Actris: Session-Type Based Reasoning in Separation Logic

Jonas Kastberg Hinrichsen, ITU

Joint work with
Jesper Bengtson, ITU
Robbert Krebbers, TU Delft

28. October 2019
Iris Workshop, Aarhus

The Actor Model and Message Passing

- Principled way of writing concurrent programs
  - Isolation of concurrent behaviour
  - Threads as services and clients
  - Used in Erlang, Elixir, Go, Java, Scala, F# and C#
- Primitives

  $\texttt{new\_chan}\ (),\ \texttt{send}\ \mathsf{c}\ \mathsf{v},\ \texttt{recv}\ \mathsf{c}$

- Example: $\begin{array}{l}\texttt{let}\ (c, c') = \texttt{new\_chan}\ ()\ \texttt{in} \\ \texttt{fork}\ \{\texttt{send}\ c'\ 42\}\,; \\ \texttt{recv}\ c\end{array}$
- Many variants exists
  - In our case: Asynchronous, Order-Preserving and Reliable

Problem

- Message-Passing is not a silver bullet for concurrency

  > "We studied 15 large, mature, and actively maintained actor programs written in Scala and found that 80% of them mix the actor model with another concurrency model." [Tasharofi et al., ECOOP'13]

- No existing solution where high-level actor-based reasoning is readily available in combination with existing concurrency models for functional verification
  - Allowing communication of references, channels and higher-order functions

Key Idea

Combine

- Session Types [Honda et al., ESOP'98]
  - Type system for channels
  - Example: $!\mathbb{N}.?\mathbb{B}.\mathsf{end}$
  - Ensures safety through static type checking
- Concurrent Separation Logic [O'Hearn & Brooks, CONCUR'04]
  - Logic for reasoning about concurrent programs with mutable state.
  - Example: $\{x \mapsto a * y \mapsto b\}\, \mathtt{swap}\ x\ y\, \{x \mapsto b * y \mapsto a\}$
  - Ensures functional correctness through manual proofs

Contributions

Actris: A concurrent separation logic for proving *functional correctness* of programs that combine *message passing* with other programming and concurrency paradigms

- We introduce the notion of *Dependent Separation Protocols*
- Integration with Iris and its existing concurrency mechanisms, e.g. locks and ghost state
- Verification of feature-heavy programs including a variant of Map-Reduce
- A full mechanization of all of the above in Coq with tactic support

Demonstration

Demonstration

Language: Iris's ML-like language extended with message-passing primitives

$$e \in \text{Expr} ::= \texttt{new\_chan}\ () \mid \texttt{send}\ c\ v \mid \texttt{recv}\ c \mid \ldots$$

Program

$$\begin{array}{l} \texttt{let}\ (c, c') = \texttt{new\_chan}\ ()\ \texttt{in} \\ \texttt{fork}\ \{\texttt{send}\ c'\ 42\}\ ; \\ \texttt{recv}\ c \end{array}$$

Dependent Separation Protocols

Definition

$$
\begin{aligned}
\mathit{prot} \triangleq\ & \mathbf{!}\, \vec{x} : \vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot} \quad \mid \\
& \mathbf{?}\, \vec{x} : \vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot} \quad \mid \\
& \mathsf{end}
\end{aligned}
$$

Example

$$\mathbf{!}\, (x : \mathbb{N})\, \langle x \rangle \{\mathsf{True}\}.\, \mathbf{?}\, (b : \mathbb{B})\, \langle b \rangle \{b = \mathit{is\_even}\ x\}.\, \mathsf{end}$$

Duality

$$
\begin{aligned}
\overline{\mathbf{!}\, \vec{x} : \vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot}} &= \mathbf{?}\, \vec{x} : \vec{\tau}\, \langle v \rangle \{P\}.\, \overline{\mathit{prot}} \\
\overline{\mathbf{?}\, \vec{x} : \vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot}} &= \mathbf{!}\, \vec{x} : \vec{\tau}\, \langle v \rangle \{P\}.\, \overline{\mathit{prot}} \\
\overline{\mathsf{end}} &= \mathsf{end}
\end{aligned}
$$

