resource composition

$$
\begin{array}{lrcll}
\text{Resources} & (h_1, (h_1'; \mathcal{E}_1), I_1) & \otimes & (h_2, (h_2'; \mathcal{E}_2), I_2) & \triangleq\ (h_1 \uplus h_2, (h_1' \uplus h_2'; \mathcal{E}_1 \uplus \mathcal{E}_2), I_1 \uplus I_2) \\
\text{Islands} & (\theta, \Phi, s, T) & \otimes & (\theta', \Phi', s', T') & \triangleq\ (\theta, \Phi, s, T \uplus T') \quad \text{when } \theta = \theta',\ s = s',\ \Phi = \Phi' \\
\text{Worlds} & (k, \omega, B) & \otimes & (k', \omega', B') & \triangleq\ (k,\ \lambda i.\omega(i) \otimes \omega'(i),\ B) \quad \text{when } k = k',\ B = B',\ \mathrm{dom}(\omega) = \mathrm{dom}(\omega')
\end{array}
$$

4.4 World and resource ordering

$$
\begin{array}{rcl}
\eta \ \leq\ \eta'' & \triangleq & \exists \eta'.\ \eta \otimes \eta' = \eta'' \\
\iota \ \leq\ \iota'' & \triangleq & \exists \iota_1' \overset{\text{rely}}{\sqsupseteq} \iota,\ \iota_2' \# \iota_1'.\ \iota_1' \otimes \iota_2' = \iota'' \\
W \ \leq\ W'' & \triangleq & \exists W_1' \overset{\text{rely}}{\sqsupseteq} W,\ W_2' \# W_1'.\ W_1' \otimes W_2' = W''
\end{array}
$$

4.5 Protocol conformance

$$
\begin{array}{lrcl}
\text{Protocol step} & \theta \vdash (s, T) \ \rightsquigarrow\ (s', T') & \triangleq & s \rightsquigarrow_\theta s',\quad \theta.\mathcal{T}(s) \uplus T = \theta.\mathcal{T}(s') \uplus T' \\
\text{Frame token set} & \mathrm{frame}_\theta(s, T) & \triangleq & (s,\ \mathrm{Token} - \theta.\mathcal{T}(s) - T) \\
\\
\text{Guarantee move} & \theta \vdash (s, T) \overset{\text{guar}}{\sqsubseteq} (s', T') & \triangleq & \theta \vdash (s, T) \rightsquigarrow^* (s', T') \\
\text{Rely move} & \theta \vdash (s, T) \overset{\text{rely}}{\sqsubseteq} (s', T') & \triangleq & \theta \vdash \mathrm{frame}_\theta(s, T) \rightsquigarrow^* \mathrm{frame}_\theta(s', T') \\
\\
\text{Island move} & (\theta, \Phi, s, T) \overset{\text{rg}}{\sqsubseteq} (\theta', \Phi', s', T') & \triangleq & \theta = \theta',\quad \Phi = \Phi',\quad \theta \vdash (s, T) \overset{\text{rg}}{\sqsubseteq} (s', T') \\
\text{World move} & (k, \omega, B) \overset{\text{rg}}{\sqsubseteq} (k', \omega', B') & \triangleq & k = k',\quad \forall i \in \mathrm{dom}(\omega).\ \omega(i) \overset{\text{rg}}{\sqsubseteq} \omega'(i),\quad B \subseteq B' \\
& \text{where rg} \ ::=\ \text{rely} \mid \text{guar} &&
\end{array}
$$

4.6 World satisfaction

$$
(\varsigma_{\mathrm{I}}, \varsigma_{\mathrm{S}}) : W, \eta \ \triangleq\ W.k > 0 \implies \begin{cases}
\eta \otimes \overline{\eta_i} = (\varsigma_{\mathrm{I}}.h,\ \varsigma_{\mathrm{S}},\ I) \\
\forall i \in \mathrm{dom}(W.\omega).\ \eta_i \in \mathrm{interp}(W.\omega(i))(\triangleright|W|) \\
\mathrm{dom}(W.B) \subseteq \mathrm{dom}(\varsigma_{\mathrm{I}}.\mathcal{E}), \\
\mathrm{rng}(W.B) \subseteq \mathrm{dom}(\varsigma_{\mathrm{S}}.\mathcal{E}), \\
I \subseteq \mathrm{dom}(\varsigma_{\mathrm{I}}.\mathcal{E})
\end{cases}
$$

4.7 Semantics of variable contexts

$$
\begin{array}{rcl}
[\![\cdot]\!]_n & \triangleq & \{\ \emptyset\ \} \\
[\![\mathcal{X}, X : \Sigma]\!]_n & \triangleq & \{\ \rho[X \mapsto V] \ \mid\ \rho \in [\![\mathcal{X}]\!]_n,\ V \in [\![\Sigma]\!]\ \} \\
[\![\mathcal{X}, p : \mathbb{P}(\Sigma)]\!]_n & \triangleq & \{\ \rho[p \mapsto \Phi] \ \mid\ \rho \in [\![\mathcal{X}]\!]_n,\ \Phi \in [\![\mathbb{P}(\Sigma)]\!]_n\ \}
\end{array}
$$

4.8 Semantics of terms

Elided, but if $\mathcal{X} \vdash M : \Sigma$ and $\rho \in [\![\mathcal{X}]\!]$ then $[\![M]\!]^\rho \in [\![\Sigma]\!]$.

4.9 Semantics of absolute propositions

Assume that there is some $\mathcal{X}$ with $\mathcal{X} \vdash A : \mathbb{B}$, $\rho \in [\![\mathcal{X}]\!]_n$. Then:

A	$[\![A]\!]^\rho$ iff
$M = N$	$[\![M]\!]^\rho = [\![N]\!]^\rho$
$\mathrm{TokPure}(P)$	$\forall W \in \mathrm{World}_n.\ \forall \eta.\ [\![P]\!]^\rho_{W,\eta} \iff [\![P]\!]^\rho_{\lvert W\rvert,\eta}$
$M \overset{\text{pure}}{\to} N$	$[\![M]\!]^\rho \overset{\text{pure}}{\to} [\![N]\!]^\rho$
$P \to_{\mathrm{S}} Q$	$\forall W \in \mathrm{World}_n, h, I, \varsigma, \varsigma_F \# \varsigma.$ if $[\![P]\!]^\rho_{W,(h,\varsigma,I)}$ then $\exists \varsigma' \# \varsigma_F.\ (\varsigma \uplus \varsigma_F) \to^* (\varsigma' \uplus \varsigma_F),\ \ [\![Q]\!]^\rho_{W,(h,\varsigma',I)}$
$M \overset{\text{rely}}{\sqsubseteq}_\pi N$	$\pi.\theta \vdash [\![M]\!]^\rho \overset{\text{rely}}{\sqsubseteq} [\![N]\!]^\rho$
$M \overset{\text{guar}}{\sqsubseteq}_\pi N$	$\pi.\theta \vdash [\![M]\!]^\rho \overset{\text{guar}}{\sqsubseteq} [\![N]\!]^\rho$

4.10 Semantics of propositions

Assume that there is some $\mathcal{X}$ with $\mathcal{X} \vdash R : \mathbb{B}$, $\rho \in [\![\mathcal{X}]\!]_n$, $W \in \mathrm{World}_n$. Then:

R	$[\![R]\!]^\rho_{W,\eta}$ iff
A	$[\![A]\!]^\rho$
$P \wedge Q$	$[\![P]\!]^\rho_{W,\eta} \wedge [\![Q]\!]^\rho_{W,\eta}$
$P \vee Q$	$[\![P]\!]^\rho_{W,\eta} \vee [\![Q]\!]^\rho_{W,\eta}$
$P \Rightarrow Q$	$\forall W' \geq W,\ \eta' \geq \eta.\ [\![P]\!]^\rho_{W',\eta'} \Rightarrow [\![Q]\!]^\rho_{W',\eta'}$
$\forall X \in \Sigma.\ P$	$\forall V \in [\![\Sigma]\!].\ [\![P]\!]^{\rho[X \mapsto V]}_{W,\eta}$
$\exists X \in \Sigma.\ P$	$\exists V \in [\![\Sigma]\!].\ [\![P]\!]^{\rho[X \mapsto V]}_{W,\eta}$
$\forall \alpha \in \mathbb{P}(\Sigma).\ P$	$\forall \Phi \in [\![\mathbb{P}(\Sigma)]\!]_{W.k}.\ [\![P]\!]^{\rho[\alpha \mapsto \Phi]}_{W,\eta}$
$\exists \alpha \in \mathbb{P}(\Sigma).\ P$	$\exists \Phi \in [\![\mathbb{P}(\Sigma)]\!]_{W.k}.\ [\![P]\!]^{\rho[\alpha \mapsto \Phi]}_{W,\eta}$
$\Box P$	$[\![P]\!]^\rho_{\lvert W\rvert,\emptyset}$
$\triangleright P$	$W.k > 0 \Rightarrow [\![P]\!]^\rho_{\triangleright W,\eta}$
$P(M)$	$[\![M]\!]^\rho \in [\![P]\!]^\rho_{W,\eta}$
$M \hookrightarrow_{\mathrm{I}} N$	$\eta.h([\![M]\!]^\rho) = [\![N]\!]^\rho$
$M \hookrightarrow_{\mathrm{S}} N$	$\eta.\varsigma.h([\![M]\!]^\rho) = [\![N]\!]^\rho$
$M \Mapsto_{\mathrm{S}} N$	$\eta.\varsigma.\mathcal{E}([\![M]\!]^\rho) = [\![N]\!]^\rho$
$\mathit{Tid}(M)$	$[\![M]\!]^\rho \in \eta.I$
$M \bowtie N$	$([\![M]\!]^\rho, [\![N]\!]^\rho) \in W.B$
$P_1 * P_2$	$W = W_1 \otimes W_2,\ \eta = \eta_1 \otimes \eta_2,\ [\![P_i]\!]^\rho_{W_i,\eta_i}$
$\boxed{M}^N_{(\theta,\varphi)}$	$W.\omega([\![N]\!]^\rho) \geq (\theta,\ [\![\varphi]\!]^\rho_-,\ [\![M]\!]^\rho)$

In addition, we extend the syntax of propositions as follows:

$$P \ ::= \ \cdots \mid \mathsf{safe}(M, N, \varphi)$$

and treat concurrent triples as *sugar*:

$$\{P\}\ M \Mapsto N\ \{\varphi\} \ \triangleq \ \Box(P \Rightarrow \mathsf{safe}([M \mapsto N], M, \varphi))$$

The semantics of atomic triples and safe are given below.

4.11 Semantics of predicates

Assume that there is some $\mathcal{X}$ with $\mathcal{X} \vdash \varphi : \mathbb{P}(\Sigma)$, $\rho \in [\![\mathcal{X}]\!]_n$, $W \in \mathrm{World}_n$. Then for any $V \in [\![\Sigma]\!]$:

φ	$V \in [\![\varphi]\!]^\rho_{W,\eta}$ iff
$X \in \Sigma.\ P$	$[\![P]\!]^{\rho[X \mapsto V]}_{W,\eta}$
$\mu\alpha \in \mathbb{P}(\Sigma).\ \psi$	$V \in [\![\psi[\mu\alpha.P/\alpha]]\!]^\rho_{W,\eta}$
α	$V \in \rho(\alpha)$

4.12 Safety

$[\![\mathsf{safe}(M, N, \varphi)]\!]^\rho_{W_0,\eta} \triangleq \forall W \overset{\text{rely}}{\sqsupseteq} W_0,\ \eta_F \# \eta,\ \mathcal{E} = [\![M]\!]^\rho,\ \mathcal{E}_F \# \mathcal{E}.$

if $W.k > 0$ and $(h; \mathcal{E} \uplus \mathcal{E}_F,\ \varsigma_\mathrm{s}) : W, \eta \otimes \eta_F$ and $\mathcal{E}' \# \mathcal{E}_F$ then:

$h; \mathcal{E} \to h'; \mathcal{E}' \implies \exists \varsigma'_\mathrm{s}, \eta', W' \overset{\text{guar}}{\sqsupseteq_1} W.\ \varsigma_\mathrm{s} \to^* \varsigma'_\mathrm{s},\ (h'; \mathcal{E}' \uplus \mathcal{E}_F,\ \varsigma'_\mathrm{s}) : W', \eta' \otimes \eta_F,\ [\![\mathsf{safe}(\mathcal{E}', N, \varphi)]\!]^\rho_{W',\eta'}$

$\mathcal{E} = \mathcal{E}_0 \uplus [[\![N]\!]^\rho \mapsto v] \implies \exists \varsigma'_\mathrm{s}, \eta', W' \overset{\text{guar}}{\sqsupseteq_0} W.\ \varsigma_\mathrm{s} \to^* \varsigma'_\mathrm{s},\ (h; \mathcal{E} \uplus \mathcal{E}_F,\ \varsigma'_\mathrm{s}) : W', \eta' \otimes \eta_F,\ [\![\mathsf{safe}(\mathcal{E}_0, N, \mathrm{True}) * \varphi(v)]\!]^\rho_{W',\eta'}$

4.13 Atomic triples

For implementation code, the semantics of atomic triples is:

$$[\![(\!|P|\!)\ M \Mapsto_\mathrm{i} N\ (\!|\varphi|\!)]\!]^\rho_{W_0,\eta_0} \triangleq \forall W \geq |W_0|, \eta, \eta_F \# \eta.$$
$$\text{if } W.k > 0 \text{ and } [\![P]\!]^\rho_{\triangleright W,\eta} \text{ and } (\eta \otimes \eta_F).h; [\![N]\!]^\rho \xrightarrow{i} h'; v \text{ then } \exists \eta' \# \eta_F.$$
$$h' = (\eta' \otimes \eta_F).h,\quad (\eta \otimes \eta_F).\varsigma \to^* (\eta' \otimes \eta_F).\varsigma,\quad \eta.I = \eta'.I,\quad [\![\varphi(v)]\!]^\rho_{\triangleright W,\eta'}$$

For spec code, we treat the atomic triple as sugar:

$$(\!|P|\!)\ M \Mapsto_\mathrm{s} N\ (\!|\varphi|\!) \triangleq \forall \kappa.\ (P * M \Mapsto_\mathrm{s} \kappa[N]) \to_\mathrm{s} (\exists x.\ \varphi(x) * M \Mapsto_\mathrm{s} \kappa[x])$$

5 Logic: proof theory

5.1 Judgments

$$\begin{array}{rcl} \mathcal{P} & \in & \wp_{\text{fin}}(\text{Prop}) \\ \mathcal{C} & ::= & \mathcal{X}; \mathcal{P} \end{array}$$

$$\begin{array}{rcl} \mathcal{X}; \mathcal{P} \vdash Q & \triangleq & (\forall P \in \mathcal{P}.\ \mathcal{X} \vdash P : \mathbb{B}), \\ & & \mathcal{X} \vdash Q : \mathbb{B}, \\ & & \forall n.\ \forall \rho \in [\![\mathcal{X}]\!].\ \forall W \in \text{World}_n,\ \forall \eta.\ \text{if } (\forall P \in \mathcal{P}.\ [\![P]\!]^{\rho}_{W,\eta}) \text{ then } [\![Q]\!]^{\rho}_{W,\eta} \end{array}$$

Proof rules implicitly assume well-sortedness.

