Actris: Session-Type Based Reasoning in Separation Logic

Jonas Kastberg Hinrichsen, IT University of Copenhagen

Joint work with
Jesper Bengtson, IT University of Copenhagen
Robbert Krebbers, Delft University of Technology

22 January 2020
POPL 2020, New Orleans

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#

Message passing primitives

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

Example: $\texttt{let}\,(c, c') = \texttt{new\_chan}\ ()\ \texttt{in}$
$\texttt{fork}\ \{\texttt{let}\ x = \texttt{recv}\ c'\ \texttt{in}\ \texttt{send}\ c'\ (x + 2)\};$
$\texttt{send}\ c\ 40;\ \texttt{recv}\ c$

Many variants of message passing exist

We consider: 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]

Problem: No existing solution for dependent high-level actor-based reasoning in combination with existing concurrency models for functional correctness

- **Dependent:** dependency on previously communicated messages
- **High-level:** communication of references, channels and higher-order functions

Key Idea

Protocols akin to **session types** for specifications in **concurrent separation logic**

Session types [Honda et al., ESOP'98]

- Type system for channels
- Example: $!\mathbb{N}.?\mathbb{N}.\mathsf{end}$
- Ensures safety and session fidelity

Concurrent separation logic [O'Hearn & Brooks, CONCUR'04]

- Logic for reasoning about concurrent programs with mutable state
- Example: $\{l \mapsto v\}\ l \leftarrow w\ \{l \mapsto w\}$
- Ensures functional correctness

Contributions

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

- Introducing *dependent separation protocols*
- Integration with Iris and its existing concurrency mechanisms
- Verification of feature-heavy programs including a variant of map-reduce
- Full mechanization in Coq (https://gitlab.mpi-sws.org/iris/actris/)

Features of dependent separation protocols

Specification and proof system for message passing that allows

- **Resources:** sending references
- **Higher-order:** sending function closures
- **Delegation:** sending channels over channels
- **Dependent:** dependency on previous messages
- **Recursion:** looping protocols
- **Branching:** protocols with choice
- **Manifest sharing:** sharing channel endpoints

Features of dependent separation protocols

Specification and proof system for message passing that allows

- **Resources:** sending references
- **Higher-order:** sending function closures
- **Delegation:** sending channels over channels
- **Dependent:** dependency on previous messages
- **Recursion:** looping protocols
- **Branching:** protocols with choice
- **Manifest sharing:** sharing channel endpoints

Tour of Actris

Tour of Actris - Goal

Language: ML-like language extended with concurrency, state and message passing

$$
\begin{aligned}
e \in \mathrm{Expr} ::= {} & \texttt{new\_chan}\ () & \mid \\
& \texttt{send}\ c\ v & \mid \\
& \texttt{recv}\ c & \mid \ldots
\end{aligned}
$$

Example program:

$$
\begin{aligned}
& \texttt{let}\ (c, c') = \texttt{new\_chan}\ ()\ \texttt{in} \\
& \texttt{fork}\ \{\texttt{let}\ x = \texttt{recv}\ c'\ \texttt{in}\ \texttt{send}\ c'\ (x + 2)\}\ ; \\
& \texttt{send}\ c\ 40;\ \texttt{recv}\ c
\end{aligned}
$$

Goal: prove that returned value is 42

Session types

Symbols

$$
\begin{aligned}
st \triangleq\ & !T.st && \mid \\
& ?T.st && \mid \\
& \mathsf{end} && \mid \ldots
\end{aligned}
$$

Example

$$!\mathbb{N}.?\mathbb{N}.\mathsf{end}$$

Duality

$$
\begin{aligned}
\overline{!T.st} &= ?T.\overline{st} \\
\overline{?T.st} &= !T.\overline{st} \\
\overline{\mathsf{end}} &= \mathsf{end}
\end{aligned}
$$

Rules

$$\texttt{new\_chan}\ () : st \otimes \overline{st}$$

$$\texttt{send} : (!T.st \otimes T) \multimap st$$

$$\texttt{recv} : ?T.st \multimap (T \otimes st)$$

Tour of Actris - Type checked

Example program:

$$\begin{aligned}&\texttt{let}\,(c, c') = \texttt{new\_chan}\,()\,\texttt{in}\\&\texttt{fork}\ \{\texttt{let}\,x = \texttt{recv}\ c'\ \texttt{in}\,\texttt{send}\ c'\ (x+2)\}\,;\\&\texttt{send}\ c\ 40;\ \texttt{recv}\ c\end{aligned}$$

