Actris: Session-Type Based Reasoning in Separation Logic

Jonas Kastberg Hinrichsen, ITU

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

2nd October 2019
Harvard University

The Actor Model and Message Passing

- Principled way of writing concurrent programs
  - Better separation of concurrent behaviour
  - Used in Erlang, Elixir, Go, Java, Scala, F# and C#
- Primitives

  `new_chan` (), `send` c v, `recv` c
- Example:
```
let (c, c') = new_chan () in
fork {send c' 42};
recv c
```
- 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 work where actor-based reasoning is readily available in combination with existing concurrency models for functional verification

Key Idea

Combine

- Session Types [Honda et al., ESOP'98]
  - Type system for channels
  - Example: $!\mathbb{N}.?\mathbb{B}.\textbf{end}$
  - Ensures safety through static type checking
- Concurrent Separation Logic [O'Hearn & Brooks, CONCUR'04]
  - Logic for reasoning about concurrent programs with mutable state.
  - Example: $\{x \mapsto a * y \mapsto b\}\,\texttt{swap}\ x\ y\,\{x \mapsto b * y \mapsto a\}$
  - 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-complete programs including a variant of Map-Reduce
- A full mechanization of all of the above in Coq with tactic support

Demonstration

Syntax

ML-like language with concurrency and mutable state

$$e \in \mathrm{Expr} ::= \texttt{new\_chan}\ () \mid \texttt{send}\ e_1\ e_2 \mid \texttt{recv}\ e \mid \ldots$$

Program

```
let (c, c') = new_chan () in
fork {send c' 42} ;
recv c
```

Goal: Prove that the returned value is 42

Session Types

Definitions

$$st \triangleq \; !T.st \;\mid\; ?T.st \;\mid\; \mathbf{end} \;\mid\; \ldots$$

Example: $!\mathbb{N}.?\mathbb{B}.\mathbf{end}$

Duality:

$$\overline{!T.st} = ?T.\overline{st}$$
$$\overline{?T.st} = !T.\overline{st}$$
$$\overline{\mathbf{end}} = \mathbf{end}$$

Typing: $c : st$

Rules

NewChan

$\texttt{newchan}\;() : st \otimes \overline{st}$

Send

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

Recv

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

Demonstration - Type Checking

Program

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

Session Type

$$c : ?\mathbb{N}.\mathbf{end} \quad \texttt{and}$$
$$c' : !\mathbb{N}.\mathbf{end}$$

Session types do not provide functional correctness

Cannot prove that result is 42

Dependent Separation Protocols - Definitions

Dependent Separation Protocols

$$prot \triangleq \ \mathbf{!}\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot \mid \mathbf{?}\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot \mid \mathbf{end}$$

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

$$\mathbf{!}\,x\,\langle x\rangle\{\mathrm{True}\}.\,\mathbf{?}\,b\,\langle b\rangle\{b = is\_even\ x\}.\,\mathbf{end}$$

$$c \rightarrowtail prot$$

Session Types

$$st \triangleq \ !\,T.st \mid ?\,T.st \mid \mathbf{end} \mid \ldots$$

$$\overline{!\,T.st} = ?\,T.\overline{st}$$
$$\overline{?\,T.st} = !\,T.\overline{st}$$
$$\overline{\mathbf{end}} = \mathbf{end}$$

$$!\mathbb{N}.?\mathbb{B}.\mathbf{end}$$

$$c : st$$

Dependent Separation Protocols - Rules

Ht-NewChan

$\{\mathsf{True}\}$

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

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

NewChan

$\texttt{newchan}\ () : st \otimes \overline{st}$

Ht-Send

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

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

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

Send

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

Ht-Recv

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

$\quad \texttt{recv}\ c$

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

Recv

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

Demonstration - Verified

Logic

$$P, Q, prot ::= \ \mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot \quad | \quad \mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot \quad | \quad \mathbf{end} \quad | \quad c \rightarrowtail st \quad | \ \ldots$$

Program

```
let (c, c') = new_chan () in
fork {send c' 42} ;
recv c
```

Protocol

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

Demonstration - References

Syntax

$$e \in \text{Expr} ::= \ \texttt{ref}\ e \mid !\,\ell \mid \ldots$$

Logic

$$P, Q, \mathit{prot} ::= \ \ell \mapsto v \mid \ldots$$

$$\{\text{True}\}\ \texttt{ref}\ v\ \{\ell.\ell \mapsto v\}$$

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

Program

$$\texttt{let}\ (c, c') = \texttt{new\_chan}\ ()\ \texttt{in}$$
$$\texttt{fork}\ \{\texttt{send}\ c'\ (\texttt{ref}\ 42)\};$$
$$!\,(\texttt{recv}\ c)$$