Ownership

$$c \rightarrowtail \mathit{prot}$$

Dependent Separation Protocols - Rules

Ht-NewChan

$\{\text{True}\}$

$\quad \texttt{new\_chan}\ ()$

$\{(c, c').\ c \rightarrowtail prot * c' \rightarrowtail \overline{prot}\}$

Ht-Send

$\{c \rightarrowtail \mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\ prot * P[\vec{t}/\vec{x}]\}$

$\quad \texttt{send}\ c\ (v[\vec{t}/\vec{x}])$

$\{c \rightarrowtail prot[\vec{t}/\vec{x}]\}$

Ht-Recv

$\{c \rightarrowtail \mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\ prot\}$

$\quad \texttt{recv}\ c$

$\{w.\ \exists \vec{y}.\ (w = v[\vec{y}/\vec{x}]) * c \rightarrowtail prot[\vec{y}/\vec{x}] * P[\vec{y}/\vec{x}]\}$

Demonstration - Verified

Program

$$\begin{array}{ll}
\texttt{let}\ (c, c') = \texttt{new_chan}\ ()\ \texttt{in} & (*\ \{\mathsf{True}\}\ _\ \{c \rightarrowtail \mathbf{?}\,\langle 42\rangle\{\mathsf{True}\}.\,\mathsf{end} * _\}\ *) \\
\texttt{fork}\ \{\texttt{send}\ c'\ 42\}\,; & (*\ \{c' \rightarrowtail \mathbf{!}\,\langle 42\rangle\{\mathsf{True}\}.\,\mathsf{end}\}\ _\ \{c' \rightarrowtail \mathsf{end}\}\ *) \\
\texttt{recv}\ c & (*\ \{c \rightarrowtail \mathbf{?}\,\langle 42\rangle\{\mathsf{True}\}.\,\mathsf{end}\}\ _\ \{v.\ c \rightarrowtail \mathsf{end} * v = 42\}\ *)
\end{array}$$

Protocol

$$c \rightarrowtail \mathbf{?}\,\langle 42\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$c' \rightarrowtail \mathbf{!}\,\langle 42\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Demonstration - References

Program

$$
\begin{array}{ll}
\texttt{let}\ (c, c') = \texttt{new\_chan}\ ()\ \texttt{in} & (*\ \{\mathsf{True}\}\ \_\ \{c \rightarrowtail \mathbf{?}\,\ell\,\langle\ell\rangle\{\ell \mapsto 42\}.\,\mathsf{end} * \_\}\ *) \\
\texttt{fork}\ \{\texttt{send}\ c'\ (\texttt{ref}\ 42)\}\,; & (*\ \{c' \rightarrowtail \mathbf{!}\,\ell\,\langle\ell\rangle\{\ell \mapsto 42\}.\,\mathsf{end} * \ell \mapsto 42\}\ \_\ \{c' \rightarrowtail \mathsf{end}\}\ *) \\
!\,(\texttt{recv}\ c) & (*\ \{c \rightarrowtail \mathbf{?}\,\ell\,\langle\ell\rangle\{\ell \mapsto 42\}.\,\mathsf{end}\}\ \_\ \{\ell.\ c \rightarrowtail \mathsf{end} * \ell \mapsto 42\}\ *)
\end{array}
$$

Protocol

$$
\begin{array}{ll}
c \rightarrowtail \mathbf{?}\,\ell\,\langle\ell\rangle\{\ell \mapsto 42\}.\,\mathsf{end} & \quad \texttt{and} \\
c' \rightarrowtail \mathbf{!}\,\ell\,\langle\ell\rangle\{\ell \mapsto 42\}.\,\mathsf{end} &
\end{array}
$$

Demonstration - Delegation

Delegation: Passing channels over channels

Program

$$\texttt{let}\ (c_1, c_1') = \texttt{new\_chan}\ ()\ \texttt{in}$$
$$\texttt{fork}\left\{\begin{array}{l}\texttt{let}\ (c_2, c_2') = \texttt{new\_chan}\ ()\ \texttt{in}\\ \texttt{send}\ c_1'\ c_2; \texttt{send}\ c_2'\ (\texttt{ref}\ 42)\end{array}\right\};$$
$$!\,(\texttt{recv}\ (\texttt{recv}\ c_1))$$