5.2 Laws of intuitionistic second-order logic with recursion

$$\frac{P \in \mathcal{C}}{\mathcal{C} \vdash P}\ \textsc{Asm} \qquad \frac{\mathcal{C} \vdash P(M) \qquad \mathcal{C} \vdash M = M'}{\mathcal{C} \vdash P(M')}\ \textsc{Eq} \qquad \frac{\mathcal{C} \vdash P \qquad \mathcal{C} \vdash Q}{\mathcal{C} \vdash P \wedge Q}\ \wedge\text{I} \qquad \frac{\mathcal{C} \vdash P \wedge Q}{\mathcal{C} \vdash P}\ \wedge\text{EL}$$

$$\frac{\mathcal{C} \vdash P \wedge Q}{\mathcal{C} \vdash Q}\ \wedge\text{ER} \qquad \frac{\mathcal{C} \vdash P \vee Q \qquad \mathcal{C}, P \vdash R \qquad \mathcal{C}, Q \vdash R}{\mathcal{C} \vdash R}\ \vee\text{E} \qquad \frac{\mathcal{C} \vdash P}{\mathcal{C} \vdash P \vee Q}\ \vee\text{IL} \qquad \frac{\mathcal{C} \vdash Q}{\mathcal{C} \vdash P \vee Q}\ \vee\text{IR}$$

$$\frac{\mathcal{C}, P \vdash Q}{\mathcal{C} \vdash P \Rightarrow Q}\ \Rightarrow\text{I} \qquad \frac{\mathcal{C} \vdash P \Rightarrow Q \qquad \mathcal{C} \vdash P}{\mathcal{C} \vdash Q}\ \Rightarrow\text{E} \qquad \frac{\mathcal{C}, X : \Sigma \vdash P}{\mathcal{C} \vdash \forall X \in \Sigma.\ P}\ \forall_1\text{I} \qquad \frac{\mathcal{C} \vdash \forall X \in \Sigma.\ P \qquad \mathcal{C} \vdash M : \Sigma}{\mathcal{C} \vdash P[M/X]}\ \forall_1\text{E}$$

$$\frac{\mathcal{C} \vdash \exists X \in \Sigma.\ P \qquad \mathcal{C}, X : \Sigma, P \vdash Q}{\mathcal{C} \vdash Q}\ \exists_1\text{E} \qquad \frac{\mathcal{C} \vdash P[M/X] \qquad \mathcal{C} \vdash M : \Sigma}{\mathcal{C} \vdash \exists X : \Sigma.P}\ \exists_1\text{I} \qquad \frac{\mathcal{C}, \alpha : \mathbb{P}(\Sigma) \vdash P}{\mathcal{C} \vdash \forall \alpha \in \mathbb{P}(\Sigma).\ P}\ \forall_2\text{I}$$

$$\frac{\mathcal{C} \vdash \forall \alpha.P \qquad \mathcal{C} \vdash Q : \mathbb{B}}{\mathcal{C} \vdash P[Q/\alpha]}\ \forall_2\text{E} \qquad \frac{\mathcal{C} \vdash \exists \alpha \in \mathbb{P}(\Sigma).P \qquad \mathcal{C}, \alpha : \mathbb{P}(\Sigma), P \vdash Q}{\mathcal{C} \vdash Q}\ \exists_2\text{E}$$

$$\frac{\mathcal{C} \vdash P[Q/\alpha] \qquad \mathcal{C} \vdash Q : \mathbb{B}}{\mathcal{C} \vdash \exists \alpha.P}\ \exists_2\text{I} \qquad \frac{\mathcal{C} \vdash M \in (X \in \Sigma).P}{\mathcal{C} \vdash P[M/X]}\ \textsc{Elem} \qquad \frac{\mathcal{C} \vdash M \in (\mu\alpha \in \mathbb{P}(\Sigma).\varphi)}{\mathcal{C} \vdash M \in \varphi[\mu\alpha \in \mathbb{P}(\Sigma).\varphi/\alpha]}\ \textsc{Elem-}\mu$$

5.3 Axioms from the logic of (affine) bunched implications

$$\begin{array}{rcl} P * Q & \iff & Q * P \\ (P * Q) * R & \iff & P * (Q * R) \\ P * Q & \Rightarrow & P \end{array} \qquad \begin{array}{rcl} (P \vee Q) * R & \iff & (P * R) \vee (Q * R) \\ (P \wedge Q) * R & \Longrightarrow & (P * R) \wedge (Q * R) \\ (\exists x.\ P) * Q & \iff & \exists x.\ (P * Q) \\ (\forall x.\ P) * Q & \Longrightarrow & \forall x.\ (P * Q) \end{array}$$

$$\frac{\mathcal{C}, P_1 \vdash Q_1 \qquad \mathcal{C}, P_2 \vdash Q_2}{\mathcal{C}, P_1 * P_2 \vdash Q_1 * Q_2}$$

5.4 Laws for island assertions

$$\boxed{s_1, T_1}^{n}_{\pi} * \boxed{s_2, T_2}^{n}_{\pi} \quad \Leftrightarrow \quad \exists s.\ \boxed{s, T_1 \uplus T_2}^{n}_{\pi} * (s, T_1) \overset{\text{rely}}{\sqsupseteq}_{\pi} (s_1, T_1) * (s, T_2) \overset{\text{rely}}{\sqsupseteq}_{\pi} (s_2, T_2)$$

5.5 Laws for the later modality

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

$$\begin{array}{rcl@{\qquad}rcl} \triangleright \Box P & \iff & \Box \triangleright P & \triangleright \forall X \in \Sigma. P & \iff & \forall X \in \Sigma. \triangleright P \\ \triangleright (P \wedge Q) & \iff & \triangleright P \wedge \triangleright Q & \triangleright \exists X \in \Sigma. P & \iff & \exists X \in \Sigma. \triangleright P \\ \triangleright (P \vee Q) & \iff & \triangleright P \vee \triangleright Q & \triangleright (P * Q) & \iff & \triangleright P * \triangleright Q \end{array}$$

5.6 Laws for the always modality

$$\frac{\mathcal{X}, \Box \mathcal{P} \vdash \Box P}{\mathcal{X}, \Box \mathcal{P}, \mathcal{P}' \vdash \Box P}\ \Box\text{I} \qquad\qquad \Box P \ \Rightarrow\ \Box P * P$$

The following propositions P are "completely stable", *i.e.,* they enjoy the axiom $P \Leftrightarrow \Box P$:

$$A \qquad i \bowtie j \qquad (\!|P|\!)\ M \Mapsto_{\text{IS}} N\ (\!|\varphi|\!) \qquad \{P\}\ M \Mapsto N\ \{\varphi\} \qquad \boxed{M, \emptyset}^{N}_{\pi}$$

5.7 Atomic Hoare logic

5.7.1 Small axioms

$$(\!|\text{True}|\!)\ i \Mapsto_{\text{IS}} \textsf{new}\ v\ (\!|\textsf{ret. ret} \hookrightarrow_{\text{IS}} v|\!)\ \textsc{New} \qquad (\!|v \hookrightarrow_{\text{IS}} v'|\!)\ i \Mapsto_{\text{IS}} \textsf{get}\ v\ (\!|\textsf{ret. ret} = v' * v \hookrightarrow_{\text{IS}} v'|\!)\ \textsc{Get}$$

$$(\!|v \hookrightarrow_{\text{IS}} -|\!)\ i \Mapsto_{\text{IS}} v\ :=\ v'\ (\!|\textsf{ret. ret} = () * v \hookrightarrow_{\text{IS}} v'|\!)\ \textsc{Set}$$

$$(\!|v \hookrightarrow_{\text{IS}} v_o|\!)\ i \Mapsto_{\text{IS}} \textsf{CAS}(v, v_o, v_n)\ (\!|\textsf{ret. ret} = \textsf{true} * v \hookrightarrow_{\text{IS}} v_n|\!)\ \text{CAS}_{\textsf{true}}$$

$$(\!|v \hookrightarrow_{\text{IS}} v' * v' \neq v_o|\!)\ i \Mapsto_{\text{IS}} \textsf{CAS}(v, v_o, v_n)\ (\!|\textsf{ret. ret} = \textsf{false} * v \hookrightarrow_{\text{IS}} v'|\!)\ \text{CAS}_{\textsf{false}}$$

$$(\!|\text{True}|\!)\ i \Mapsto_{\text{IS}} \textsf{newLcl}\ (\!|\textsf{ret. ret} \hookrightarrow_{\text{IS}} \emptyset|\!)\ \textsc{NewLcl}$$

$$(\!|v \hookrightarrow_{\text{IS}} L|\!)\ i \Mapsto_{\text{IS}} \textsf{getLcl}(v)\ (\!|\textsf{ret. ret} = \lfloor L \rfloor(i) * v \hookrightarrow_{\text{IS}} L|\!)\ \textsc{GetLcl}$$

$$(\!|v \hookrightarrow_{\text{IS}} L|\!)\ i \Mapsto_{\text{IS}} \textsf{setLcl}(v, v')\ (\!|\textsf{ret. ret} = () * v \hookrightarrow_{\text{IS}} L[i := v']|\!)\ \textsc{SetLcl}$$

5.7.2 Spec rewriting

$$\frac{\mathcal{C} \vdash e \overset{\text{pure}}{\to} e'}{\mathcal{C} \vdash (P * j \Mapsto_{\text{S}} K[e]) \to_{\text{S}} (P * j \Mapsto_{\text{S}} K[e'])}\ \textsc{SPure} \qquad \frac{\mathcal{C} \vdash (\!|P|\!)\ j \Mapsto_{\text{S}} a\ (\!|x.\ Q|\!)}{\mathcal{C} \vdash (P * j \Mapsto_{\text{S}} K[a]) \to_{\text{S}} (\exists x.\ Q * j \Mapsto_{\text{S}} K[x])}\ \textsc{SPrim}$$

$$\frac{}{\mathcal{C} \vdash (P * j \Mapsto_{\text{S}} K[\textsf{fork}\ e]) \to_{\text{S}} (P * j \Mapsto_{\text{S}} K[()] * k \Mapsto_{\text{S}} e)}\ \textsc{SFork}$$

5.7.3 Structural and glue rules

$$\frac{\vdash P \Rightarrow P' \qquad \mathcal{C} \vdash (\!|P'|\!)\ i \Mapsto_{\text{IS}} a\ (\!|x.\ Q'|\!) \qquad x \vdash Q' \Rightarrow Q}{\mathcal{C} \vdash (\!|P|\!)\ i \Mapsto_{\text{IS}} a\ (\!|x.\ Q|\!)}\ \textsc{ACsq} \qquad \frac{\mathcal{C} \vdash (\!|P|\!)\ i \Mapsto_{\text{IS}} a\ (\!|x.\ Q|\!)}{\mathcal{C} \vdash (\!|P * R|\!)\ i \Mapsto_{\text{IS}} a\ (\!|x.\ Q * R|\!)}\ \textsc{AFrame}$$

$$\frac{\mathcal{C} \vdash (\!|P_1|\!)\ i \Mapsto_{\text{IS}} a\ (\!|\varphi|\!) \qquad \mathcal{C} \vdash (\!|P_2|\!)\ i \Mapsto_{\text{IS}} a\ (\!|\varphi|\!)}{\mathcal{C} \vdash (\!|P_1 \vee P_2|\!)\ i \Mapsto_{\text{IS}} a\ (\!|\varphi|\!)}\ \textsc{ADisj} \qquad \frac{\mathcal{C} \vdash (\!|P|\!)\ i \Mapsto_{\text{I}} a\ (\!|x.\ Q|\!) \qquad \mathcal{C} \vdash Q \to_{\text{S}} R}{\mathcal{C} \vdash (\!|P|\!)\ i \Mapsto_{\text{I}} a\ (\!|x.\ R|\!)}\ \textsc{AExecSpec}$$

5.8 Concurrent Hoare logic

5.8.1 Shorthand

A bit of shorthand:

$$\pi[\![M]\!] \ \triangleq\ \exists s, T.\ M = (s, T) \wedge \pi.\varphi(s)$$

5.8.2 Rules

$$\frac{\mathcal{C} \vdash (\!|P|\!)\ i \Mapsto_{\mathrm{I}} a\ (\!|Q|\!)}{\mathcal{C} \vdash \{\triangleright P\}\ i \Mapsto a\ \{Q\}}\ \textsc{Private} \qquad \frac{\mathcal{C} \vdash \forall b \overset{\text{rely}}{\sqsupseteq_\pi} b_0.\ \exists b' \overset{\text{guar}}{\sqsupseteq_\pi} b.\ (\!|\pi[\![b]\!] * P|\!)\ i \Mapsto_{\mathrm{I}} a\ (\!|x.\ \triangleright \pi[\![b']\!] * Q|\!)}{\mathcal{C} \vdash \left\{\boxed{b_0}^n_\pi * \triangleright P\right\}\ i \Mapsto a\ \left\{x.\ \exists b'.\ \boxed{b'}^n_\pi * Q\right\}}\ \textsc{UpdIsl}$$

$$\frac{\mathcal{C} \vdash e \overset{\text{pure}}{\to} e' \qquad \mathcal{C} \vdash \{P\}\ i \Mapsto e'\ \{\varphi\}}{\mathcal{C} \vdash \{\triangleright P\}\ i \Mapsto e\ \{\varphi\}}\ \textsc{Pure} \qquad \frac{\mathcal{C} \vdash \{P\}\ i \Mapsto e\ \{x.\ Q * \triangleright \pi[\![b]\!]\}}{\mathcal{C} \vdash \{P\}\ i \Mapsto e\ \left\{x.\ Q * \exists n.\ \boxed{b}^n_\pi\right\}}\ \textsc{NewIsl}$$

$$\frac{\mathcal{C} \vdash \{P\}\ i \Mapsto e\ \{x.\ Q\} \qquad \mathcal{C} \vdash Q \to_{\mathrm{s}} R}{\mathcal{C} \vdash \{P\}\ i \Mapsto e\ \{x.\ R\}}\ \textsc{ExecSpec}$$

$$\frac{\mathcal{C} \vdash \{P\}\ i \Mapsto e\ \{x.\ Q\} \qquad \mathcal{C}, x \vdash \{Q\}\ i \Mapsto K[x]\ \{R\}}{\mathcal{C} \vdash \{P\}\ i \Mapsto K[e]\ \{R\}}\ \textsc{Bind} \qquad \frac{}{\mathcal{C} \vdash \{\mathrm{True}\}\ i \Mapsto v\ \{x.\ x = v\}}\ \textsc{Ret}$$

$$\frac{\mathcal{C} \vdash \{P * \mathit{Tid}(j)\}\ j \Mapsto e\ \{x.\ x = ()\}}{\mathcal{C} \vdash \{P\}\ i \Mapsto \mathbf{fork}\ e\ \{x.\ x = ()\}}\ \textsc{Fork} \qquad \frac{\mathcal{C}, j \bowtie j' \vdash \{P * \mathit{Tid}(j) * j' \Mapsto_{\mathrm{s}} e_{\mathrm{s}}\}\ j \Mapsto e_{\mathrm{I}}\ \{x.\ x = ()\}}{\mathcal{C} \vdash \{P * i' \Mapsto_{\mathrm{s}} K[\mathbf{fork}\ e_{\mathrm{s}}]\}\ i \Mapsto \mathbf{fork}\ e_{\mathrm{I}}\ \{i' \Mapsto_{\mathrm{s}} K[()]\}}\ \textsc{ForkS}$$

$$\frac{\vdash P \Rightarrow P' \qquad \mathcal{C} \vdash \{P'\}\ i \Mapsto e\ \{x.\ Q'\} \qquad x \vdash Q' \Rightarrow Q}{\mathcal{C} \vdash \{P\}\ i \Mapsto e\ \{x.\ Q\}}\ \textsc{Csq}$$

$$\frac{\mathcal{C} \vdash \{P_1\}\ i \Mapsto e\ \{\varphi\} \qquad \mathcal{C} \vdash \{P_2\}\ i \Mapsto e\ \{\varphi\}}{\mathcal{C} \vdash \{P_1 \vee P_2\}\ i \Mapsto e\ \{\varphi\}}\ \textsc{Disj} \qquad \frac{\mathcal{C}, X \vdash \{P\}\ i \Mapsto e\ \{\varphi\}}{\mathcal{C} \vdash \{\exists X.\ P\}\ i \Mapsto e\ \{\varphi\}}\ \textsc{Ex}$$

$$\frac{\mathcal{C} \vdash \{P\}\ i \Mapsto e\ \{x.\ Q\}}{\mathcal{C} \vdash \{P * R\}\ i \Mapsto e\ \{x.\ Q * R\}}\ \textsc{Frame}$$

$$\frac{\mathcal{C} \vdash \Box Q \qquad \mathcal{C} \vdash \{P * \Box Q\}\ i \Mapsto e\ \{\varphi\}}{\mathcal{C} \vdash \{P\}\ i \Mapsto e\ \{\varphi\}}\ \Box\textsc{In} \qquad \frac{\mathcal{C}, \Box Q \vdash \{P\}\ i \Mapsto e\ \{\varphi\}}{\mathcal{C} \vdash \{P * \Box Q\}\ i \Mapsto e\ \{\varphi\}}\ \Box\textsc{Out}$$