Session types:

$$\begin{aligned}c &: \,!\mathbb{N}.?\mathbb{N}.\mathsf{end} \qquad \texttt{and}\\c' &: \,?\mathbb{N}.!\mathbb{N}.\mathsf{end}\end{aligned}$$

Properties obtained:

- ☑ Type safety / session fidelity
- ☒ Functional correctness

Dependent separation protocols - Definitions

	Dependent separation protocols	**Session types**
Symbols	$prot \triangleq \mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot \mid \mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot \mid \mathsf{end}$	$st \triangleq\ !\,T.st \mid ?\,T.st \mid \mathsf{end} \mid \ldots$
Example	$\mathbf{!}\,(x:\mathbb{N})\,\langle x\rangle\{\mathrm{True}\}.\,\mathbf{?}\,(y:\mathbb{N})\,\langle y\rangle\{y = x+2\}.\,\mathsf{end}$	$!\mathbb{N}.?\mathbb{N}.\mathsf{end}$
Duality	$\overline{\mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot} = \mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,\overline{prot}$ $\overline{\mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot} = \mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,\overline{prot}$ $\overline{\mathsf{end}} = \mathsf{end}$	$\overline{!\,T.st} = ?\,T.\overline{st}$ $\overline{?\,T.st} = !\,T.\overline{st}$ $\overline{\mathsf{end}} = \mathsf{end}$
Usage	$c \rightarrowtail prot$	$c : st$

Dependent separation protocols - Rules

	Dependent separation protocols	Session types
New	$\{\mathrm{True}\}$ `new_chan ()` $\{(c, c').\ c \rightarrowtail prot * c' \rightarrowtail \overline{prot}\}$	`new_chan` $() : st \otimes \overline{st}$
Send	$\{c \rightarrowtail \mathbf{!}\, \vec{x}{:}\vec{\tau}\, \langle v\rangle\{P\}.\, prot * P[\vec{t}/\vec{x}]\}$ `send` $c\ (v[\vec{t}/\vec{x}])$ $\{c \rightarrowtail prot[\vec{t}/\vec{x}]\}$	`send` $: (!\, T.st \otimes T) \multimap st$
Recv	$\{c \rightarrowtail \mathbf{?}\, \vec{x}{:}\vec{\tau}\, \langle v\rangle\{P\}.\, prot\}$ `recv` c $\{w.\ \exists\vec{y}.\ (w = v[\vec{y}/\vec{x}]) * P[\vec{y}/\vec{x}] * c \rightarrowtail prot[\vec{y}/\vec{x}]\}$	`recv` $: ?\, T.st \multimap (T \otimes st)$

Tour of Actris - Verified

Example program:

$$\begin{array}{l}\texttt{let}\,(c, c') = \texttt{new\_chan}\ ()\ \texttt{in}\\ \texttt{fork}\ \{\texttt{let}\,x = \texttt{recv}\ c'\ \texttt{in}\,\texttt{send}\ c'\ (x+2)\};\\ \texttt{send}\ c\ 40;\ \texttt{recv}\ c\end{array}$$

Dependent separation protocols:

$$\begin{array}{rl} c \rightarrowtail \mathbf{!}\,(x:\mathbb{N})\,\langle x\rangle\{\mathsf{True}\}.\,\mathbf{?}\,(y:\mathbb{N})\,\langle y\rangle\{y = x+2\}.\,\mathsf{end} & \qquad \texttt{and}\\ c' \rightarrowtail \mathbf{?}\,(x:\mathbb{N})\,\langle x\rangle\{\mathsf{True}\}.\,\mathbf{!}\,(y:\mathbb{N})\,\langle y\rangle\{y = x+2\}.\,\mathsf{end} & \end{array}$$

Properties obtained:

- ☑ Type safety / session fidelity
- ☑ Functional correctness

Example program:

$$\begin{array}{l}\texttt{let}\,(c, c') = \texttt{new\_chan}\;()\;\texttt{in} \\ \texttt{fork}\;\{\texttt{let}\,\ell = \texttt{recv}\;c'\;\texttt{in}\;\ell \leftarrow\,!\,\ell + 2;\;\texttt{send}\;c'\;()\}\,; \\ \texttt{let}\,\ell = \texttt{ref}\;40\;\texttt{in}\;\texttt{send}\;c\;\ell;\;\texttt{recv}\;c;\;\;!\,\ell\end{array}$$