Protocols

$$c_1 \rightarrowtail \mathbf{?}\, c\, \langle c \rangle \{ c \rightarrowtail \mathbf{?}\, \ell\, \langle \ell \rangle \{\ell \mapsto 42\}.\, \mathsf{end} \}.\, \mathsf{end} \qquad \texttt{and}$$
$$c_1' \rightarrowtail \mathbf{!}\, c\, \langle c \rangle \{ c \rightarrowtail \mathbf{?}\, \ell\, \langle \ell \rangle \{\ell \mapsto 42\}.\, \mathsf{end} \}.\, \mathsf{end}$$

$$c_2 \rightarrowtail \mathbf{?}\, \ell\, \langle \ell \rangle \{\ell \mapsto 42\}.\, \mathsf{end} \qquad \texttt{and}$$
$$c_2' \rightarrowtail \mathbf{!}\, \ell\, \langle \ell \rangle \{\ell \mapsto 42\}.\, \mathsf{end}$$

Demonstration - Higher-Order

Program

$\texttt{let}\,(c, c') = \texttt{new\_chan}\;()\;\texttt{in}$
$\texttt{fork}\;\{\texttt{let}\,f = \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(\lambda\,().\,f() + 2)\}\,;$
$\texttt{let}\,r = \texttt{ref}\;40\;\texttt{in}\,\texttt{send}\;c\;(\lambda\,().\;!\,r); \texttt{recv}\;c\;()$

Protocol

$$\begin{array}{ll} c \rightarrowtail & \mathbf{!}\,P\,Q\,f\,\langle f\rangle\{\{P\}\,f\;()\;\{v.\,Q(v)\}\}.\\ & \mathbf{?}\,g\,\langle g\rangle\{\{P\}\,g\;()\;\{v.\,\exists w.\,(v = w + 2) * Q(w)\}\}.\\ & \mathsf{end} \end{array} \qquad \texttt{and}$$

$$\begin{array}{ll} c' \rightarrowtail & \mathbf{?}\,P\,Q\,f\,\langle f\rangle\{\{P\}\,f\;()\;\{v.\,Q(v)\}\}.\\ & \mathbf{!}\,g\,\langle g\rangle\{\{P\}\,g\;()\;\{v.\,\exists w.\,(v = w + 2) * Q(w)\}\}.\\ & \mathsf{end} \end{array}$$

Hoare Triple for initial closure

$$\{r \mapsto 40\}\;(\lambda().\,!\,r)\;()\;\{v.v = 40\}$$

Demonstration - Locks

Program

$$\begin{array}{l}
\texttt{let}\,(c, c') = \texttt{new_chan}\;()\;\texttt{in} \\
\texttt{let}\; lk = \texttt{new_lock}\;()\;\texttt{in} \\
\texttt{fork}\;\{\texttt{acquire}\; lk; \texttt{send}\; c'\; 21;\; \texttt{release}\; lk\}\,; \\
\texttt{fork}\;\{\texttt{acquire}\; lk; \texttt{send}\; c'\; 21;\; \texttt{release}\; lk\}\,; \\
\texttt{recv}\; c + \texttt{recv}\; c
\end{array}$$

Protocol

$$\texttt{lock\_prot} \triangleq \mu\,(rec : \mathbb{N} \to \text{iProto}).\, \lambda\, n.\, \texttt{if}\; n = 0\; \texttt{then}\; \text{end}\; \texttt{else}\; \mathbf{?}\,\langle 21 \rangle.rec\; (n-1)$$

$$c \rightarrowtail \texttt{lock\_prot}\; 2 \qquad \text{and} \qquad c' \rightarrowtail \overline{\texttt{lock\_prot}\; 2}$$