6 Logic: derived notions

6.1 Derived assertions

$$\{P\}\ e\ \{\varphi\} \quad \triangleq \quad \forall i.\ \{P\}\ i \Mapsto e\ \{\varphi\}$$

6.2 Derived rules

$$\frac{\mathcal{C}, f, \forall x.\ \{\triangleright P\}\ i \Mapsto f\ x\ \{\varphi\} \vdash \forall x.\ \{P\}\ i \Mapsto e\ \{\varphi\}}{\mathcal{C} \vdash \forall x.\ \{\triangleright P\}\ i \Mapsto (\mathbf{rec}\ f(x).e)\ x\ \{\varphi\}}\ \textsc{Rec}$$

$$\frac{\begin{array}{c}\mathcal{C} \vdash \{P\}\ i \Mapsto e\ \{x.\ \exists y.\ (x = \mathbf{inj}_1\ y * \triangleright Q_1) \vee (x = \mathbf{inj}_2\ y * \triangleright Q_2)\} \\ \mathcal{C}, y \vdash \{x = \mathbf{inj}_1\ y * Q_1\}\ i \Mapsto e_1\ \{\varphi\} \qquad \mathcal{C}, y \vdash \{x = \mathbf{inj}_2\ y * Q_2\}\ i \Mapsto e_2\ \{\varphi\}\end{array}}{\mathcal{C} \vdash \{P\}\ i \Mapsto \mathbf{case}(e, \mathbf{inj}_1\ y \Rightarrow e_1, \mathbf{inj}_2\ y \Rightarrow e_2)\ \{\varphi\}}$$

6.3 Encoding refinement

$$
\begin{array}{rcl}
\mathrm{Inv}(P) & \triangleq & \exists n.\ \boxed{\mathsf{dummy}, \emptyset}^{n}_{(\theta,\ (s).P)} \text{ where } \theta = (\emptyset, \lambda s.\ \emptyset) \\
\mathrm{Type}(\varphi) & \triangleq & \Box\forall(x_{\mathrm{I}}, x_{\mathrm{S}}) \in \varphi.\ \Box(x_{\mathrm{I}}, x_{\mathrm{S}}) \in \varphi \\
(e_{\mathrm{I}}, e_{\mathrm{S}}) \downarrow \varphi & \triangleq & \forall(i \bowtie j), \kappa.\ \{\mathit{Tid}(i) * j \Rrightarrow_{\mathrm{S}} \kappa[e_{\mathrm{S}}]\}\ i \Rrightarrow e_{\mathrm{I}}\ \{x_{\mathrm{I}}.\ \exists x_{\mathrm{S}}.\ \varphi(x_{\mathrm{I}}, x_{\mathrm{S}}) * \mathit{Tid}(i) * j \Rrightarrow_{\mathrm{S}} \kappa[x_{\mathrm{S}}]\}
\end{array}
$$

$$
\begin{array}{rcl}
[\![\mathbf{1}]\!] & \triangleq & (x_{\mathrm{I}}, x_{\mathrm{S}}).\ x_{\mathrm{I}} = x_{\mathrm{S}} = () \\
[\![\mathbf{B}]\!] & \triangleq & (x_{\mathrm{I}}, x_{\mathrm{S}}).\ (x_{\mathrm{I}} = x_{\mathrm{S}} = \mathbf{true}) \vee (x_{\mathrm{I}} = x_{\mathrm{S}} = \mathbf{false}) \\
[\![\tau_1 \times \tau_2]\!] & \triangleq & (x_{\mathrm{I}}, x_{\mathrm{S}}).\quad (\mathbf{prj}_1\ x_{\mathrm{I}}, \mathbf{prj}_1\ x_{\mathrm{S}}) \downarrow [\![\tau_1]\!] \\
 & & \qquad \wedge\ (\mathbf{prj}_2\ x_{\mathrm{I}}, \mathbf{prj}_2\ x_{\mathrm{S}}) \downarrow [\![\tau_2]\!] \\
[\![\tau_1 + \tau_2]\!] & \triangleq & (x_{\mathrm{I}}, x_{\mathrm{S}}).\quad (\triangleright\exists(y_{\mathrm{I}}, y_{\mathrm{S}}) \in [\![\tau_1]\!].\ x_{\mathrm{I}} = \mathbf{inj}_1\ y_{\mathrm{I}} \wedge x_{\mathrm{S}} = \mathbf{inj}_1\ y_{\mathrm{S}}) \\
 & & \qquad \vee\ (\triangleright\exists(y_{\mathrm{I}}, y_{\mathrm{S}}) \in [\![\tau_2]\!].\ x_{\mathrm{I}} = \mathbf{inj}_2\ y_{\mathrm{I}} \wedge x_{\mathrm{S}} = \mathbf{inj}_2\ y_{\mathrm{S}}) \\
[\![\tau \to \tau']\!] & \triangleq & (x_{\mathrm{I}}, x_{\mathrm{S}}).\ \Box\forall y_{\mathrm{I}}, y_{\mathrm{S}}.\ (\triangleright(y_{\mathrm{I}}, y_{\mathrm{S}}) \in [\![\tau]\!]) \Rightarrow (x_{\mathrm{I}}\ y_{\mathrm{I}}, x_{\mathrm{S}}\ y_{\mathrm{S}}) \downarrow [\![\tau']\!] \\
[\![\forall\alpha.\tau]\!] & \triangleq & (x_{\mathrm{I}}, x_{\mathrm{S}}).\ \Box\forall\alpha \in \mathbb{P}(\mathrm{Val} \times \mathrm{Val}).\ \mathrm{Type}(\alpha) \Rightarrow (x_{\mathrm{I}}\ \_, x_{\mathrm{S}}\ \_) \downarrow [\![\tau]\!] \\
[\![\mathbf{ref}\ \tau]\!] & \triangleq & (x_{\mathrm{I}}, x_{\mathrm{S}}).\ \mathrm{Inv}(\exists(y_{\mathrm{I}}, y_{\mathrm{S}}) \in [\![\tau]\!].\ x_{\mathrm{I}} \hookrightarrow_{\mathrm{I}} y_{\mathrm{I}} * x_{\mathrm{S}} \hookrightarrow_{\mathrm{S}} y_{\mathrm{S}}) \\
[\![\mathbf{refLcl}\ \tau]\!] & \triangleq & (x_{\mathrm{I}}, x_{\mathrm{S}}).\ \mathrm{Inv}\left(\begin{array}{l}\exists L_{\mathrm{I}}, L_{\mathrm{S}}.\ x_{\mathrm{I}} \hookrightarrow_{\mathrm{I}} L_{\mathrm{I}} * x_{\mathrm{S}} \hookrightarrow_{\mathrm{S}} L_{\mathrm{S}} \\ \quad \wedge\ \forall(i \bowtie j).\ (\lfloor L_{\mathrm{I}} \rfloor(i), \lfloor L_{\mathrm{S}} \rfloor(j)) \in [\![\mathbf{1} + \tau]\!]\end{array}\right) \\
[\![\alpha]\!] & \triangleq & \alpha \\
[\![\mu\alpha.\tau]\!] & \triangleq & \mu\alpha.[\![\tau]\!]
\end{array}
$$

Logical relatedness $\boxed{\Delta; \Gamma \vdash e_{\mathrm{I}} \preceq e_{\mathrm{S}} : \tau}$

$$
\overline{\alpha}; \overline{x : \tau} \vdash e_{\mathrm{I}} \preceq e_{\mathrm{S}} : \tau \quad \triangleq \quad \overline{\alpha \in \mathbb{P}(\mathrm{Val} \times \mathrm{Val})}, \overline{x_{\mathrm{I}}, x_{\mathrm{S}} \in \mathrm{Val}}; \overline{\mathrm{Type}(\alpha)}, \overline{(x_{\mathrm{I}}, x_{\mathrm{S}}) \in [\![\tau]\!]} \vdash (e_{\mathrm{I}}\overline{[x_{\mathrm{I}}/x]}, e_{\mathrm{S}}\overline{[x_{\mathrm{S}}/x]}) \downarrow [\![\tau]\!]
$$

In order to prove congruence, we need the following proof rule, which internalizes the fact that the world asserted in the precondition is assumed to be inhabited (and thus cannot contain two islands claiming to own a single heap location):

$$
\frac{\mathcal{C} \vdash \{P * (x_{\mathrm{I}} = y_{\mathrm{I}} \Leftrightarrow x_{\mathrm{S}} = y_{\mathrm{S}})\}\ i \Rrightarrow e\ \{\varphi\}}{\mathcal{C} \vdash \{P * (x_{\mathrm{I}}, x_{\mathrm{S}}) \in [\![\sigma]\!] * (y_{\mathrm{I}}, y_{\mathrm{S}}) \in [\![\sigma]\!])\}\ i \Rrightarrow e\ \{\varphi\}}\ \textsc{Comparable}
$$

7 Example specs and proofs

7.1 Pure lists

$$
\begin{array}{rcl}
\mathsf{list}(\alpha) & \triangleq & \mu\beta.\ \mathbf{1} + (\alpha \times \beta) \\
\mathsf{foreach} & : & \forall\alpha.\ (\alpha \to \mathbf{1}) \to \mathsf{list}(\alpha) \to \mathbf{1} \\
\mathsf{foreach} & \triangleq & \Lambda\alpha.\ \lambda f.\ \textbf{rec}\ \mathsf{loop}(l).\ \textbf{case}\ l\ \textbf{of}\ \mathsf{none} \Rightarrow () \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad \mid\ \mathsf{some}(x, n) \Rightarrow f(x);\ \mathsf{loop}(n) \\
\mathrm{Map}(\varphi) & \triangleq & \mu p \in \mathbb{P}(\mathrm{Val}).\ (x \in \mathrm{Val}).\ x = \mathsf{none} \vee (\exists y, z.\ x = \mathsf{some}(y, z) * \varphi(y) * \triangleright p(z))
\end{array}
$$

Spec for foreach

$$
\begin{array}{l}
\forall p, q \in \mathbb{P}(\mathrm{Val}).\ \forall r \in \mathbb{P}(\mathbf{1}).\ \forall f.\ (\forall x.\ \{p(x) * r\}\ f(x)\ \{\mathsf{ret}.\ \mathsf{ret} = () * q(x) * r\}) \\
\quad \Rightarrow \quad \forall l.\ \{\mathrm{Map}(p)(l) * r\}\ \mathsf{foreach}\ \_\ f\ l\ \{\mathsf{ret}.\ \mathsf{ret} = () * \mathrm{Map}(q)(l) * r\}
\end{array}
$$

Verification of foreach

$\Lambda.\ \lambda f.\ \textbf{rec}\ \mathsf{loop}(l).$

Prop context:	**Variables:** $f, p, q, r, l, \mathsf{loop}$
$\forall x.\ \{p(x) * r\}\ f(x)\ \{\mathsf{ret}.\ \mathsf{ret} = () * q(x) * r\}$	
$\forall n.\ \{\triangleright(\mathrm{Map}(p)(n) * r)\}\ \mathsf{loop}\ n\ \{\mathsf{ret}.\ \mathsf{ret} = () * \mathrm{Map}(q)(n) * r\}$	

$\{\mathrm{Map}(p)(l) * r\}$
case l
of $\mathsf{none} \Rightarrow$
$\{l = \mathsf{none} * \mathrm{Map}(p)(l) * r\}$
$\{\mathrm{Map}(q)(l) * r\}$
$()$
$\{\mathsf{ret}.\ \mathsf{ret} = () * \mathrm{Map}(q)(l) * r\}$
$\mid \mathsf{some}(x, n) \Rightarrow$
$\{l = \mathsf{some}(x, n) * \mathrm{Map}(p)(l) * r\}$
$\{l = \mathsf{some}(x, n) * p(x) * \triangleright\mathrm{Map}(p)(n) * r\}$
$f(x);$
$\{l = \mathsf{some}(x, n) * q(x) * \triangleright\mathrm{Map}(p)(n) * r\}$
$\mathsf{loop}(n)$
$\{\mathsf{ret}.\ \mathsf{ret} = () * l = \mathsf{some}(x, n) * q(x) * \mathrm{Map}(q)(n) * r\}$
$\{\mathsf{ret}.\ \mathsf{ret} = () * \mathrm{Map}(q)(l) * r\}$

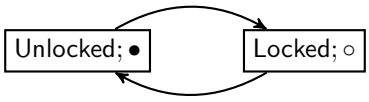
7.2 Locking

$$
\begin{array}{rcl}
\mathsf{tryAcq} & \triangleq & \lambda x.\ \mathbf{CAS}(x, \mathbf{false}, \mathbf{true}) \\
\mathsf{acq} & \triangleq & \mathbf{rec}\ \mathsf{loop}(x).\ \mathbf{if}\ \mathsf{tryAcq}(x)\ \mathbf{then}\ ()\ \mathbf{else}\ \mathsf{loop}(x) \\
\mathsf{rel} & \triangleq & \lambda x.\ x\ :=\ \mathbf{false} \\
\mathsf{withLock}(f, \mathsf{lock}) & \triangleq & \lambda x.\ \mathsf{acq}(\mathsf{lock}); \mathbf{let}\ z = f(x)\ \mathbf{in}\ \mathsf{rel}(\mathsf{lock});\ z \\
\mathsf{mkSync} & \triangleq & \lambda().\mathbf{let}\ \mathsf{lock} = \mathbf{new}\ \mathbf{false}\ \mathbf{in}\ \Lambda[\alpha].\Lambda[\beta].\lambda f : [\alpha \to \beta].\mathsf{withLock}(f, \mathsf{lock}) \\
\\
\mathrm{Syncer}(p) & \triangleq & (s \in \mathrm{Val}).\ \forall f, x, q, r. \quad \{p * q\}\ f\ x\ \{z.\ p * r(z)\} \\
& & \qquad\qquad \Rightarrow\ \{q\}\ s\ \_\ \_\ f\ x\ \{z.\ r(z)\}
\end{array}
$$

Spec for mkSync

$$\forall p \in \mathbb{P}(\mathbf{1}).\ \mathrm{TokPure}(p) \Rightarrow \{p\}\ \ i \Rrightarrow \mathsf{mkSync}\ ()\ \ \{s.\ s \in \mathrm{Syncer}(p)\}$$

Protocol $\pi(p) \triangleq (\theta,\ \ (s \in \mathrm{State}).\ s = \mathsf{Locked} \vee (s = \mathsf{Unlocked} * p))$, where θ is the following STS:



Verification of mkSync

$\lambda().$

Prop context: **Variables:** p
$\mathrm{TokPure}(p)$

$\{p\}$
let lock = **new false**
$\{p * \mathsf{lock} \hookrightarrow_{\mathrm{I}} \mathbf{false}\}$
$\{\exists n.\ \boxed{\mathsf{Unlocked}, \emptyset}^{n}_{\pi(p)}\}$
$\Lambda.\Lambda.\lambda f.\lambda x.$