Dependent separation protocols:

$$\begin{array}{ll}c \rightarrowtail \mathbf{!}\,(\ell : \mathsf{Loc})\;(x : \mathbb{N})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{?}\,\langle () \rangle \{\ell \mapsto x + 2\}.\,\mathsf{end} & \quad\texttt{and} \\ c \rightarrowtail \mathbf{?}\,(\ell : \mathsf{Loc})\;(x : \mathbb{N})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,\langle () \rangle \{\ell \mapsto x + 2\}.\,\mathsf{end} & \end{array}$$

Example program:

$$\begin{array}{l}\texttt{let}\,(c, c') = \texttt{new\_chan}\;()\;\texttt{in}\\ \texttt{fork}\;\{\texttt{let}\,\ell = \texttt{recv}\;c'\;\texttt{in}\,\ell \leftarrow\, !\,\ell + 2;\;\texttt{send}\;c'\;()\}\,;\\ \texttt{let}\,\ell = \texttt{ref}\;40\;\texttt{in}\,\texttt{send}\;c\;\ell;\;\texttt{recv}\;c;\;\;!\,\ell\end{array}$$

$$\{\mathsf{True}\}\,\texttt{ref}\;v\,\{\ell.\,\ell \mapsto v\}$$

Dependent separation protocols:

$$c \rightarrowtail \mathbf{!}\,(\ell : \mathsf{Loc})\;(x : \mathbb{N})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{?}\,\langle () \rangle \{\ell \mapsto x + 2\}.\,\mathsf{end} \qquad \texttt{and}$$
$$c \rightarrowtail \mathbf{?}\,(\ell : \mathsf{Loc})\;(x : \mathbb{N})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,\langle () \rangle \{\ell \mapsto x + 2\}.\,\mathsf{end}$$

Example program:

$$\begin{array}{l}\texttt{let}\,(c, c') = \texttt{new\_chan}\;()\;\texttt{in} \\ \texttt{fork}\;\{\texttt{let}\;\ell = \texttt{recv}\;c'\;\texttt{in}\;\ell \leftarrow\,!\,\ell + 2;\;\;\texttt{send}\;c'\;()\}\,; \\ \texttt{let}\;\ell = \texttt{ref}\;40\;\texttt{in}\;\texttt{send}\;c\;\ell;\;\;\texttt{recv}\;c;\;\;!\,\ell\end{array}$$

$$\{\mathsf{True}\}\;\texttt{ref}\;v\;\{\ell.\,\ell \mapsto v\}$$

$$\{\ell \mapsto v\}\;!\,\ell\;\{w.\,w = v \land \ell \mapsto v\}$$

Dependent separation protocols:

$$\begin{array}{ll}c \rightarrowtail \mathbf{!}\,(\ell : \mathsf{Loc})\;(x : \mathbb{N})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{?}\,\langle () \rangle \{\ell \mapsto x + 2\}.\,\mathsf{end} & \quad \texttt{and} \\ c \rightarrowtail \mathbf{?}\,(\ell : \mathsf{Loc})\;(x : \mathbb{N})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,\langle () \rangle \{\ell \mapsto x + 2\}.\,\mathsf{end} & \end{array}$$

Soundness and implementation of Actris

Soundness of Actris

If $\{\text{True}\}\, e\, \{v.\, \phi(v)\}$ is provable in Actris then:

- ☑ **Type safety/session fidelity:** e will not crash and not send wrong messages
- ☑ **Functional correctness:** If e terminates with v, the postcondition $\phi(v)$ holds

Obtained by modeling Actris as an embedded domain-specific logic in **Iris**

- A language-independent higher-order impredicative concurrent separation logic in Coq
- Exactly what we need

Implementation and model of Actris in Iris

Approach:

- Implement `new_chan`, `send`, and `recv` as a library using lock-protected buffers
- Define *prot* using Iris's recursive domain equation solver
- Define $c \rightarrowtail prot$ using Iris's invariant and ghost state machinery
- Prove Actris's proof rules as lemmas in Iris

Benefits:

- ☑ Actris's soundness result is a corollary of Iris's soundness
- ☑ Can readily reuse Iris's support for interactive proofs in Coq
- ☑ Very small Coq mechanization
  (200 lines for channel implementation and proofs, 1000 lines for the definition and proof rules of $c \rightarrowtail prot$, 450 lines for Coq tactics specific for message passing)
- ☑ Readily integrates with other concurrency mechanisms in Iris

Integration with other concurrency mechanisms in Iris

Example program:

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

Dependent separation protocols:

$$\texttt{lock\_prot}\ (n : \mathbb{N}) \triangleq \texttt{if}\ n = 0\ \texttt{then}\ \mathsf{end}\ \texttt{else}\ \mathbf{?}\langle 21\rangle.\texttt{lock\_prot}\ (n-1)$$

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

Proof:

- Main thread: follows immediately from Actris's rules
- Forked-off thread: requires reasoning about locks using Iris

Beyond this talk

Features:

- **Higher-order:** sending function closures
- **Delegation:** sending channels over channels
- **Branching:** protocols with choice

Case Studies:

- Various distributed merge sort variants
- Distributed load-balancing mapper
- A variant of map-reduce

Model:

- Protocols: *prot*
- Ownership: $c \rightarrowtail \mathit{prot}$

In the paper!

Ongoing and future work

- Semantic model of session types via logical relations (Daniël Louwrink)
- Reasoning about deadlock freedom in separation logic (Jules Jacobs)
- Multi-party variant of dependent separation protocols (based on [Honda et al., POPL'08])
- Communication between distributed systems with logical marshalling
- Linearity of channels through Iron [Bizjak et al., POPL'19]
  - Ensure channels are closed

Locks

Example program:

$$\begin{array}{l} \texttt{let}\,(c, c') = \texttt{new\_chan}\,()\ \texttt{in} \\ \texttt{fork} \left\{ \begin{array}{l} \texttt{let}\ lk = \texttt{new\_lock}\,()\ \texttt{in} \\ \texttt{fork}\ \{\texttt{acquire}\ lk; \texttt{send}\ c'\ 21;\ \texttt{release}\ lk\}\,; \\ \texttt{acquire}\ lk; \texttt{send}\ c'\ 21;\ \texttt{release}\ lk \end{array} \right\}; \\ \texttt{recv}\ c + \texttt{recv}\ c \end{array}$$

Dependent separation protocols:

$$\texttt{lock\_prot}\ (n : \mathbb{N}) \triangleq \texttt{if}\ n = 0\ \texttt{then}\ \mathsf{end}\ \texttt{else}\ \mathbf{?}\,\langle 21 \rangle.\texttt{lock\_prot}\ (n-1)$$

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

Hoare Triple for critical section:

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

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

$$\{\mathsf{True}\}$$

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

$$
\begin{array}{rll}
\{\mathsf{True}\} & \texttt{new\_chan}\ () & \{(c_1, c_2).\ (c_1, c_2) \rightarrowtail (\epsilon, \epsilon)\} \\
\{(c_1, c_2) \rightarrowtail (\vec{v}_1, \vec{v}_2)\} & \texttt{send}\ c_1\ w & \{(c_1, c_2) \rightarrowtail (\vec{v}_1 \cdot [w], \vec{v}_2)\} \\
\{(c_1, c_2) \rightarrowtail (\vec{v}_1, \vec{v}_2)\} & \texttt{send}\ c_2\ w & \{(c_1, c_2) \rightarrowtail (\vec{v}_1, \vec{v}_2 \cdot [w])\} \\
\{(c_1, c_2) \rightarrowtail (\vec{v}_1, \vec{v}_2)\} & \texttt{recv}\ c_1 & \{w.\ (\vec{v}_2 = [w] \cdot \vec{w}) * (c_1, c_2) \rightarrowtail (\vec{v}_1, \vec{w})\} \\
\{(c_1, c_2) \rightarrowtail (\vec{v}_1, \vec{v}_2)\} & \texttt{recv}\ c_2 & \{w.\ (\vec{v}_1 = [w] \cdot \vec{w}) * (c_1, c_2) \rightarrowtail (\vec{w}, \vec{v}_2)\}
\end{array}
$$

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 eventually 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

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

Distributed merge sort

Program:

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

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

Dependent separation protocol:

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

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

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

Fine-grained merge sort

Program:

$$
\begin{array}{l}
\texttt{sort\_service}_{fg}\ cmp\ c\ \triangleq\\
\quad \mathtt{branch}\ c\ \mathtt{with}\\
\quad\quad \texttt{right} \Rightarrow \mathtt{select}\ c\ \texttt{right}\\
\quad \mid \texttt{left} \Rightarrow\\
\quad\quad \mathtt{let}\ x_1 = \mathtt{recv}\ c\ \mathtt{in}\\
\quad\quad \mathtt{branch}\ c\ \mathtt{with}\\
\quad\quad\quad \texttt{right} \Rightarrow \mathtt{select}\ c\ \texttt{left};\\
\quad\quad\quad\quad\quad \mathtt{send}\ c\ x_1;\\
\quad\quad\quad\quad\quad \mathtt{select}\ c\ \texttt{right}\\
\quad\quad \mid \texttt{left} \Rightarrow\\
\quad\quad\quad \mathtt{let}\ x_2 = \mathtt{recv}\ c\ \mathtt{in}\\
\quad\quad\quad \mathtt{let}\ c_1 = \mathtt{start}\ \texttt{sort\_service}_{fg}\ cmp\ \mathtt{in}\\
\quad\quad\quad \mathtt{let}\ c_2 = \mathtt{start}\ \texttt{sort\_service}_{fg}\ cmp\ \mathtt{in}\\
\quad\quad\quad \mathtt{select}\ c_1\ \texttt{left}; \mathtt{send}\ c_1\ x_1;\\
\quad\quad\quad \mathtt{select}\ c_2\ \texttt{left}; \mathtt{send}\ c_2\ x_2;\\
\quad\quad\quad \texttt{split}_{fg}\ c\ c_1\ c_2;\ \texttt{merge}_{fg}\ cmp\ c\ c_1\ c_2\\
\quad\quad \mathtt{end}\\
\quad \mathtt{end}
\end{array}
$$

Dependent separation protocol:

$$
\begin{array}{l}
\text{sort\_prot}_{fg}\ (I : T \to \text{Val} \to \text{iProp})\ (R : T \to T \to \mathbb{B}) \triangleq\\
\quad \text{sort\_prot}_{fg}^{\texttt{head}}\ I\ R\ \epsilon
\end{array}
$$

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

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

Distributed mapper

Program:

$$
\begin{array}{ll}
\texttt{mapper\_worker}\ f_v\ lk\ c \triangleq & \\
\quad \texttt{acquire}\ lk;\ \texttt{select}\ c\ \texttt{left}; & \\
\quad \texttt{branch}\ c\ \texttt{with} & \\
\quad\ \ \texttt{right} \Rightarrow \texttt{release}\ lk & \\
\quad \mid \texttt{left}\ \ \Rightarrow \texttt{let}\ x := \texttt{recv}\ c\ \texttt{in}\ \texttt{release}\ lk; & \texttt{(* acquire work *)} \\
\qquad\qquad\quad \texttt{let}\ y := f_v\ x\ \texttt{in} & \texttt{(* map it *)} \\
\qquad\qquad\quad \texttt{acquire}\ lk;\ \texttt{select}\ c\ \texttt{right};\ \texttt{send}\ c\ y;\ \texttt{release}\ lk; & \texttt{(* send it back *)} \\
\qquad\qquad\quad \texttt{mapper\_worker}\ f_v\ lk\ c & \\
\quad \texttt{end} &
\end{array}
$$

Dependent separation protocol:

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

$$
\begin{array}{l}
\mathit{prot}_1\ {}_{\{Q_1\}}\!\oplus_{\{Q_2\}}\ \mathit{prot}_2 \triangleq \\
\quad !\, (b : \mathbb{B})\, \langle b \rangle \{ \texttt{if}\ b\ \texttt{then}\ Q_1\ \texttt{else}\ Q_2 \}. \\
\quad \texttt{if}\ b\ \texttt{then}\ \mathit{prot}_1\ \texttt{else}\ \mathit{prot}_2
\end{array}
$$

$$
\begin{array}{l}
\mathit{prot}_1\ {}_{\{Q_1\}}\!\&_{\{Q_2\}}\ \mathit{prot}_2 \triangleq \\
\quad ?\, (b : \mathbb{B})\, \langle b \rangle \{ \texttt{if}\ b\ \texttt{then}\ Q_1\ \texttt{else}\ Q_2 \}. \\
\quad \texttt{if}\ b\ \texttt{then}\ \mathit{prot}_1\ \texttt{else}\ \mathit{prot}_2
\end{array}
$$

Distributed mapper - Specification

Concurrent Spec of Mapper Worker:

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

$$\texttt{mapper\_worker}\ f_{\mathsf{v}}\ \mathit{lk}\ c$$

$$\{\mathsf{True}\}$$

$$\mathsf{f\_spec}\ (T\ \ U : \mathsf{Type})\ (f_{\mathsf{v}} : \mathsf{Val}) \triangleq \\ \forall x\, v.\, \{I_T\ \ x\ \ v\}\ f_{\mathsf{v}}\ \ v\ \{\ell.\, \ell \mapsto_{I_U} f\ \ x\}$$

Ghost Theory:

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