Hoare Triple for critical section

$$\left\{\text{is\_lock}\; lk\; (\exists n.\, c' \rightarrowtail \overline{\texttt{lock\_prot}\; n} * \boxed{\bullet\, n : \textsc{Auth}(\mathbb{N})}^{\gamma}) * \boxed{\circ\, 1 : \textsc{Auth}(\mathbb{N})}^{\gamma}\right\}$$

$$\texttt{acquire}\; lk; \texttt{send}\; c'\; 21;\; \texttt{release}\; lk$$

$$\{\text{True}\}$$

Model

The Model of Actris - Channels

Channels encoded directly on top of HeapLang as a pair of lock-protected buffers

$(c_1, c_2) \rightarrowtail (\vec{v_1}, \vec{v_2})$

The rules enqueue and dequeue as one would expect

$$
\begin{array}{rll}
\{\mathsf{True}\} & \texttt{new\_chan}\ () & \{(c_1, c_2).\ (c_1, c_2) \rightarrowtail (\epsilon, \epsilon)\} \\
\{(c_1, c_2) \rightarrowtail (\vec{v_1}, \vec{v_2})\} & \texttt{send}\ c_1\ w & \{(c_1, c_2) \rightarrowtail (\vec{v_1} \cdot [w], \vec{v_2})\} \\
\{(c_1, c_2) \rightarrowtail (\vec{v_1}, \vec{v_2})\} & \texttt{send}\ c_2\ w & \{(c_1, c_2) \rightarrowtail (\vec{v_1}, \vec{v_2} \cdot [w])\} \\
\{(c_1, c_2) \rightarrowtail (\vec{v_1}, \vec{v_2})\} & \texttt{recv}\ c_1 & \{w.\ (\vec{v_2} = [w] \cdot \vec{w}) * (c_1, c_2) \rightarrowtail (\vec{v_1}, \vec{w})\} \\
\{(c_1, c_2) \rightarrowtail (\vec{v_1}, \vec{v_2})\} & \texttt{recv}\ c_2 & \{w.\ (\vec{v_1} = [w] \cdot \vec{w}) * (c_1, c_2) \rightarrowtail (\vec{w}, \vec{v_2})\}
\end{array}
$$

The Model of Actris - Dependent Separation Protocols

Defined in Continuation-Passing Style to allow use of quantifiers and have positive position of recursive occurrence

$$
\begin{array}{rcl}
\mathrm{iProto} & \cong & 1 + (\mathbb{B} \times (\mathrm{Val} \to (\blacktriangleright \mathrm{iProto} \to \mathrm{iProp}) \to \mathrm{iProp})) \\
\mathrm{end} & \triangleq & \mathtt{inj}_1\,() \\
\mathbf{!}\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,\mathit{prot} & \triangleq & \mathtt{inj}_2\,(\mathtt{true}, \lambda\, w\,(f : \blacktriangleright \mathrm{iProto} \to \mathrm{iProp}).\, \exists(\vec{x}:\vec{\tau}).\,(v = w) * \triangleright P * f(\mathrm{next}\ \mathit{prot})) \\
\mathbf{?}\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,\mathit{prot} & \triangleq & \mathtt{inj}_2\,(\mathtt{false}, \lambda\, w\,(f : \blacktriangleright \mathrm{iProto} \to \mathrm{iProp}).\, \exists(\vec{x}:\vec{\tau}).\,(v = w) * \triangleright P * f(\mathrm{next}\ \mathit{prot}))
\end{array}
$$

Supplied function always chosen as an equivalence

$$f \triangleq (\lambda\, \mathit{prot}'.\, \mathit{prot}' = \mathrm{next}\ \mathit{prot})$$

The Model of Actris - Endpoint Ownership

Protocol ownership "$c \rightarrowtail \mathit{prot}$" encoded via ghost state and invariants

$$c \rightarrowtail \mathit{prot} \triangleq \exists \gamma_1\, \gamma_2\, c_1\, c_2.\, ((c = c_1 * \gamma_1 \mapsto_\circ \mathit{prot}) \vee (c = c_2 * \gamma_2 \mapsto_\circ \mathit{prot})) * \boxed{I\ \gamma_1\ \gamma_2\ c_1\ c_2}$$