Prop context: **Variables:** f, x, q, r, n
$\boxed{\mathsf{Unlocked}, \emptyset}^{n}_{\pi(p)}$
$\{p * q\}\ f\ x\ \{z.\ p * r(z)\}$

$\{q\}$
$\{q * \boxed{\mathsf{Unlocked}, \emptyset}^{n}_{\pi(p)}\}$
acq(lock);
$\{q * p * \boxed{\mathsf{Locked}, \bullet}^{n}_{\pi(p)}\}$
let $z = f(x)$
$\{r(z) * p * \boxed{\mathsf{Locked}, \bullet}^{n}_{\pi(p)}\}$
rel(lock);
$\{r(z) * \boxed{\mathsf{Unlocked}, \emptyset}^{n}_{\pi(p)}\}$
$\{r(z)\}$
z
$\{r(z)\}$

7.3 Bag spec for stacks

The principal spec for stacks

$\mathsf{stack_s} \;:\; \forall\alpha.\ (\alpha \to \mathbf{1}) \times (\mathbf{1} \to \alpha_?) \times ((\alpha \to \mathbf{1}) \to \mathbf{1})$
$\mathsf{stack_s} \triangleq \Lambda[\alpha].$
  let $\mathsf{hd_s}$ $= \mathbf{new}\ (\mathsf{none})$
  let sync $= \mathsf{mkSync}()$
  let push $= \lambda x.\ \mathsf{hd_s} := \mathsf{some}(x, \mathsf{get}(\mathsf{hd_s}))$
  let pop $= \lambda().\ \mathbf{case}\ \mathsf{get}(\mathsf{hd_s})\ \mathbf{of}\ \mathsf{none} \Rightarrow \mathsf{none}$
    $\mid\ \mathsf{some}(x, n) \Rightarrow \mathsf{hd_s} := n;\ \mathsf{some}(x)$
  let snap $= \mathsf{sync}\ [\mathbf{1}]\ [\mathsf{list}(\alpha)]\ (\lambda().\mathsf{get}(\mathsf{hd_s}))$
  let iter $= \lambda f.\ \mathsf{foreach}\ [\alpha]\ f\ (\mathsf{snap}())$
  in $(\mathsf{sync}\ [\alpha]\ [\mathbf{1}]\ \ \mathsf{push},\ \mathsf{sync}\ [\mathbf{1}]\ [\alpha_?]\ \ \mathsf{pop},\ \mathsf{iter})$

The abstract bag spec

$\mathrm{Bag} \triangleq (e \in \mathrm{Expr}).$
  $\forall p, q \in \mathbb{P}(\mathrm{Val}).\ (\forall x.\ \mathrm{TokPure}(p(x)) \ \wedge\ (p(x) \Rightarrow \Box q(x))) \Rightarrow$
    $\{\mathrm{True}\}\ e\ \{\mathsf{bag}.\ \mathsf{bag} = (\mathsf{add}, \mathsf{rem}, \mathsf{iter}) * P\}$

  where $P \triangleq \forall x.\ \{p(x)\}\ \mathsf{add}\ \{\mathsf{ret}.\ \mathsf{ret} = ()\}$
    $\wedge\ \{\mathrm{True}\}\ \mathsf{rem}\ \{\mathsf{ret}.\ \mathsf{ret} = \mathsf{none} \vee (\exists x.\ \mathsf{ret} = \mathsf{some}(x) * p(\mathsf{ret}))\}$
    $\wedge\ \forall f, r.\ (\forall x.\ \{q(x) * r\}\ f(x)\ \{\mathsf{ret}.\ \mathsf{ret} = () * r\}) \Rightarrow$
      $\{r\}\ \mathsf{iter}(f)\ \{\mathsf{ret}.\ \mathsf{ret} = () * r\}$

Representation invariant

$$\mathrm{Rep}_p(\mathsf{hd_s}) \triangleq \exists l.\ \mathsf{hd_s} \hookrightarrow_{\mathrm{I}} l * \mathrm{Map}(p)(l)$$

Abstracting stacks to bags We want to show $\mathrm{Bag}(\mathsf{stack_s}\ \_)$.

$\Lambda.$ | **Prop context:** $\forall x.\ \mathrm{TokPure}(p(x)) \ \wedge\ (p(x) \Rightarrow \Box q(x))$ | **Variables:** p, q |

  $\{\mathrm{True}\}$
  let $\mathsf{hd_s} =$ **new** (none)
  $\{s \hookrightarrow_{\mathrm{I}} \mathsf{none}\}$
  $\{\mathrm{Rep}_p(\mathsf{hd_s})\}$
  let sync $= \mathsf{mkSync}()$
  $\{\mathsf{sync} \in \mathrm{Syncer}(\mathrm{Rep}_p(\mathsf{hd_s}))\}$

We split out the characterization of the internal procedures, implicitly inheriting the logical and variable context above.

Prop context: **Variables:** p, q
$\forall x.\ \mathrm{TokPure}(p(x)) \ \wedge\ (p(x) \Rightarrow \Box q(x))$
$\mathsf{sync} \in \mathrm{Syncer}(\mathrm{Rep}_p(\mathsf{hd_s}))$

let push $= \lambda x.$
$\{\mathrm{Rep}_p(\mathsf{hd_s}) * p(x)\}$
$\{\exists l.\ \mathsf{hd_s} \hookrightarrow_{\mathrm{I}} l * \mathrm{Map}(p)(l) * p(x)\}$
$\mathsf{hd_s} := \mathsf{some}(x, \mathsf{get}(s))$
$\{\mathsf{ret}.\ \mathsf{ret} = () * \exists l.\ \mathsf{hd_s} \hookrightarrow_{\mathrm{I}} \mathsf{some}(x, l) * \mathrm{Map}(p)(l) * p(x)\}$
$\{\mathsf{ret}.\ \mathsf{ret} = () * \exists l'.\ \mathsf{hd_s} \hookrightarrow_{\mathrm{I}} l' * \mathrm{Map}(p)(l')\}$
$\{\mathsf{ret}.\ \mathsf{ret} = () * \mathrm{Rep}_p(\mathsf{hd_s})\}$

Prop context: **Variables:** p, q
$\forall x.\ \mathrm{TokPure}(p(x)) \ \wedge\ (p(x) \Rightarrow \Box q(x))$
$\mathsf{sync} \in \mathrm{Syncer}(\mathrm{Rep}_p(\mathsf{hd_s}))$

let pop $= \lambda().$
$\{\mathrm{Rep}_p(\mathsf{hd_s})\}$
$\{\exists l.\ \mathsf{hd_s} \hookrightarrow_{\mathrm{I}} l * \mathrm{Map}(p)(l) * p(x)\}$
case $\mathsf{get}(\mathsf{hd_s})$
of none $\Rightarrow$
$\{\mathsf{hd_s} \hookrightarrow_{\mathrm{I}} \mathsf{none} * \mathrm{Map}(p)(\mathsf{none})\}$
$\{\mathrm{Rep}_p(\mathsf{hd_s})\}$
none
$\{\mathsf{ret}.\ \mathsf{ret} = \mathsf{none} * \mathrm{Rep}_p(\mathsf{hd_s})\}$
| $\mathsf{some}(x, n)$ $\Rightarrow$
$\{\mathsf{hd_s} \hookrightarrow_{\mathrm{I}} \mathsf{some}(x, n) * \mathrm{Map}(p)(\mathsf{some}(x, n))\}$
$\{\mathsf{hd_s} \hookrightarrow_{\mathrm{I}} \mathsf{some}(x, n) * \triangleright \mathrm{Map}(p)(n) * p(x)\}$
$\mathsf{hd_s} := n;$
$\{\mathsf{hd_s} \hookrightarrow_{\mathrm{I}} n * \mathrm{Map}(p)(n) * p(x)\}$
$\{\mathrm{Rep}_p(\mathsf{hd_s}) * p(x)\}$
$\mathsf{some}(x)$
$\{\mathsf{ret}.\ \mathsf{ret} = \mathsf{some}(x) * \mathrm{Rep}_p(\mathsf{hd_s}) * p(x)\}$

Prop context: **Variables:** p, q
$\forall x.\ \mathrm{TokPure}(p(x)) \ \wedge\ (p(x) \Rightarrow \Box q(x))$
$\mathsf{sync} \in \mathrm{Syncer}(\mathrm{Rep}_p(\mathsf{hd_s}))$

let snap = sync _ _ $\lambda()$.

$\{\mathrm{Rep}_p(\mathsf{hd_s})\}$

$\{\exists l.\ \mathsf{hd_s} \hookrightarrow_{\mathrm{I}} l * \mathrm{Map}(p)(l)\}$

$\mathsf{get}(\mathsf{hd_s})$

$\{\mathsf{ret}.\ \mathsf{hd_s} \hookrightarrow_{\mathrm{I}} \mathsf{ret} * \mathrm{Map}(p)(\mathsf{ret})\}$

$\{\mathsf{ret}.\ \mathsf{hd_s} \hookrightarrow_{\mathrm{I}} \mathsf{ret} * \mathrm{Map}(p)(\mathsf{ret}) * \mathrm{Map}(q)(\mathsf{ret})\}$

$\{\mathsf{ret}.\ \mathrm{Rep}_p(\mathsf{hd_s}) * \mathrm{Map}(q)(\mathsf{ret})\}$

Prop context: **Variables:** p, q
$\forall x.\ \mathrm{TokPure}(p(x)) \ \wedge\ (p(x) \Rightarrow \Box q(x))$
$\mathsf{sync} \in \mathrm{Syncer}(\mathrm{Rep}_p(\mathsf{hd_s}))$

let iter = λf.

Prop context: **Variables:** f, r
$\forall x.\ \{q(x) * r\}\ f(x)\ \{\mathsf{ret}.\ \mathsf{ret} = () * r\}$

$\{r\}$

let snapshot = snap()

$\{\mathrm{Map}(q)(\mathsf{snapshot}) * r\}$

foreach _ f snapshot

$\{\mathsf{ret}.\ \mathsf{ret} = () * \mathrm{Map}(\mathrm{True})(\mathsf{snapshot}) * r\}$

$\{\mathsf{ret}.\ \mathsf{ret} = () * r\}$

7.4 Treiber's stack

Protocol We define a protocol $\mathrm{TSProt}(p, \mathsf{hd}_\mathrm{I}, \mathsf{hd}_\mathrm{S}, \mathsf{lock}_\mathrm{S})$ as follows.

We define a simple STS θ, with state space $\mathbb{N} \xrightarrow{\text{fin}} \mathrm{Val}$ and transition relation $s \rightsquigarrow_\theta s'$ iff $s \subseteq s'$. There are no free tokens.

$$
\begin{array}{rcl}
\mathrm{Link} & \triangleq & \mu q.\ (s, x_\mathrm{I}, x_\mathrm{S}).\ \exists x_0, s'.\ s = [x_\mathrm{I} \mapsto x_0] \uplus s' \\
& & \quad (x_\mathrm{S} = \mathsf{none} * x_0 = \mathsf{none}) \\
& * & \quad \vee \left(\begin{array}{l} \exists y_\mathrm{S}, z_\mathrm{S}.\ x_\mathrm{S} = \mathsf{some}(y_\mathrm{S}, z_\mathrm{S}) * \\ \exists y_\mathrm{I}, z_\mathrm{I}.\ x_0 = \mathsf{some}(y_\mathrm{I}, z_\mathrm{I}) * \\ (y_\mathrm{I}, y_\mathrm{S}) \in p * \triangleright q(s', z_\mathrm{I}, z_\mathrm{S}) \end{array} \right) \\
[\![s]\!] & \triangleq & \mathsf{lock}_\mathrm{S} \hookrightarrow_\mathrm{S} \mathbf{false} * \mathop{\Large *}_{\ell \in \mathrm{dom}(s)} \ell \hookrightarrow_\mathrm{I} s(\ell) \\
& * & \exists x_\mathrm{I}, x_\mathrm{S}.\ \mathsf{hd}_\mathrm{I} \hookrightarrow_\mathrm{I} x_\mathrm{I} * \mathsf{hd}_\mathrm{S} \hookrightarrow_\mathrm{S} x_\mathrm{S} \\
& * & \mathrm{Link}(s, x_\mathrm{I}, x_\mathrm{S})
\end{array}
$$

Refinement

$\mathsf{stack}_\mathrm{I} \triangleq \Lambda.$

Prop context:	**Variables:** p, i, j, κ
$\mathrm{Type}(p)$	$i \bowtie j$

$\{j \Rightarrow_\mathrm{S} \kappa[\mathsf{stack}_\mathrm{S}\ \_]\}$
let $\mathsf{hd}_\mathrm{I} =$ **new** nil
$\{j \Rightarrow_\mathrm{S} \kappa[\mathsf{stack}_\mathrm{S}\ \_] * \exists x.\ \mathsf{hd}_\mathrm{I} \hookrightarrow_\mathrm{I} x * x \hookrightarrow_\mathrm{I} \mathsf{none}\}$
$\{\exists\ \mathsf{hd}_\mathrm{S}, \mathsf{lock}_\mathrm{S}.\ \mathsf{hd}_\mathrm{S} \hookrightarrow_\mathrm{S} \mathsf{none} * \mathsf{lock}_\mathrm{S} \hookrightarrow_\mathrm{S} \mathbf{false} * j \Rightarrow_\mathrm{S} \kappa[\mathsf{stack}'_\mathrm{S}] * \exists x.\ \mathsf{hd}_\mathrm{I} \hookrightarrow_\mathrm{I} x * x \hookrightarrow_\mathrm{I} \mathsf{none}\}$
$\{j \Rightarrow_\mathrm{S} \kappa[\mathsf{stack}'_\mathrm{S}] * \exists x.\ \boxed{[x \mapsto \mathsf{none}]}_\pi\}$
$\{j \Rightarrow_\mathrm{S} \kappa[\mathsf{stack}'_\mathrm{S}] * \boxed{\emptyset}_\pi\}$

where

$$
\begin{array}{rcl}
\pi & \triangleq & \mathrm{TSProt}(p, \mathsf{hd}_\mathrm{I}, \mathsf{hd}_\mathrm{S}, \mathsf{lock}_\mathrm{S}) \\
\mathsf{push}_\mathrm{S} & \triangleq & \lambda x.\ \mathsf{hd}_\mathrm{S} := \mathsf{some}(x, \mathsf{get}(\mathsf{hd}_\mathrm{S})) \\
\mathsf{pop}_\mathrm{S} & \triangleq & \lambda().\ \mathbf{case}\ \mathsf{get}(\mathsf{hd}_\mathrm{S})\ \mathbf{of}\ \mathsf{none} \Rightarrow \mathsf{none} \\
& & \qquad\qquad \mid \mathsf{some}(x, n) \Rightarrow \mathsf{hd}_\mathrm{S} := n;\ \mathsf{some}(x) \\
\mathsf{snap} & \triangleq & \mathsf{withLock}(\lambda().\mathsf{get}(\mathsf{hd}_\mathrm{S}), \mathsf{lock}_\mathrm{S}) \\
\mathsf{iter}_\mathrm{S} & \triangleq & \lambda f.\ \mathsf{foreach}\ \_\ f\ (\mathsf{snap}()) \\
\mathsf{stack}'_\mathrm{S} & \triangleq & (\mathsf{withLock}(\mathsf{push}_\mathrm{S}, \mathsf{lock}_\mathrm{S}),\ \mathsf{withLock}(\mathsf{pop}_\mathrm{S}, \mathsf{lock}_\mathrm{S}),\ \mathsf{iter}_\mathrm{S})
\end{array}
$$

let push $=$ **rec** try(x).