Protocol

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

Demonstration - Delegation

Delegation: Passing channels over channels

Program

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

Protocols

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

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

Demonstration - Dependency

Program

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

Protocol

$$c \rightarrowtail \mathbf{!}\,x\,\langle x\rangle\{\mathrm{True}\}.\,\mathbf{?}\,\langle x+2\rangle\{\mathrm{True}\}.\,\mathbf{end} \qquad \texttt{and}$$
$$c' \rightarrowtail \mathbf{?}\,x\,\langle x\rangle\{\mathrm{True}\}.\,\mathbf{!}\,\langle x+2\rangle\{\mathrm{True}\}.\,\mathbf{end}$$

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

$$c \rightarrowtail \quad \mathbf{!}\,P\,Q\,f\,\langle f\rangle\{\{P\}\ f\ ()\ \{v.\ Q(v)\}\}.\ \mathbf{?}\,g\,\langle g\rangle\{\{P\}\ g\ ()\ \{v.\ \exists w.\ (v = w + 2) * Q(w)\}\}.\ \mathbf{end}$$

and

$$c \rightarrowtail \quad \mathbf{?}\,P\,Q\,f\,\langle f\rangle\{\{P\}\ f\ ()\ \{v.\ Q(v)\}\}.\ \mathbf{!}\,g\,\langle g\rangle\{\{P\}\ g\ ()\ \{v.\ \exists w.\ (v = w + 2) * Q(w)\}\}.\ \mathbf{end}$$

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

Distributed Merge Sort

```
sort_service cmp c ≜
  let l = recv c in
  if |l| ≤ 1 then send c () else
  let l' = split l in
  let c1 = start (sort_service cmp) in
  let c2 = start (sort_service cmp) in
  send c1 l; send c2 l';
  recv c1; recv c2;
  merge cmp l l'; send c ()
```

```
start e ≜
  let f = e in
  let (c, c') = new_chan () in
  fork {f c'}; c
```

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

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

$$\mathsf{cmp\_spec}\ I\ R\ cmp \triangleq$$
$$\forall x_1\ x_2\ v_1\ v_2.\ \{I\ x_1\ v_1 * I\ x_2\ v_2\}$$
$$cmp\ v_1\ v_2$$
$$\{r.\ r = R\ x_1\ x_2 * I\ x_1\ v_1 * I\ x_2\ v_2\}$$

Choice in Session Types

Definitions

$$e \in \text{Expr} ::= \texttt{select}\ e_1\ e_2 \mid \texttt{branch}\ e_1\ e_2\ e_3 \mid \ldots$$

$$st \triangleq st \oplus st \mid st \mathbin{\&} st \mid \ldots$$

Example: $\textbf{end} \oplus (!\mathbb{N}.\textbf{end} \mathbin{\&} \textbf{end})$

Duality: $\overline{st \oplus st} = \overline{st} \mathbin{\&} \overline{st}$

$\overline{st \mathbin{\&} st} = \overline{st} \oplus \overline{st}$

NB: Conventional Session Type have n-ary branching

Rules

Select

$\texttt{select} : (st_1 \oplus st_2) \multimap st_i \ \textit{with}\ i \in \{1, 2\}$

Branch

$\texttt{branch} : (st_1 \mathbin{\&} st_2) \otimes (st_1 \multimap T) \times (st_2 \multimap T) \multimap T$

Choice in Dependent Separation Protocols - Definitions

Dependent Separation Protocols

$prot_1 \ {}_{\{Q_1\}}{\oplus}_{\{Q_2\}} \ prot_2$
$prot_1 \ {}_{\{Q_1\}}{\&}_{\{Q_2\}} \ prot_2$

$\overline{prot_1 \ {}_{\{Q_1\}}{\oplus}_{\{Q_2\}} \ prot_2} = \overline{prot_1} \ {}_{\{Q_1\}}{\&}_{\{Q_2\}} \ \overline{prot_2}$
$\overline{prot_1 \ {}_{\{Q_1\}}{\&}_{\{Q_2\}} \ prot_2} = \overline{prot_1} \ {}_{\{Q_1\}}{\oplus}_{\{Q_2\}} \ \overline{prot_2}$

$\mathbf{end} \ \oplus \ (\mathbf{!} \ v \ \langle v \rangle \{v > 5\}. \ \mathbf{end} \ \ \& \ \ \mathbf{end})$

Session Types

$st \oplus st$
$st \ \& \ st$

$\overline{st \oplus st} = \overline{st} \ \& \ \overline{st}$
$\overline{st \ \& \ st} = \overline{st} \oplus \overline{st}$

$\mathbf{end} \oplus (!\mathbb{N}.\mathbf{end} \ \& \ \mathbf{end})$

Choice as derivations

Defined as encodings of send and receive

$$\texttt{select}\ e\ e' \triangleq \texttt{send}\ e\ e'$$

$$\texttt{branch}\ e\ \texttt{with}\ \texttt{left} \Rightarrow e_1 \mid \texttt{right} \Rightarrow e_2\ \texttt{end} \triangleq \texttt{if}\ \texttt{recv}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2$$

$$\texttt{left} \triangleq \texttt{true}$$

$$\texttt{right} \triangleq \texttt{false}$$

$$prot_1\ {}_{\{Q_1\}}{\oplus}_{\{Q_2\}}\ prot_2 \triangleq \mathbf{!}\,(b:\mathbb{B})\,\langle b\rangle\{\texttt{if}\ b\ \texttt{then}\ Q_1\ \texttt{else}\ Q_2\}.\ \texttt{if}\ b\ \texttt{then}\ prot_1\ \texttt{else}\ prot_2$$