Protocol Invariant - One buffer is always empty

$$\begin{aligned} I\ \gamma_1\ \gamma_2\ c_1\ c_2 \triangleq \exists \vec{v}_1\, \vec{v}_2\, \mathit{prot}_1\, \mathit{prot}_2.\, & (c_1, c_2) \rightarrowtail (\vec{v}_1, \vec{v}_2) * \\ & \gamma_1 \mapsto_\bullet \mathit{prot}_1 * \gamma_2 \mapsto_\bullet \mathit{prot}_2 * \\ & \triangleright \begin{pmatrix} (\vec{v}_2 = \epsilon * \text{interp } \vec{v}_1\ \mathit{prot}_1\ \mathit{prot}_2) \vee \\ (\vec{v}_1 = \epsilon * \text{interp } \vec{v}_2\ \mathit{prot}_2\ \mathit{prot}_1) \end{pmatrix} \end{aligned}$$

Duality and ownership of resources in transit

$$\text{interp } \epsilon\ \mathit{prot}_1\ \mathit{prot}_2 \triangleq \mathit{prot}_1 = \overline{\mathit{prot}_2}$$

$$\begin{aligned} \text{interp } ([v] \cdot \vec{v})\ \mathit{prot}_1\ \mathit{prot}_2 \triangleq \exists \Phi\ \mathit{prot}_2'.\, & (\mathit{prot}_2 = \texttt{inj}_2\, (\texttt{false}, \Phi)) * \\ & \Phi\ v\ (\lambda\, \mathit{prot}'.\, \mathit{prot}' = \text{next } \mathit{prot}_2') * \\ & \triangleright \text{interp } \vec{v}\ \mathit{prot}_1\ \mathit{prot}_2' \end{aligned}$$

Future work

- Semantic model of Session Types via logical relations

  $[\![\_]\!] : \tau \to \mathsf{Val} \to \mathsf{iProp}$
  $[\![\mathbb{N}]\!] \triangleq \lambda v.\, \exists n \in \mathbb{N}.\, v = n$
  $[\![st]\!] \triangleq ???$

- Multi-party Dependent Separation Protocols (Based on [Honda et al., POPL'08])
- Linearity of channels through Iron
  - Preventing dropping of channel obligation
- Communication between distributed systems

Conclusion

Actris: A concurrent separation logic for proving *functional correctness* of programs that combine *message passing* with other programming and concurrency paradigms

- We introduce the notion of *Dependent Separation Protocols*
- Integration with Iris and its existing concurrency mechanisms, e.g. locks and ghost state
- Verification of feature-heavy programs including a variant of Map-Reduce
- A full mechanization of all of the above in Coq with tactic support
  - https://gitlab.mpi-sws.org/iris/actris
- A paper on Actris has been conditionally accepted for POPL'20.

Extras

Distributed Merge Sort

$$
\begin{array}{l}
\texttt{sort\_service}\ \mathit{cmp}\ c\ \triangleq \\
\quad \texttt{let}\ l = \texttt{recv}\ c\ \texttt{in} \\
\quad \texttt{if}\ |l| \leq 1\ \texttt{then}\ \texttt{send}\ c\ ()\ \texttt{else} \\
\quad \texttt{let}\ l' = \texttt{split}\ l\ \texttt{in} \\
\quad \texttt{let}\ c_1 = \texttt{start}\ (\texttt{sort\_service}\ \mathit{cmp})\ \texttt{in} \\
\quad \texttt{let}\ c_2 = \texttt{start}\ (\texttt{sort\_service}\ \mathit{cmp})\ \texttt{in} \\
\quad \texttt{send}\ c_1\ l;\ \texttt{send}\ c_2\ l'; \\
\quad \texttt{recv}\ c_1;\ \texttt{recv}\ c_2; \\
\quad \texttt{merge}\ \mathit{cmp}\ l\ l';\ \texttt{send}\ c\ ()
\end{array}
$$

$$
\begin{array}{l}
\texttt{start}\ e \triangleq \\
\quad \texttt{let}\ f = e\ \texttt{in} \\
\quad \texttt{let}\ (c, c') = \texttt{new\_chan}\ ()\ \texttt{in} \\
\quad \texttt{fork}\ \{f\ c'\};\ c
\end{array}
$$