Prop context:	**Variables:** $i, j, \kappa, x, x_{\mathrm{S}}$
$i \bowtie j$ $\quad \boxed{\emptyset}_{\pi}$	$(x, x_{\mathrm{S}}) \in p$

$\{j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{push}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ x_{\mathrm{S}}]\}$
let $c =$ **get** hd_{I}
$\{j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{push}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ x_{\mathrm{S}}]\}$
let $n = \mathsf{cons}(x, c)$
$\{j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{push}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ x_{\mathrm{S}}] * n \hookrightarrow_{\mathrm{I}} \mathsf{some}(x, c)\}$
$\{j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{push}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ x_{\mathrm{S}}] * n \hookrightarrow_{\mathrm{I}} \mathsf{some}(x, c) * \boxed{\emptyset}_{\pi}\}$
if CAS$(\mathsf{hd}_{\mathrm{I}}, c, n)$ **then**
  $\{j \Rightarrow_{\mathrm{S}} \kappa[()] * \boxed{[n \hookrightarrow_{\mathrm{I}} \mathsf{some}(x, c)]}_{\pi}\}$
  $()$
  $\{\mathsf{ret}.\ \mathsf{ret} = () * j \Rightarrow_{\mathrm{S}} \kappa[()]\}$
else
  $\{j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{push}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ x_{\mathrm{S}}] * n \hookrightarrow_{\mathrm{I}} \mathsf{some}(x, c) * \boxed{\emptyset}_{\pi}\}$
  $\{j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{push}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ x_{\mathrm{S}}]\}$
  $\mathsf{try}(x)$
  $\{\mathsf{ret}.\ \mathsf{ret} = () * j \Rightarrow_{\mathrm{S}} \kappa[()]\}$

let pop $=$ **rec** try$()$.

Prop context:	**Variables:** i, j, κ
$i \bowtie j$ $\quad \boxed{\emptyset}_{\pi}$	

$\{j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{pop}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ ()]\}$
$\{j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{pop}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ ()] * \boxed{\emptyset}_{\pi}\}$
let $c =$ **get** hd_{I}
$\{\exists p.\ \boxed{p}_{\pi} * \begin{pmatrix} p = [c \mapsto \mathsf{none}] * j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{none}] \\ \vee\ \exists d, n.\ p = [c \mapsto \mathsf{some}(d, n)] * j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{pop}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ ()] \end{pmatrix}\}$
case get c **of**
  $\mathsf{none} \Rightarrow$
    $\{\boxed{[c \mapsto \mathsf{none}]}_{\pi} * j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{none}]\}$
    none
    $\{\mathsf{ret}.\ \exists \mathsf{ret}_{\mathrm{S}}.\ (\mathsf{ret}, \mathsf{ret}_{\mathrm{S}}) \in [\![p_?]\!] * j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{ret}_{\mathrm{S}}]\}$
  $\mid \mathsf{some}(d, n) \Rightarrow$
    $\{\boxed{[c \mapsto \mathsf{some}(d, n)]}_{\pi} * j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{pop}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ ()]\}$
    if CAS$(\mathsf{hd}_{\mathrm{I}}, c, n)$ **then**
      $\{\boxed{[c \mapsto \mathsf{some}(d, n)]}_{\pi} * \exists d_{\mathrm{S}}.\ (d, d_{\mathrm{S}}) \in p * j \Rightarrow_{\mathrm{S}} \kappa[d_{\mathrm{S}}]\}$
      $\mathsf{some}(d)$
      $\{\mathsf{ret}.\ \exists \mathsf{ret}_{\mathrm{S}}.\ (\mathsf{ret}, \mathsf{ret}_{\mathrm{S}}) \in [\![p_?]\!] * j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{ret}_{\mathrm{S}}]\}$
    else
      $\{\boxed{[c \mapsto \mathsf{some}(d, n)]}_{\pi} * j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{pop}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ ()]\}$
      $\{j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(\mathsf{pop}_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ ()]\}$
      $\mathsf{try}()$
      $\{\mathsf{ret}.\ \exists \mathsf{ret}_{\mathrm{S}}.\ (\mathsf{ret}, \mathsf{ret}_{\mathrm{S}}) \in [\![p_?]\!] * j \Rightarrow_{\mathrm{S}} \kappa[\mathsf{ret}_{\mathrm{S}}]\}$

let iter $= \lambda f.$

Prop context:	**Variables:** $i, j, \kappa, f, f_{\mathrm{s}}$
$i \bowtie j$	$\boxed{\emptyset}_\pi$ $\quad (f, f_{\mathrm{s}}) \in [\![p \to \mathbf{1}]\!]$

$\{\mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{iter}_{\mathrm{s}}\ f_{\mathrm{s}}]\}$

let rec loop(c) $=$

Prop context:	**Variables:** c, c_{s}

$\{\mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{foreach'}\ c_{\mathrm{s}}] * \exists s.\ \boxed{s}_\pi * \mathrm{Link}(s, c, c_{\mathrm{s}})\}$

case get c **of**

none $\Rightarrow$

$\{\mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{foreach'}\ c_{\mathrm{s}}] * \exists s'.\ \boxed{[c \mapsto \mathsf{none}] \uplus s'}_\pi * \mathrm{Link}(s, c, c_{\mathrm{s}})\}$

$\{\mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{foreach'}\ c_{\mathrm{s}}] * c_{\mathrm{s}} = \mathsf{none}\}$

$\{\mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[()]\}$

$()$

$\{\mathsf{ret}.\ \mathsf{ret} = () * \mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[()]\}$

| some(d, n) $\Rightarrow$

$\{\mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{foreach'}\ c_{\mathrm{s}}] * \exists s'.\ \boxed{[c \mapsto \mathsf{some}(d, n)] \uplus s'}_\pi * \mathrm{Link}(s, c, c_{\mathrm{s}})\}$

$\{\mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{foreach'}\ c_{\mathrm{s}}] * \exists s'.\ \boxed{s'}_\pi * \exists d_{\mathrm{s}}, n_{\mathrm{s}}.\ c_{\mathrm{s}} = \mathsf{some}(d_{\mathrm{s}}, n_{\mathrm{s}}) * (d, d_{\mathrm{s}}) \in p * \triangleright \mathrm{Link}(s', n, n_{\mathrm{s}})\}$

$\{\exists d_{\mathrm{s}}, n_{\mathrm{s}}.\ \mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[f_{\mathrm{s}}(d_{\mathrm{s}}); \mathsf{foreach'}\ n_{\mathrm{s}}] * \exists s'.\ \boxed{s'}_\pi * (d, d_{\mathrm{s}}) \in p * \triangleright \mathrm{Link}(s', n, n_{\mathrm{s}})\}$

$f(d);$

$\{\exists n_{\mathrm{s}}.\ \mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{foreach'}\ n_{\mathrm{s}}] * \exists s'.\ \boxed{s'}_\pi * \triangleright \mathrm{Link}(s', n, n_{\mathrm{s}})\}$

loop(n)

$\{\mathsf{ret}.\ \mathsf{ret} = () * \mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[()]\}$

let $h =$ **get** hd_{I}

$\{\exists h_{\mathrm{s}}.\ \mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{foreach'}\ h_{\mathrm{s}}] * \exists s.\ \boxed{s}_\pi * \mathrm{Link}(s, h, h_{\mathrm{s}})\}$

loop h

$\{\mathsf{ret}.\ \mathsf{ret} = () * \mathit{Tid}(i) * j \Rrightarrow_{\mathrm{s}} \kappa[()]\}$

7.5 Universal construction

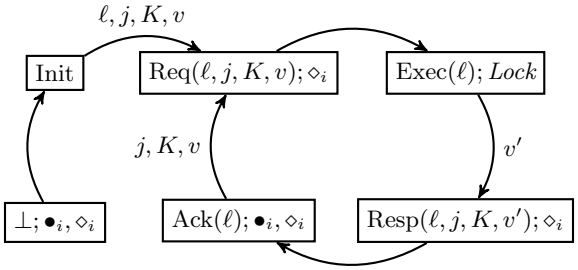
$\mathsf{flat}_\mathrm{S} \;:\; \forall\alpha.\, \forall\beta.\, (\alpha \to \beta) \to (\alpha \to \beta)$
$\mathsf{flat}_\mathrm{S} \triangleq \mathsf{mkSync}()$
$(e_\mathrm{I}, e_\mathrm{S}) \downarrow^{\mathrm{pure}} \varphi \triangleq \forall j, \kappa.\, \{j \Rightarrow_\mathrm{S} \kappa[e_\mathrm{S}]\}\; e_\mathrm{I}\; \{\mathsf{ret}_\mathrm{I}.\, \exists \mathsf{ret}_\mathrm{S}.\, \varphi(\mathsf{ret}_\mathrm{I}, \mathsf{ret}_\mathrm{S}) * j \Rightarrow_\mathrm{S} \kappa[\mathsf{ret}_\mathrm{S}]\}$

Specification: a qualified refinement

$\forall \alpha, \beta, f_\mathrm{I}, f_\mathrm{S}.\; \mathrm{Type}(\alpha) \wedge \mathrm{Type}(\beta) \wedge \square(\forall (x_\mathrm{I}, x_\mathrm{S}) \in \alpha.\, (f_\mathrm{I}\; x_\mathrm{I}, f_\mathrm{S}\; x_\mathrm{S}) \downarrow^{\mathrm{pure}} \beta)$
$\Rightarrow \;\; \cdot \vdash \mathsf{flat}_\mathrm{I}\; \_\; \_\; f_\mathrm{I} \preceq \mathsf{flat}_\mathrm{S}\; \_\; \_\; f_\mathrm{S} : \alpha \to \beta$

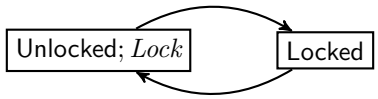
Protocol We define a protocol $\mathrm{FCProt}(f_\mathrm{S}, \mathsf{lock}_\mathrm{S}, \mathsf{lock}, \mathsf{localOps})$ as follows.
First, we have a following per-thread STS θ_i, with state space

$$\mathrm{OpState} ::= \bot \mid \mathrm{Init} \mid \mathrm{Req}(\ell, j, K, v) \mid \mathrm{Exec}(\ell) \mid \mathrm{Resp}(\ell, j, K, v') \mid \mathrm{Ack}(\ell)$$



Second, we have a single lock STS θ_{lock} with state space

$$\mathrm{LockState} ::= \mathsf{Unlocked} \mid \mathsf{Locked}$$



The full STS θ is then a product:

$$\begin{array}{ll}
\text{State space} & \left\{ (S, s) \;\middle|\; \begin{array}{l} S \in \mathbb{N} \xrightarrow{\mathrm{fin}} \mathrm{OpState},\; s \in \mathrm{LockState}, \\ (\forall i.\; \theta_i.\mathcal{T}(S(i)) \# \theta_{\mathsf{lock}}.\mathcal{T}(s)), \\ (\forall i, j.\; \theta_i.\mathcal{T}(S(i)) \# \theta_j.\mathcal{T}(S(j))) \end{array} \right\} \\
\text{Transitions} & (S, s) \rightsquigarrow_\theta (S', s') \;\triangleq\; s \rightsquigarrow^?_{\theta_{\mathsf{lock}}} s' \wedge \forall i.\; S(i) \rightsquigarrow^?_{\theta_i} S'(i) \\
\text{Free tokens} & \theta.\mathcal{T}(S, s) \;\triangleq\; \theta_{\mathsf{lock}}.\mathcal{T}(s) \cup \bigcup_i \theta_i.\mathcal{T}(S(i))
\end{array}$$

Note that, in giving the state space, we pun the $\bot$ state with an "undefined" input to a partial function—and thus, we require that only a finite number of threads i have a non-$\bot$ state in S.

Next we define an interpretation of states for the internal transition systems:

$$
\begin{array}{rcl}
[\![\mathrm{Init}]\!]_i^L & \triangleq & \mathit{Tid}(i) \\
[\![\mathrm{Req}(\ell, j, \kappa, x_{\mathrm{I}})]\!]_i^L & \triangleq & L(i) = \ell * \ell \hookrightarrow_{\mathrm{I}} \mathbf{inj}_1\ x_{\mathrm{I}} * \mathit{Tid}(i) * \exists x_{\mathrm{S}}.\ (x_{\mathrm{I}}, x_{\mathrm{S}}) \in \alpha * j \Rrightarrow_{\mathrm{S}} \kappa[\mathsf{withLock}(f_{\mathrm{S}}, \mathsf{lock}_{\mathrm{S}})\ x_{\mathrm{S}}] \\
[\![\mathrm{Exec}(\ell)]\!]_i^L & \triangleq & L(i) = \ell * \ell \hookrightarrow_{\mathrm{I}} \mathbf{inj}_1\ - * \mathit{Tid}(i) \\
[\![\mathrm{Resp}(\ell, j, \kappa, y_{\mathrm{I}})]\!]_i^L & \triangleq & L(i) = \ell * \ell \hookrightarrow_{\mathrm{I}} \mathbf{inj}_2\ y_{\mathrm{I}} * \mathit{Tid}(i) * \exists y_{\mathrm{S}}.\ (y_{\mathrm{I}}, y_{\mathrm{S}}) \in \beta * j \Rrightarrow_{\mathrm{S}} \kappa[y_{\mathrm{S}}] \\
[\![\mathrm{Ack}(\ell)]\!]_i^L & \triangleq & L(i) = \ell * \ell \hookrightarrow_{\mathrm{I}} \mathbf{inj}_2\ - \\
[\![\mathsf{Unlocked}]\!] & \triangleq & \mathsf{lock} \hookrightarrow_{\mathrm{I}} \mathbf{false} * \mathsf{lock}_{\mathrm{S}} \hookrightarrow_{\mathrm{S}} \mathbf{false} \\
[\![\mathsf{Locked}]\!] & \triangleq & \mathsf{lock} \hookrightarrow_{\mathrm{I}} \mathbf{true}
\end{array}
$$

Finally, we lift these internal interpretations to interpret global states:

$$(S, s).\ [\![s]\!] * \exists L.\ \mathsf{lclOps} \hookrightarrow_{\mathrm{I}} L * \mathop{\Large *}_{i \in \mathrm{dom}(S)} [\![S(i)]\!]_i^L$$

Flat combining implementation

$\mathsf{flat}_\mathrm{I} \triangleq \Lambda.\ \Lambda.\ \lambda f_\mathrm{I}.$

Prop context:	**Variables:** $\alpha, \beta, f_\mathrm{I}, f_\mathrm{S}, i, j, \kappa$
$\mathrm{Type}(\alpha)$ $\quad \mathrm{Type}(\beta)$	$i \bowtie j$
$\forall(x_\mathrm{I}, x_\mathrm{S}) \in \alpha.\ (f_\mathrm{I}\ x_\mathrm{I}, f_\mathrm{S}\ x_\mathrm{S}) \downarrow^{\mathrm{pure}} \beta$	

$\{\mathit{Tid}(i) * j \Rrightarrow_\mathrm{S} \kappa[\mathsf{flat}_\mathrm{S}\ f_\mathrm{S}]\}$
let lock = **new** (**false**)
$\{\mathit{Tid}(i) * j \Rrightarrow_\mathrm{S} \kappa[\mathsf{flat}_\mathrm{S}\ f_\mathrm{S}] * \mathsf{lock} \hookrightarrow_\mathrm{I} \mathbf{false}\}$
let localOps = **newLcl**
$\{\mathit{Tid}(i) * j \Rrightarrow_\mathrm{S} \kappa[\mathsf{flat}_\mathrm{S}\ f_\mathrm{S}] * \mathsf{lock} \hookrightarrow_\mathrm{I} \mathbf{false} * \mathsf{localOps} \hookrightarrow_\mathrm{I} \emptyset\}$
$\{\mathit{Tid}(i) * \exists\ \mathsf{lock}_\mathrm{S}.\ j \Rrightarrow_\mathrm{S} \kappa[\mathsf{withLock}(f_\mathrm{S}, \mathsf{lock}_\mathrm{S})] * \mathsf{lock}_\mathrm{S} \hookrightarrow_\mathrm{S} \mathbf{false} * \mathsf{lock} \hookrightarrow_\mathrm{I} \mathbf{false} * \mathsf{localOps} \hookrightarrow_\mathrm{I} \emptyset\}$
$\{\mathit{Tid}(i) * \exists\ \mathsf{lock}_\mathrm{S}.\ j \Rrightarrow_\mathrm{S} \kappa[\mathsf{withLock}(f_\mathrm{S}, \mathsf{lock}_\mathrm{S})] * \exists n.\ \boxed{\emptyset}^{n}_{\pi}\}$
let (add, _, iter) = $\mathsf{stack}_\mathrm{S}$ _

where $\pi = \mathrm{FCProt}(f_\mathrm{S}, \mathsf{lock}_\mathrm{S}, \mathsf{lock}, \mathsf{localOps})$.

We instantiate the stack-as-bag spec using the *same* predicate for element resources and for iteration resources:

$$\mathrm{Op}(\ell) \quad \triangleq \quad \exists k.\ \boxed{k \mapsto \mathrm{Req}(\ell, -, -, -)}^{n}_{\pi}$$

$\mathsf{let}\ \mathsf{doOp} = \lambda o.$

$\boxed{\textbf{Variables: } o, i}$

$\{\mathrm{Op}(o) * \boxed{\emptyset; \mathit{Lock}}^{n}_{\pi} * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{false}\}$

$\{\exists s.\ \boxed{s; \mathit{Lock}}^{n}_{\pi} * (s = k \mapsto \mathrm{Req}(o, -, -, -) \vee s = k \mapsto \mathrm{Resp}(o, -, -, -)) * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{false}\}$

$\mathsf{case}\ \mathsf{get}(o)$

(1) $\mathsf{of}\ \mathsf{inj}_1\ \mathsf{req} \Rightarrow$

$\{\boxed{k \mapsto \mathrm{Exec}(o); \diamond_i}^{n}_{\pi} * \exists j, \mathsf{req}_{\mathrm{s}}.\ (\mathsf{req}, \mathsf{req}_{\mathrm{s}}) \in \alpha * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{withLock}(f_{\mathrm{s}}, \mathsf{lock}_{\mathrm{s}})\ \mathsf{req}_{\mathrm{s}}] * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{false}\}$

$\{\boxed{k \mapsto \mathrm{Exec}(o); \diamond_i}^{n}_{\pi} * \exists j, \mathsf{req}_{\mathrm{s}}.\ (\mathsf{req}, \mathsf{req}_{\mathrm{s}}) \in \alpha * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{let}\ r = f_{\mathrm{s}}\ \mathsf{req}_{\mathrm{s}}\ \mathsf{in}\ \mathsf{rel}(\mathsf{lock}_{\mathrm{s}}); r] * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{true}\}$

$\{\boxed{k \mapsto \mathrm{Exec}(o); \diamond_i}^{n}_{\pi} * \exists j, \mathsf{req}_{\mathrm{s}}.\ (\mathsf{req}, \mathsf{req}_{\mathrm{s}}) \in \alpha * j \Rrightarrow_{\mathrm{s}} (\kappa[\mathsf{let}\ r = []\ \mathsf{in}\ \mathsf{rel}(\mathsf{lock}_{\mathrm{s}}); r])[f_{\mathrm{s}}\ \mathsf{req}_{\mathrm{s}}] * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{true}\}$

$\mathsf{let}\ \mathsf{res} = f_1(\mathsf{req})$

$\{\boxed{k \mapsto \mathrm{Exec}(o); \diamond_i}^{n}_{\pi} * \exists j, \mathsf{res}_{\mathrm{s}}.\ (\mathsf{res}, \mathsf{res}_{\mathrm{s}}) \in \beta * j \Rrightarrow_{\mathrm{s}} (\kappa[\mathsf{let}\ r = []\ \mathsf{in}\ \mathsf{rel}(\mathsf{lock}_{\mathrm{s}}); r])[\mathsf{res}_{\mathrm{s}}] * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{true}\}$

$\{\boxed{k \mapsto \mathrm{Exec}(o); \diamond_i}^{n}_{\pi} * \exists j, \mathsf{res}_{\mathrm{s}}.\ (\mathsf{res}, \mathsf{res}_{\mathrm{s}}) \in \beta * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{let}\ r = \mathsf{res}_{\mathrm{s}}\ \mathsf{in}\ \mathsf{rel}(\mathsf{lock}_{\mathrm{s}}); r] * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{true}\}$

$\{\boxed{k \mapsto \mathrm{Exec}(o); \diamond_i}^{n}_{\pi} * \exists j, \mathsf{res}_{\mathrm{s}}.\ (\mathsf{res}, \mathsf{res}_{\mathrm{s}}) \in \beta * j \Rrightarrow_{\mathrm{s}} \kappa[\mathsf{res}_{\mathrm{s}}] * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{false}\}$

(2) $o := \mathsf{inj}_2\ \mathsf{res}$

$\{\mathsf{ret}.\ \mathsf{ret} = () * \boxed{k \mapsto \mathrm{Resp}(o, -, -, \mathsf{res}); \mathit{Lock}}^{n}_{\pi} * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{false}\}$

$\{\mathsf{ret}.\ \mathsf{ret} = () * \boxed{\emptyset; \mathit{Lock}}^{n}_{\pi} * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{false}\}$

(3) $\mid \mathsf{inj}_2\ \mathsf{res} \Rightarrow$

$\{\boxed{k \mapsto \mathrm{Resp}(o, -, -, \mathsf{res}); \mathit{Lock}}^{n}_{\pi} * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{false}\}$

$()$

$\{\mathsf{ret}.\ \mathsf{ret} = () * \boxed{\emptyset; \mathit{Lock}}^{n}_{\pi} * \mathsf{lock}_{\mathrm{s}} \hookrightarrow_{\mathrm{s}} \mathsf{false}\}$

1. Suppose $o \hookrightarrow_{\mathrm{I}} \mathsf{inj}_1\ -$ instantaneously, and the state is rely-bounded by Req; *Lock* or Resp; *Lock*. The real state cannot be Exec because *Lock* is locally-owned; it cannot be Resp or Ack because o is currently $\mathsf{inj}_1\ -$. Thus, the state *must* be Req. We make a guarantee move to Exec; $\diamond_i$ (trading *Lock* for this token).

2. Given that our rely bound is Exec; $\diamond_i$, the actual state *must* still be Exec, because the token $\diamond_i$ must be given up to move forward, and we own the token locally. We, however, can make a guarantee move to Resp; *Lock*, trading $\diamond_i$ for *Lock*, by both updating o to $\mathsf{inj}_2\ -$ and handing over the associated spec resources.

3. Suppose $o \hookrightarrow_{\mathrm{I}} \mathsf{inj}_2\ -$ instantaneously, and the state is rely-bounded by Req; *Lock* or Resp; *Lock*. The real state cannot be Req or Exec, which require o to be $\mathsf{inj}_1\ -$. So it must be either Resp or Ack, either of which is in the rely-future of Resp; *Lock*.

let install $= \lambda$ req.

Prop context:	**Variables:** $\mathsf{req}, \mathsf{req_s}, i, j, \kappa$
$(\mathsf{req}, \mathsf{req_s}) \in \alpha$	
$\boxed{\emptyset}^n_\pi$	

$\{Tid(i) * j \Rrightarrow_s \kappa[\mathsf{withLock}(\mathsf{lock_s}, f_s)\ \mathsf{req_s}]\}$

$\{\boxed{\emptyset}^n_\pi * Tid(i) * j \Rrightarrow_s \kappa[\mathsf{withLock}(\mathsf{lock_s}, f_s)\ \mathsf{req_s}]\}$

case getLcl(localOps)

(1) **of** none $\Rightarrow$

$\{\boxed{i \mapsto \mathrm{Init}; \bullet_i, \diamond_i}^n_\pi * j \Rrightarrow_s \kappa[\mathsf{withLock}(\mathsf{lock_s}, f_s)\ \mathsf{req_s}]\}$

let o = **new** ($\mathbf{inj}_1$ req)

$\{\boxed{i \mapsto \mathrm{Init}; \bullet_i, \diamond_i}^n_\pi * j \Rrightarrow_s \kappa[\mathsf{withLock}(\mathsf{lock_s}, f_s)\ \mathsf{req_s}] * o \hookrightarrow_1 \mathbf{inj}_1\ \mathsf{req}\}$

(2) **setLcl**(localOps, o);

$\{\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, \mathsf{req}); \bullet_i}^n_\pi\}$

$\{\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, \mathsf{req}); \bullet_i}^n_\pi * \mathrm{Op}(o)\}$

add(o);

$\{\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, \mathsf{req}); \bullet_i}^n_\pi\}$

o

$\{\mathsf{ret}.\ \boxed{i \mapsto \mathrm{Req}(\mathsf{ret}, j, \kappa, \mathsf{req}); \bullet_i}^n_\pi\}$

(3) | some(o) $\Rightarrow$

$\{\boxed{i \mapsto \mathrm{Ack}(o)}^n_\pi * Tid(i) * j \Rrightarrow_s \kappa[\mathsf{withLock}(\mathsf{lock_s}, f_s)\ \mathsf{req_s}]\}$

(4) $o :=$ **inj**$_1$ req;

$\{\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, \mathsf{req}); \bullet_i}^n_\pi\}$

o

$\{\mathsf{ret}.\ \boxed{i \mapsto \mathrm{Req}(\mathsf{ret}, j, \kappa, \mathsf{req}); \bullet_i}^n_\pi\}$

1. Given that $\lfloor \mathsf{localOps} \rfloor(i) = \mathsf{none}$ instantaneously and that we own $Tid(i)$, the only possible state for the thread i protocol is $\bot$. We make a guarantee move to Init, gaining both tokens, by giving up ownership of $Tid(i)$.

2. Given our assertion $\boxed{i \mapsto \mathrm{Init}; \bullet_i, \diamond_i}^n_\pi$, the protocol for i *must* be in the Init state (we own the token necessary to move forward). On the other hand, by initializing the localOps at index i, we must make a guarantee move to the request state, giving up the $\diamond_i$ token, the spec code, and ownership of o.

3. Given that $\lfloor \mathsf{localOps} \rfloor(i) = \mathsf{some}(o)$ instantaneously and that we own $Tid(i)$, the only possible state for the thread i protocol is Ack.

4. Given that the state is bounded by Ack and that we own $Tid(i)$, the state must instantaneously be some (future) Ack, *i.e.,* the protocol may have gone around the cycle some number of times but must have stopped at Ack. In writing $\mathbf{inj}_1\ -$ to o we must make a guarantee move to Req, choosing a new j, K, v for the protocol and giving up ownership of $Tid(i)$ and our spec code (and copying the knowledge, from our proof context, about the relatedness of arguments to f_s). In return we gain the token $\bullet_i$.

in $\lambda x.$

Prop context:	**Variables:** $i, j, \kappa, x, x_\mathrm{s}$
$i \bowtie j$	
$(x, x_\mathrm{s}) \in \alpha$	

$\{\mathit{Tid}(i) * j \Rightarrow_\mathrm{s} \kappa[\mathsf{withLock}(f_\mathrm{s}, \mathsf{lock}_\mathrm{s})\ x_\mathrm{s}]\}$
let $o = \mathsf{install}(x)$
$\{\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, x); \bullet_i}^{n}_{\pi}\}$
let rec loop() =
  $\{\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, x); \bullet_i}^{n}_{\pi}\}$
  case get(o)
    of $\mathbf{inj}_1$ _ $\Rightarrow$
      $\{\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, x); \bullet_i}^{n}_{\pi}\}$
      if not(**get** lock) **and** tryAcq(lock) **then**
(1) $\{\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, x); \bullet_i}^{n}_{\pi} * \boxed{\emptyset; \mathit{Lock}}^{n}_{\pi} * \mathsf{lock}_\mathrm{s} \hookrightarrow_\mathrm{s} \mathbf{false}\}$
(2)   iter doOp;
        $\{\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, x); \bullet_i}^{n}_{\pi} * \boxed{\emptyset; \mathit{Lock}}^{n}_{\pi} * \mathsf{lock}_\mathrm{s} \hookrightarrow_\mathrm{s} \mathbf{false}\}$
        rel(lock);
        $\{\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, x); \bullet_i}^{n}_{\pi}\}$
        loop()
        $\{\mathsf{res}.\ \boxed{i \mapsto \mathrm{Ack}(o)}^{n}_{\pi} * \mathit{Tid}(i) * \exists \mathsf{res}_\mathrm{s}.\ (\mathsf{res}, \mathsf{res}_\mathrm{s}) \in \beta * j \Rightarrow_\mathrm{s} \kappa[\mathsf{res}_\mathrm{s}]\}$
      else
        $\{\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, x); \bullet_i}^{n}_{\pi}\}$
        loop()
        $\{\mathsf{res}.\ \boxed{i \mapsto \mathrm{Ack}(o)}^{n}_{\pi} * \mathit{Tid}(i) * \exists \mathsf{res}_\mathrm{s}.\ (\mathsf{res}, \mathsf{res}_\mathrm{s}) \in \beta * j \Rightarrow_\mathrm{s} \kappa[\mathsf{res}_\mathrm{s}]\}$
(3) | $\mathbf{inj}_2$ res $\Rightarrow$
        $\{\boxed{i \mapsto \mathrm{Ack}(o)}^{n}_{\pi} * \mathit{Tid}(i) * \exists \mathsf{res}_\mathrm{s}.\ (\mathsf{res}, \mathsf{res}_\mathrm{s}) \in \beta * j \Rightarrow_\mathrm{s} \kappa[\mathsf{res}_\mathrm{s}]\}$
        res
  in loop()
  $\{\mathsf{res}.\ \boxed{i \mapsto \mathrm{Ack}(o)}^{n}_{\pi} * \mathit{Tid}(i) * \exists \mathsf{res}_\mathrm{s}.\ (\mathsf{res}, \mathsf{res}_\mathrm{s}) \in \beta * j \Rightarrow_\mathrm{s} \kappa[\mathsf{res}_\mathrm{s}]\}$

1. The tryAcq function will succeed only if we can atomically update lock from **false** to **true**. In so doing, we move the locking protocol from the Unlocked to the Locked state, gaining control of the *Lock* token and ownership of lock_s.

2. Here we apply the iter spec with loop invariant $\boxed{\emptyset; \mathit{Lock}}^{n}_{\pi} * \mathsf{lock}_\mathrm{s} \hookrightarrow_\mathrm{s} \mathbf{false}$. The assertion $\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, x); \bullet_i}^{n}_{\pi}$ acts as a frame.

3. Given our assertion $\boxed{i \mapsto \mathrm{Req}(o, j, \kappa, x); \bullet_i}^{n}_{\pi}$ and that o is instantaneously $\mathbf{inj}_2\ -$, the only possible state is Resp (the state cannot be Req or Exec, which require $o \hookrightarrow_\mathrm{I} \mathbf{inj}_1\ -$, nor Ack, which requires $\bullet_i$). We make a guarantee move to Ack, giving up the token $\bullet_i$ in return for $\mathit{Tid}(i)$ and our spec code.

8 Soundness

8.1 Basic Properties

Lemma 1 (Rely-guarantee Preorders)**.** The relations $\overset{\text{rely}}{\sqsubseteq}$ and $\overset{\text{guar}}{\sqsubseteq}$ are preorders.

Lemma 2 (Rely-closure of Assertions)**.** $[\![P]\!]^{\rho}_{W,\eta}$ and $W' \overset{\text{rely}}{\sqsupseteq} W$ implies $[\![P]\!]^{\rho}_{W',\eta}$.

Lemma 3. $|W| \otimes W = W$.