$$
\begin{array}{l}
\mathsf{sort\_prot}\ (I : T \to \mathsf{Val} \to \mathsf{iProp})\ (R : T \to T \to \mathbb{B}) \triangleq \\
\quad \mathbf{?}\, \vec{x}\ \ell\, \langle \ell \rangle \{ \ell \mapsto_I \vec{x} \}. \\
\quad \mathbf{!}\, \vec{y}\, \langle () \rangle \{\ \ell \mapsto_I \vec{y} * \mathsf{sorted\_of}_R\ \vec{y}\ \vec{x}\ \}.\, \mathsf{end}
\end{array}
$$

$$
\begin{array}{l}
\left\{ \begin{array}{l} \mathsf{cmp\_spec}\ I\ R\ \mathit{cmp}\ * \\ c \rightarrowtail \mathsf{sort\_prot}\ I\ R \end{array} \right\} \\
\quad \texttt{sort\_service}\ \mathit{cmp}\ c \\
\{ c \rightarrowtail \mathsf{end} \}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{cmp\_spec}\ I\ R\ \mathit{cmp} \triangleq \\
\quad \forall x_1\ x_2\ v_1\ v_2.\ \{ I\ x_1\ v_1 * I\ x_2\ v_2 \} \\
\qquad\qquad \mathit{cmp}\ v_1\ v_2 \\
\qquad\qquad \{ r.\ r = R\ x_1\ x_2 * I\ x_1\ v_1 * I\ x_2\ v_2 \}
\end{array}
$$

Fine-Grained Merge Sort

```
sort_service_fg cmp c ≜
  branch c with
   right ⇒ select c right
  | left  ⇒
    let x1 = recv c in
    branch c with
     right ⇒ select c left;
             send c x1;
             select c right
    | left  ⇒
      let x2 = recv c in
      let c1 = start sort_service_fg cmp in
      let c2 = start sort_service_fg cmp in
      select c1 left; send c1 x1;
      select c2 left; send c2 x2;
      split_fg c c1 c2; merge_fg cmp c c1 c2
    end
  end
```

$$\mathsf{sort\_prot}_{fg}\ (I : T \to \mathsf{Val} \to \mathsf{iProp})\ (R : T \to T \to \mathbb{B}) \triangleq$$
$$\quad \mathsf{sort\_prot}^{\mathtt{head}}_{fg}\ I\ R\ \epsilon$$

$$\mathsf{sort\_prot}^{\mathtt{head}}_{fg}\ I\ R \triangleq$$
$$\quad \mu\,(rec : \mathsf{List}\ T \to \mathsf{iProto}).$$
$$\qquad \lambda\,\vec{x}.\,(\mathbf{?}\,x\ v\,\langle v\rangle\{I\ x\ v\}.\ rec\ (\vec{x}\cdot[x]))$$
$$\qquad\quad \&\ \ \mathsf{sort\_prot}^{\mathtt{tail}}_{fg}\ I\ R\ \vec{x}\ \epsilon$$

$$\mathsf{sort\_prot}^{\mathtt{tail}}_{fg}\ I\ R \triangleq$$
$$\quad \mu\,(rec : \mathsf{List}\ T \to \mathsf{List}\ T \to \mathsf{iProto}).$$
$$\qquad \lambda\,\vec{x}\ \vec{y}.\,(\mathbf{!}\,y\ v\,\langle v\rangle\{(\forall i < |\vec{y}|.\ R\ \vec{y}_i\ y) * I\ y\ v\}.\ rec\ \vec{x}\ (\vec{y}\cdot[y]))$$
$$\qquad\quad {}_{\{\mathsf{True}\}}\!\oplus_{\{\vec{x}\equiv_{\mathrm{p}}\vec{y}\}}\ \ \mathsf{end}$$