Lemma 4. If $W \overset{\text{rely}}{\sqsubseteq} W'$ then $|W| \overset{\text{rely}}{\sqsubseteq} |W'|$.

Lemma 5 (Rely Decomposition)**.** If $W_1 \otimes W_2 \overset{\text{rely}}{\sqsubseteq} W'$ then there are W_1' and W_2' with $W' = W_1' \otimes W_2'$, $W_1 \overset{\text{rely}}{\sqsubseteq} W_1'$ and $W_2 \overset{\text{rely}}{\sqsubseteq} W_2'$.

Lemma 6 (Token Framing)**.** If $W \overset{\text{guar}}{\sqsubseteq} W'$ and $W \otimes W_f$ is defined then there exists some $W_f' \overset{\text{rely}}{\sqsupseteq} W_f$ such that $W' \otimes W_f'$ is defined and $W \otimes W_f \overset{\text{guar}}{\sqsubseteq} W' \otimes W_f'$.

Lemma 7. If $(\varsigma_{\text{I}}, \varsigma_{\text{S}}) : W, \eta$ then $(\varsigma_{\text{I}}, \varsigma_{\text{S}}) : W \otimes W', \eta$.

Lemma 8. If $(\varsigma_{\text{I}}, \varsigma_{\text{S}}) : W \otimes W', \eta$ then $(\varsigma_{\text{I}}, \varsigma_{\text{S}}) : W, \eta$.

Lemma 9. If $W.k > 0$ then $\triangleright W \overset{\text{guar}}{\sqsupseteq} W$ and $\triangleright W \overset{\text{rely}}{\sqsupseteq} W$.

Lemma 10. If $W.k > 0$ then $|\triangleright W| = \triangleright |W|$.

Lemma 11 (Later Satisfaction)**.** If $W.k > 0$ and $(\varsigma_{\text{I}}, \varsigma_{\text{S}}) : W, \eta$ then $(\varsigma_{\text{I}}, \varsigma_{\text{S}}) : \triangleright W, \eta$.

8.2 Properties of safe

Lemma 12 (Framing)**.** $[\![\mathsf{safe}(M, N, x.\ Q)]\!]^{\rho}_{W,\eta}$ and $W_f, \eta_f \models^{\rho} R$ with $W \# W_f$, $\eta \# \eta_f$ gives

$$[\![\mathsf{safe}(M, N, x.\ Q * R)]\!]^{\rho}_{W \otimes W_f, \eta \otimes \eta_f}.$$

Proof. The proof proceeds by induction on $W.k$.

Case $\boxed{W.k = 0}$

1. Let $W' \overset{\text{rely}}{\sqsupseteq} W \otimes W_f$.
2. $W'.k = (W \otimes W_f).k = W.k = 0$.

Case $\boxed{W.k > 0}$

3. Let $\mathcal{E} = [\![M]\!]\rho$.
4. Let $W' \overset{\text{rely}}{\sqsupseteq} W \otimes W_f, \eta'_f \#, \eta \otimes \eta_f, \mathcal{E}_f \# \mathcal{E}$.
5. Write $W' = W'_1 \otimes W'_2$ with $W'_1 \overset{\text{rely}}{\sqsupseteq} W, W'_2 \overset{\text{rely}}{\sqsupseteq} W_f$ by Lem. 5.
6. Suppose $(W'_1 \otimes W'_2).k > 0$.
7. $W'_1.k = (W'_1 \otimes W'_2).k > 0$.
8. Suppose $h; \mathcal{E} \uplus \mathcal{E}_F, \varsigma_{\mathrm{s}} : W'_1 \otimes W'_2, \eta \otimes \eta_f \otimes \eta'_f$.
9. $h; \mathcal{E} \uplus \mathcal{E}_F, \varsigma_{\mathrm{s}} : W'_1, \eta \otimes \eta_f \otimes \eta'_f$ by Lem. 8.
10. Suppose $\mathcal{E}' \# \mathcal{E}_f$.

Case $\boxed{h; \mathcal{E} \to h'; \mathcal{E}'}$

11. Pick $\varsigma'_{\mathrm{s}}, \eta'$, and W''_1 with by assumption.
    $W''_1 \overset{\text{guar}}{\sqsupseteq_1} W'_1$,
    $\varsigma_{\mathrm{s}} \to^* \varsigma'_{\mathrm{s}}$,
    $h'; \mathcal{E}' \uplus \mathcal{E}_f, \varsigma'_{\mathrm{s}} : W''_1, \eta' \otimes \eta_f \otimes \eta'_f$,
    $W''_1.k = W'_1.k - 1$,
    $[\![\mathsf{safe}(\mathcal{E}', N, x.\ Q)]\!]^{\rho}_{W''_1, \eta'}$
12. Pick $W''_2 \overset{\text{rely}}{\sqsupseteq} W'_2$ with $W''_1 \otimes W''_2 \overset{\text{guar}}{\sqsupseteq} W'_1 \otimes W'_2$ by Lem. 6.
13. $h'; \mathcal{E}' \uplus \mathcal{E}_f, \varsigma'_{\mathrm{s}} : W''_1 \otimes W''_2, \eta' \otimes \eta_f \otimes \eta'_f$ by Lem. 7.
14. $(W''_1 \otimes W''_2).k = W''_1.k = W'_1.k - 1 = (W'_1 \otimes W'_2).k - 1$.
15. $W''_2, \eta_f \models^{\rho} R$.
16. $[\![\mathsf{safe}(\mathcal{E}', N, x.\ Q * R)]\!]^{\rho}_{W''_1 \otimes W''_2, \eta' \otimes \eta_f}$ by induction hypothesis.

Case $\boxed{[\![N]\!]^{\rho} = i \text{ and } \mathcal{E} = \mathcal{E}_0 \uplus [i \mapsto v]}$

17. Pick $\varsigma'_{\mathrm{s}}, \eta'$, and W''_1 with by assumption.
    $W''_1 \overset{\text{guar}}{\sqsupseteq_0} W'_1$,
    $\varsigma_{\mathrm{s}} \to^* \varsigma'_{\mathrm{s}}$,
    $h; \mathcal{E} \uplus \mathcal{E}_f, \varsigma'_{\mathrm{s}} : W''_1, \eta' \otimes \eta_f \otimes \eta'_f$,
    $[\![\mathsf{safe}(\mathcal{E}_0, N, \mathrm{True}) * Q(v)]\!]^{\rho}_{W''_1, \eta'}$

18. Pick $W_2'' \overset{\text{rely}}{\sqsupseteq} W_2'$ with $W_1'' \otimes W_2'' \overset{\text{guar}}{\sqsupseteq} W_1' \otimes W_2'$ by Lem. 6.
19. $h'; \mathcal{E} \uplus \mathcal{E}_f, \varsigma_s' : W_1'' \otimes W_2'', \eta' \otimes \eta_f \otimes \eta_f'$ by Lem. 7.
20. $(W_1'' \otimes W_2'').k = W_1''.k = W_1'.k = (W_1' \otimes W_2').k$.
21. $W_2'', \eta_f \models^\rho R$.
22. $[\![\mathsf{safe}(\mathcal{E}_0, N, \mathrm{True}) * Q(v) * R]\!]^\rho_{W_1'' \otimes W_2'', \eta' \otimes \eta_f}$.

□

Lemma 13 (Parallel Composition)**.** If we have $[\![\mathsf{safe}(M_1, N_1, x.\ Q_1)]\!]^{\rho}_{W_1,\eta_1}$ and $[\![\mathsf{safe}(M_2, N_2, x.\ Q_2)]\!]^{\rho}_{W_2,\eta_2}$ with $W_1 \# W_2$, $\eta_1 \# \eta_2$, $[\![M_1]\!]\rho \# [\![M_2]\!]\rho$ and $[\![N_1]\!] \in \mathrm{dom}([\![M_1]\!]\rho)$ then we also have

$$[\![\mathsf{safe}(M_1 \uplus M_2, N_1, x.\ Q_1)]\!]^{\rho}_{W_1 \otimes W_2, \eta_1 \otimes \eta_2}.$$

Proof. Let $\mathcal{E}_1 = [\![M_1]\!]\rho$, $\mathcal{E}_2 = [\![M_2]\!]\rho$, $n_1 = [\![N_1]\!]\rho$, $n_2 = [\![N_2]\!]\rho$. The proof proceeds by induction on the measure $M(\mathcal{E}_1, W_1, n_1)$ defined by

$$M(\mathcal{E}, W, n) = \begin{cases} W.k & n \notin \mathrm{dom}(\mathcal{E}) \\ W.k + 1 & n \in \mathrm{dom}(\mathcal{E}) \end{cases}$$

Case $\boxed{M(\mathcal{E}_1, W_1, n_1) = 0}$

1. $(W_1 \otimes W_2).k = W_1.k = 0$.

Case $\boxed{M(\mathcal{E}_1, W_1, n_1) > 0}$

2. Let $W' \overset{\text{rely}}{\sqsupseteq} W_1 \otimes W_2, \eta_f \# \eta_1 \otimes \eta_2, \mathcal{E}_f \# \mathcal{E}_1 \uplus \mathcal{E}_2$.
3. Write $W' = W_1' \otimes W_2'$ with $W_1' \overset{\text{rely}}{\sqsupseteq} W_1, W_2' \overset{\text{rely}}{\sqsupseteq} W_2$ by Lem. 5.
4. Suppose $(W_1' \otimes W_2').k > 0$.
5. $W_1'.k = (W_1' \otimes W_2').k > 0$.
6. Suppose $h; \mathcal{E}_1 \uplus \mathcal{E}_2 \uplus \mathcal{E}_f, \varsigma_{\mathrm{s}} : W_1' \otimes W_2', \eta_1 \otimes \eta_2 \otimes \eta_f$.
7. Suppose $\mathcal{E}' \# \mathcal{E}_f$.
8. $h; \mathcal{E}_1 \uplus \mathcal{E}_2 \uplus \mathcal{E}_f, \varsigma_{\mathrm{s}} : W_1', \eta_1 \otimes \eta_2 \otimes \eta_f$ by Lem. 8.

Case $\boxed{h; \mathcal{E}_1 \uplus \mathcal{E}_2 \to h'; \mathcal{E}'}$

9. Write $\mathcal{E}' = \mathcal{E}_1' \uplus \mathcal{E}_2$ WLOG.
10. $h; \mathcal{E}_1 \to h'; \mathcal{E}_1'$ by nondeterminism of fork.
11. Pick $\varsigma_{\mathrm{s}}', \eta_1'$, and W_1'' with by assumption.
    $W_1'' \overset{\text{guar}}{\sqsupseteq_1} W_1'$,
    $\varsigma_{\mathrm{s}} \to^* \varsigma_{\mathrm{s}}'$,
    $W_1''.k = W_1'.k - 1$,
    $h'; \mathcal{E}_1' \uplus \mathcal{E}_2 \uplus \mathcal{E}_f, \varsigma_{\mathrm{s}}' : W_1'', \eta_1' \otimes \eta_2 \otimes \eta_f$,
    $[\![\mathsf{safe}(\mathcal{E}_1', N_1, x_1.\ Q_1)]\!]^{\rho}_{W_1'', \eta_1'}$
12. Pick $W_2'' \overset{\text{rely}}{\sqsupseteq} W_2'$ with $W_1'' \otimes W_2'' \overset{\text{guar}}{\sqsupseteq} W_1' \otimes W_2'$ by Lem. 6.
13. $(W_1'' \otimes W_2'').k = W_1''.k = W_1'.k - 1 = (W_1' \otimes W_2').k - 1$.
14. $h'; \mathcal{E}_1' \uplus \mathcal{E}_2 \uplus \mathcal{E}_f, \varsigma_{\mathrm{s}}' : W_1'' \otimes W_2'', \eta_1' \otimes \eta_2 \otimes \eta_f$ by Lem. 7.
15. $[\![\mathsf{safe}(\mathcal{E}_2, m_2, x_2.\ Q_2)]\!]^{\rho}_{W_2'', \eta_2}$ by assumption.
16. $[\![\mathsf{safe}(\mathcal{E}_1' \uplus \mathcal{E}_2, m_1, x_1.\ Q_1)]\!]^{\rho}_{W_1'' \otimes W_2'', \eta_1' \otimes \eta_2}$ by induction hypothesis.

Case $\boxed{\mathcal{E}_1 \uplus \mathcal{E}_2 = \mathcal{E}_0 \uplus [n_1 \mapsto v_1]}$

17. $n_1 \in \mathrm{dom}(\mathcal{E}_1)$ by assumption.

18. Write $\mathcal{E}_1 = \mathcal{E}_1' \uplus [n_1 \mapsto v_1]$.
19. Pick $\varsigma_s', \eta_1',$ and W_1'' with — by assumption.
    $W_1'' \overset{\text{guar}}{\sqsupseteq_0} W_1'$,
    $\varsigma_s \to^* \varsigma_s'$,
    $W_1''.k = W_1'.k$,
    $h; \mathcal{E}_1 \uplus \mathcal{E}_2 \uplus \mathcal{E}_f, \varsigma_s' : W_1'', \eta_1' \otimes \eta_2 \otimes \eta_f$,
    $[\![\mathsf{safe}(\mathcal{E}_1', N_1, \mathrm{True}) * Q_1(v_1)]\!]^{\rho}_{W_1'', \eta_1'}$
20. Pick $W_2'' \overset{\text{rely}}{\sqsupseteq} W_2'$ with $W_1'' * W_2'' \overset{\text{guar}}{\sqsupseteq} W_1' * W_2'$ by Lem. 6.
21. $(W_1'' * W_2'').k = W_1''.k = W_1'.k = (W_1' * W_2').k$.
22. $h; \mathcal{E}_1 \uplus \mathcal{E}_2 \uplus \mathcal{E}_f, \varsigma_s' : W_1'' * W_2'', \eta_1' * \eta_2 * \eta_f$ — by Lem. 7.
23. $[\![\mathsf{safe}(\mathcal{E}_2, N_2, x_2.\ Q_2)]\!]^{\rho}_{W_2'', \eta_2}$. — by assumption.
24. $[\![\mathsf{safe}(\mathcal{E}_0, N_1, \mathrm{True}) * Q_1(v_1)]\!]^{\rho}_{W_1'' * W_2'', \eta_1' * \eta_2}$ — by induction hypothesis.

□

Lemma 14 (Sequential Composition)**.** If we have $[\![\mathsf{safe}([N \mapsto e] \uplus M, N, \varphi)]\!]^{\rho}_{W,\eta}$ and, for all v and any W', η' with $W', \eta' \models \varphi(v)$ we have $[\![\mathsf{safe}([N \mapsto M'[v]], N, \varphi')]\!]^{\rho}_{W',\eta'}$ then

$$[\![\mathsf{safe}([N \mapsto M'[e]] \uplus M, N, \varphi')]\!]^{\rho}_{W,\eta}.$$

Proof. Let $i = [\![N]\!]\rho$, $K = [\![M']\!]\rho$, $\mathcal{E}_0 = [\![M]\!]\rho$, $\mathcal{E} = [i \mapsto K[e]] \uplus \mathcal{E}_0$.

The proof proceeds by induction on $W.k$; the case $W.k = 0$ is trivial so we assume $W.k > 0$. We branch on the structure of e:

1. Let $W' \overset{\text{rely}}{\sqsupseteq} W, \eta_f \# \eta, \mathcal{E}_f \# \mathcal{E}$.
2. Suppose $W'.k > 0$.
3. Suppose $h; \mathcal{E} \uplus \mathcal{E}_f, \varsigma_{\mathrm{s}} : W', \eta \otimes \eta_f$.
4. Suppose $\mathcal{E}' \# \mathcal{E}_f$.

Case $\boxed{e = v}$

5. Pick $\varsigma'_{\mathrm{s}}, \eta'$, and W'' with — by assumption.
   $W'' \overset{\text{guar}}{\sqsupseteq}_0 W'$,
   $\varsigma_{\mathrm{s}} \to^* \varsigma'_{\mathrm{s}}$,
   $h; \mathcal{E} \uplus \mathcal{E}_f, \varsigma'_{\mathrm{s}} : W'', \eta' \otimes \eta_f$,
   $W''.k = W'.k$,
   $[\![\mathsf{safe}(\mathcal{E}_0, i, \mathrm{True}) * \varphi(v)]\!]^{\rho}_{W'',\eta'}$
6. Write $W'' = W''_1 \otimes W''_2$ and $\eta' = \eta'_1 \otimes \eta'_2$. with
   $W''_1, \eta'_1 \models^{\rho} \varphi(v)$,
   $[\![\mathsf{safe}(\mathcal{E}_0, i, \mathrm{True})]\!]^{\rho}_{W''_2,\eta'_2}$.
7. $[\![\mathsf{safe}([i \mapsto K[v]], i, \varphi')]\!]^{\rho}_{W''_1,\eta'_1}$ — by assumption.
8. $[\![\mathsf{safe}([i \mapsto K[v]] \uplus \mathcal{E}_0, i, \varphi')]\!]^{\rho}_{W'',\eta'}$ — by Lem. 13.

Case $\boxed{K[v] = v'}$

9. Pick $\varsigma''_{\mathrm{s}}, \eta''$, and W''' with — by (8).
   $W''' \overset{\text{guar}}{\sqsupseteq}_0 W''$,
   $\varsigma'_{\mathrm{s}} \to^* \varsigma''_{\mathrm{s}}$,
   $h; \mathcal{E} \uplus \mathcal{E}_f, \varsigma''_{\mathrm{s}} : W''', \eta'' \otimes \eta_f$,
   $W'''.k = W''.k$,
   $[\![\mathsf{safe}(\mathcal{E}_0, i, \mathrm{True}) * \varphi'(v')]\!]^{\rho}_{W''',\eta''}$
10. $\varsigma_{\mathrm{s}} \to^* \varsigma''_{\mathrm{s}}$.

Case $\boxed{h; [i \mapsto K[v]] \uplus \mathcal{E}_0 \to h'; \mathcal{E}'}$

11. Pick $\varsigma''_{\mathrm{s}}, \eta''$, and W''' with — by (8).
    $W''' \overset{\text{guar}}{\sqsupseteq}_1 W''$,
    $\varsigma'_{\mathrm{s}} \to^* \varsigma''_{\mathrm{s}}$,
    $h'; \mathcal{E}' \cup \mathcal{E}_f, \varsigma''_{\mathrm{s}} : W''', \eta'' \otimes \eta_f$,
    $W'''.k = W''.k - 1$,
    $[\![\mathsf{safe}(\mathcal{E}', i, \varphi')]\!]^{\rho}_{W''',\eta''}$
12. $\varsigma_{\mathrm{s}} \to^* \varsigma''_{\mathrm{s}}$.

Case $\boxed{e \neq v}$

13. Suppose $h; [i \mapsto K[e]] \uplus \mathcal{E}_0 \to h'; [i \mapsto K[e']] \uplus \mathcal{E}'$.
14. $h; [i \mapsto e] \uplus \mathcal{E}_0 \to h'; [i \mapsto e'] \uplus \mathcal{E}'$.
15. Pick $\varsigma_{\mathrm{s}}', \eta',$ and W'' with — by assumption.
    $W'' \overset{\mathrm{guar}}{\sqsupseteq_1} W'$,
    $\varsigma_{\mathrm{s}} \to^* \varsigma_{\mathrm{s}}'$,
    $h'; [i \mapsto e'] \uplus \mathcal{E}' \uplus \mathcal{E}_f, \varsigma_{\mathrm{s}}' : W'', \eta' \otimes \eta_f$,
    $W''.k = W'.k - 1$,
    $[\![\mathsf{safe}([i \mapsto e'] \uplus \mathcal{E}', i, \varphi)]\!]^{\rho}_{W'', \eta'}$
16. $[\![\mathsf{safe}([i \mapsto K[e']] \uplus \mathcal{E}', i, \varphi')]\!]^{\rho}_{W'', \eta'}$ — by induction hypothesis.

□

8.3 Soundness of key proof rules

The soundness of Frame, Bind, and Fork follow easily from the lemmas proved in the previous section. Here we give proofs for the key rules dealing with islands and lifting atomic triples.

Lemma 15.

$$\frac{\mathcal{C} \vdash \{P\}\ i \Rrightarrow e\ \{\varphi * \triangleright \pi[\![b]\!]\}}{\mathcal{C} \vdash \{P\}\ i \Rrightarrow e\ \left\{\varphi * \exists n.\ \boxed{b}^n_\pi\right\}}\ \textsc{NewIsl}$$

Let $(s, T) = [\![p]\!]$ and let $(\theta, \varphi') = \pi$. We prove the result by a straightforward induction on the step index in the underlying safety definition. Thus, the proof boils down to the following internal lemma: if

$$h; \mathcal{E} \uplus \mathcal{E}_f, \varsigma_\mathrm{s} : W, \eta \otimes \eta_F \quad \text{and} \quad \mathcal{E} = \mathcal{E}_0 \uplus [i \mapsto v] \quad \text{and} \quad W, \eta \models^\rho \varphi(v) * \triangleright\varphi'(s) * \mathsf{safe}(\mathcal{E}_0, i, \mathrm{True})$$

then

$$\exists \eta', W' \overset{\mathrm{guar}}{\sqsupseteq_0} W.\ h; \mathcal{E} \uplus \mathcal{E}_f, \varsigma_\mathrm{s} : W', \eta' \otimes \eta_F \quad \text{and} \quad W', \eta' \models^\rho \varphi(v) * \exists n.\ \boxed{b}^n_\pi * \mathsf{safe}(\mathcal{E}_0, i, \mathrm{True})$$

Proof.

1. Let $n \in \mathbb{N} \setminus \mathrm{dom}(W.\omega)$.
2. $W = W_1 \otimes W_2 \otimes W_3, \quad \eta = \eta_1 \otimes \eta_2 \otimes \eta_3,$
   $W_1, \eta_1 \models^\rho \varphi(v) \quad W_2, \eta_2 \models^\rho \triangleright\varphi'(s), \quad W_3, \eta_3 \models^\rho \mathsf{safe}(\mathcal{E}_0, i, \mathrm{True}).$
3. Let $W' = (W.k, W.\omega \uplus [n \mapsto (\theta, [\![\varphi']\!]^\rho_-, (s, T))], W.B)$
4. $W' \overset{\mathrm{guar}}{\sqsupseteq_0} W$
5. $\triangleright|W'|, \eta_2 \models^\rho \triangleright\varphi'(s)$
6. Let $\eta' = \eta_1 \otimes \eta_3$
7. $h; \mathcal{E} \uplus \mathcal{E}_f, \varsigma_\mathrm{s} : W', \eta' \otimes \eta_F$
8. $W_2 \otimes W_3, \eta_3 \models^\rho \mathsf{safe}(\mathcal{E}_0, i, \mathrm{True})$ by framing
9. $W', \eta' \models^\rho \varphi(v) * \exists n.\ \boxed{b}^n_\pi * \mathsf{safe}(\mathcal{E}_0, i, \mathrm{True})$

□

Lemma 16.

$$\frac{\mathcal{C} \vdash (\!|P|\!)\ i \Rrightarrow_{\mathrm{I}} a\ (\!|Q|\!)}{\mathcal{C} \vdash \{\triangleright P\}\ i \Rrightarrow a\ \{Q\}}\ \textsc{Private}$$

We show $(\!|P|\!)\ i \Rrightarrow_{\mathrm{I}} a\ (\!|Q|\!) \Rightarrow \{\triangleright P\}\ i \Rrightarrow a\ \{Q\}$. Suppose for notational simplicity that the assertions are variable-closed (allowing us, e.g., to write i instead of $[\![i]\!]^\rho$).

Proof.

1. Suppose $W_0, \eta \models^\rho (\!|P|\!)\ i \Rrightarrow_{\mathrm{I}} a\ (\!|Q|\!)$
2. Suppose $|W_0|, \emptyset \models^\rho \triangleright P \Rightarrow \mathsf{safe}([i \mapsto a], i, Q)$
3. Suppose $W \geq |W_0| \quad \text{and} \quad \eta \geq \emptyset$
4. Suppose $W, \eta \models^\rho \triangleright P$
5. Suppose $W' \overset{\text{rely}}{\sqsupseteq} W, \quad W'.k > 0,$
   $\eta_F \# \eta, \quad \mathcal{E} = [i \mapsto a], \quad \mathcal{E}_F \# \mathcal{E}, \quad \mathcal{E}' \# \mathcal{E}_F,$
   $h; \mathcal{E} \uplus \mathcal{E}_F, \varsigma_{\mathrm{S}} : W', \eta \otimes \eta_F, \quad h; [i \mapsto a] \to h'; \mathcal{E}'$
6. $\exists v. \quad h; a \xrightarrow{i} h'; v \quad \text{and} \quad \mathcal{E}' = [i \mapsto v].$
7. $\exists \overline{\eta_i}. \quad \forall i \in \mathrm{dom}(W'.\omega).\ \eta_i \in \mathrm{interp}(W'.\omega(i))(\triangleright |W'|)$
8. Let $\eta'_F = \eta_F \otimes \overline{\eta_i}$
9. $\varsigma_{\mathrm{S}} = (\eta \otimes \eta'_F).\varsigma$
10. $\triangleright W', \eta \models^\rho P$
11. $(\eta \otimes \eta'_F).h; a \xrightarrow{i} h'; v$
12. Let $h' = (\eta' \otimes \eta'_F).h$
13. $\exists \eta' \# \eta'_F. \quad (\eta \otimes \eta'_F).\varsigma \to^* (\eta' \otimes \eta'_F).\varsigma$
    $(\eta \otimes \eta'_F).I = (\eta' \otimes \eta'_F).I$
    $[\![Q(v)]\!]^\rho_{\triangleright W', \eta'}$
14. Let $\varsigma'_{\mathrm{S}} = (\eta' \otimes \eta_F).\varsigma$
15. $\triangleright W' \overset{\text{guar}}{\sqsupseteq_1} W'$
16. $h'; [i \mapsto v], \varsigma'_{\mathrm{S}} : \triangleright W', \eta' \otimes \eta'_F$
17. $[\![\mathsf{safe}([i \mapsto v], i, Q)]\!]^\rho_{\triangleright W', \eta'}$

□

Lemma 17.

$$\frac{\mathcal{C} \vdash \forall b \overset{\text{rely}}{\sqsupseteq}_\pi b_0.\ \exists b' \overset{\text{guar}}{\sqsupseteq}_\pi b.\ (\!|\pi[\![b]\!] * P|\!)\ i \Rrightarrow_{\text{I}} a\ (\!|x.\ \triangleright\, \pi[\![b']\!] * Q|\!)}{\mathcal{C} \vdash \left\{\boxed{b_0}^n_\pi * \triangleright P\right\}\ i \Rrightarrow a\ \left\{x.\ \exists b'.\ \boxed{b'}^n_\pi * Q\right\}}\ \textsc{UpdIsl}$$

Suppose for notational simplicity that the assertions are variable-closed (allowing us, e.g., to write i instead of $[\![i]\!]^\rho$).

Proof.

1. Suppose $W_0, \eta \models^\rho \mathcal{C}$
2. Let $\mathcal{E} = [i \mapsto a]$
3. Suppose $|W_0|, \eta \models^\rho \boxed{b_0}^n_\pi * \triangleright P$
4. Suffices to show $|W_0|, \eta \models^\rho \mathsf{safe}([i \mapsto a], i, \exists b'.\ \boxed{b'}^n_\pi * Q)$
5. Fix $W \overset{\text{rely}}{\sqsupseteq} |W_0|$ and $\eta_F \# \eta$ and $\mathcal{E}_F \# [i \mapsto a]$
6. Suppose $W.k > 0$ and $h; \mathcal{E} \uplus \mathcal{E}_F, \varsigma_{\text{S}} : W, \eta \otimes \eta_F$ and $\mathcal{E}' \# \mathcal{E}_F$
7. Let $\iota_0 = (\pi.\theta, \pi.\varphi, b_0)$
8. $\exists W_1, W_2, \eta_1, \eta_2.\ W = W_1 \otimes W_2$
   $\eta = \eta_1 \otimes \eta_2$
   $W_1, \eta_1 \models^\rho \iota_0$
   $W_2, \eta_2 \models^\rho \triangleright P$
9. $\exists \iota \overset{\text{rely}}{\sqsupseteq} \iota_0, \omega_F.\ W_1.\omega = \omega_F \uplus [n \mapsto \iota]$
10. $\exists \eta_\iota, \eta'_F, I.\ \eta \otimes \eta'_F \otimes \eta_\iota \otimes \eta_F = (h, \varsigma_{\text{S}}, I)$
    $\eta_\iota \in \text{interp}(\iota)(\triangleright |W_1|)$
    $\eta'_F \in \text{interp}(\omega_F)(\triangleright |W_1|)$
11. $\triangleright |W_1|, \eta_\iota \models^\rho \iota.\varphi(\iota.s)$
12. Suppose $h; \mathcal{E} \to h'; \mathcal{E}'$
13. $\exists v.\ \mathcal{E}' = [i \mapsto v]$
14. Suppose $W.k > 1$ WLOG
15. Let $b = (\iota.s, \iota.T)$
16. $\pi.\theta \vdash b \overset{\text{rely}}{\sqsupseteq}_\pi b_0$
17. $\exists b'.\ \pi.\theta \vdash b' \overset{\text{guar}}{\sqsupseteq}_\pi b$ by assumption
    $(\!|\pi[\![b]\!] * P|\!)\ i \Rrightarrow_{\text{I}} a\ (\!|x.\ \triangleright\, \pi[\![b']\!] * Q|\!)$
18. $\triangleright W, \eta_\iota \otimes \eta_2 \models^\rho \pi[\![b]\!] * P$
19. Let $\widehat{\eta}_F = \eta'_F \otimes \eta_F \otimes \eta_1$
20. Let $\widehat{\eta} = \eta_\iota \otimes \eta_2$
21. $(\widehat{\eta} \otimes \widehat{\eta}_F).h; a \xrightarrow{i} h'; v$
22. $\exists \widehat{\eta}' \# \widehat{\eta}_F.\ h' = (\widehat{\eta}' \otimes \widehat{\eta_F}).h$
    $(\widehat{\eta} \otimes \widehat{\eta_F}).\varsigma \to^* (\widehat{\eta}' \otimes \widehat{\eta_F}).\varsigma$
    $\widehat{\eta}.I = \widehat{\eta}'.I$
23. $\triangleright W, \widehat{\eta}' \models^\rho x.\ \triangleright\, \pi[\![b']\!] * Q$
24. Let $W' = (W.k - 1, \lfloor \omega.F \rfloor_{W.k-1} \uplus [n \mapsto (\pi.\theta, \pi.\varphi, b')])$
25. $W' \overset{\text{guar}}{\sqsupseteq}_1 W$

26. $W', \widehat{\eta}' \models^{\rho} x.\pi[\![b']\!] * Q$
27. $\varsigma_{\mathrm{s}} \rightarrow^{*} (\widehat{\eta}' \otimes \widehat{\eta_F}).\varsigma$
28. $h'; \mathcal{E}' \uplus \mathcal{E}_F, (\widehat{\eta}' \otimes \widehat{\eta_F}).\varsigma : W', (\widehat{\eta}' \otimes \eta'_F \otimes \eta_1) \otimes \eta_F$
29. $W', \widehat{\eta}' \otimes \eta'_F \otimes \eta_1 \models^{\rho} \mathsf{safe}(\mathcal{E}', i, x.\exists b'.\ \boxed{b'}^{n}_{\pi} * Q)$

□