Distributed Mapper

```
mapper_worker f_v lk c ≜
  acquire lk; select c left;
  branch c with
    right ⇒ release lk
  | left  ⇒ let x := recv c in release lk;                          (* acquire work *)
            let y := f_v x in                                       (* map it *)
            acquire lk; select c right; send c y; release lk;       (* send it back *)
            mapper_worker f_v lk c
  end
```

$$\begin{array}{l}
\mathrm{mapper_prot}\ I_T\ I_U\ (f : T \to \mathrm{List}\ U) \triangleq \\
\quad \mu\, rec.\, \lambda\, (n : \mathbb{N})\ (X : \mathrm{MultiSet}\, T). \\
\qquad \texttt{if}\ n = 0\ \texttt{then}\ \mathrm{end}\ \texttt{else} \\
\qquad (\mathbf{?}\, x\ v\, \langle v \rangle \{ I_T\ x\ v \}.\, rec\ n\ (X \uplus \{x\}))\ \ \&\ \ rec\ (n-1)\ X \\
\qquad\quad {}_{\{(n=1) \Rightarrow (X = \emptyset)\}} \oplus_{\{\mathrm{True}\}} \\
\qquad \mathbf{!}\, x\ \ell\, \langle \ell \rangle \{ x \in X * \ell \mapsto_{I_U} (f\ x) \}.\, rec\ n\ (X \setminus \{x\})
\end{array}$$

$$\begin{array}{l}
prot_1\ {}_{\{Q_1\}} \oplus_{\{Q_2\}}\ prot_2 \triangleq \\
\quad \mathbf{!}\, (b : \mathbb{B})\, \langle b \rangle \{ \texttt{if}\ b\ \texttt{then}\ Q_1\ \texttt{else}\ Q_2 \}. \\
\quad \texttt{if}\ b\ \texttt{then}\ prot_1\ \texttt{else}\ prot_2
\end{array}$$

$$\begin{array}{l}
prot_1\ {}_{\{Q_1\}} \&_{\{Q_2\}}\ prot_2 \triangleq \\
\quad \mathbf{?}\, (b : \mathbb{B})\, \langle b \rangle \{ \texttt{if}\ b\ \texttt{then}\ Q_1\ \texttt{else}\ Q_2 \}. \\
\quad \texttt{if}\ b\ \texttt{then}\ prot_1\ \texttt{else}\ prot_2
\end{array}$$

Distributed Mapper - Specification

Concurrent Spec of Mapper Worker

$$\left\{\begin{array}{l}\text{f\_spec } I_T \ I_U \ f \ f_v * \text{contrib}_\gamma \ \emptyset \ * \\ \text{is\_lock } lk \ \left(\begin{array}{l}\exists n \ X. \text{auth}_\gamma \ n \ X \ * \\ c \rightarrowtail \overline{\text{mapper\_prot } I_T \ I_U \ f \ n \ X}\end{array}\right)\end{array}\right\}$$

$$\texttt{mapper\_worker} \ f_v \ lk \ c$$

$$\{\text{True}\}$$

$$\begin{array}{l}\text{f\_spec } (T \ U : \text{Type}) \ (f_v : \text{Val}) \triangleq \\ \quad \forall x \, v. \{I_T \ x \ v\} \ f_v \ v \ \{\ell. \ell \mapsto_{I_U} f \ x\}\end{array}$$

Ghost Theory:

$$\begin{array}{rl}
\text{True} \Rrightarrow & \exists \gamma. \text{auth}_\gamma \ 0 \ \emptyset \\
\text{auth}_\gamma \ n \ X \Rrightarrow & \text{auth}_\gamma \ (1+n) \ X \ * \text{contrib}_\gamma \ \emptyset \\
\text{auth}_\gamma \ n \ X \ * \text{contrib}_\gamma \ \emptyset \Rrightarrow & \text{auth}_\gamma \ (n-1) \ X \\
\text{auth}_\gamma \ n \ X * \text{contrib}_\gamma \ Y \Rrightarrow & \text{auth}_\gamma \ n \ (X \uplus Z) * \text{contrib}_\gamma \ (Y \uplus Z) \\
Z \subseteq Y * \text{auth}_\gamma \ n \ X * \text{contrib}_\gamma \ Y \Rrightarrow & \text{auth}_\gamma \ n \ (X \setminus Z) * \text{contrib}_\gamma \ (Y \setminus Z) \\
\text{auth}_\gamma \ n \ X * \text{contrib}_\gamma \ Y \twoheadrightarrow & n > 0 * Y \subseteq X \\
\text{auth}_\gamma \ 1 \ X * \text{contrib}_\gamma \ Y \twoheadrightarrow & Y = X
\end{array}$$