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

Possible due to dependent behaviour of protocols

Choice in Dependent Separation Protocols - Rules

SELECT

$\texttt{select} : (st_1 \oplus st_2) \multimap st_i \ \textit{with}\ i \ \in \{1, 2\}$

HT-SELECT

$\{c \rightarrowtail prot_1 \ {}_{\{Q_1\}}{\oplus}_{\{Q_2\}} \ prot_2 * \texttt{if}\ b\ \texttt{then}\ Q_1\ \texttt{else}\ Q_2\}\ \texttt{select}\ c\ b\ \{c \rightarrowtail \texttt{if}\ b\ \texttt{then}\ prot_1\ \texttt{else}\ prot_2\}$

BRANCH

$\texttt{branch} : (st_1 \,\&\, st_2) \otimes (st_1 \multimap T) \times (st_2 \multimap T) \multimap T$

HT-BRANCH

$$\frac{\{P * Q_1 * c \rightarrowtail prot_1\}\ e_1\ \{v.\, R\} \qquad \{P * Q_2 * c \rightarrowtail prot_2\}\ e_2\ \{v.\, R\}}{\{P * c \rightarrowtail prot_1 \ {}_{\{Q_1\}}{\&}_{\{Q_2\}} \ prot_2\}\ \texttt{branch}\ c\ \texttt{with}\ \texttt{left} \Rightarrow e_1 \mid \texttt{right} \ \Rightarrow e_2\ \texttt{end}\ \{v.\, R\}}$$

Recursion

Dependent Separation Protocols

$\mu X.prot$

$\overline{\mu X.prot} = \mu X.\overline{prot}$
$\mu X.prot = prot[\mu X.prot/X]$

$\mu X.\, \lambda n, (\mathbf{!}\, m\, \langle m \rangle \{n < m\}.\, X\ \ m) \oplus \mathbf{end}$

Session Types

$\mu X.st$

$\overline{\mu X.st} = \mu X.\overline{st}$
$\mu X.st = st[\mu X.st/X]$

$\mu X.\, (!\mathbb{N}.X) \oplus \mathbf{end}$

Derived entirely from the logic, as channels and protocols are first-class citizens

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
```

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

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

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

Integration with other Concurrency Mechanisms

- Protocols and their ownership are first-class citizens of the logic
- Integration with existing concurrency mechanisms of the logic is inherent

$$
\begin{aligned}
&\texttt{let}\ c = \texttt{start}\ (\lambda\, c.\ \texttt{let}\ lk = \texttt{new\_lock}\ ()\ \texttt{in}\\
&\qquad\qquad\qquad\quad \texttt{fork}\ \{\texttt{acquire}\ lk;\texttt{send}\ c\ 21;\ \texttt{release}\ lk\};\\
&\qquad\qquad\qquad\quad \texttt{acquire}\ lk;\texttt{send}\ c\ 21;\ \texttt{release}\ lk)\ \texttt{in}\\
&\texttt{recv}\ c + \texttt{recv}\ c
\end{aligned}
$$

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

The Model of Actris

- Iris has all the necessary features [Jung et al., JFP'18]
  - Concurrency, Higher-Order, Step-Indexing, Recursion, Ghost State, ...
- Channels encoded as a mutable shared pair of buffers

$$\{\mathsf{True}\}\quad \texttt{new\_chan}\ ()\quad \{(c_1, c_2).\, (c_1, c_2) \rightarrowtail (\epsilon, \epsilon)\}$$

$$\{(c_1, c_2) \rightarrowtail (\vec{v}_1, \vec{v}_2)\}\quad \texttt{send}\ c_1\ w\quad \{(c_1, c_2) \rightarrowtail (w \cdot [\vec{v}_1], \vec{v}_2)\}$$

$$\{(c_1, c_2) \rightarrowtail (\vec{v}_1, \vec{v}_2)\}\quad \texttt{recv}\ c_1\quad \{w.\, (\vec{v}_1 = [w] \cdot \vec{w}) * (c_1, c_2) \rightarrowtail (\vec{w}, \vec{v}_2)\}$$

- Dependent Separation Protocols encoded as continuations

$$\mathrm{iProto} \triangleq 1 + (\mathbb{B} * (\mathsf{Val} \to (\blacktriangleright \mathrm{iProto} \to \mathrm{iProp}) \to \mathrm{iProp}))$$

$$\mathbf{end} \triangleq \texttt{inj}_1\ ()$$

$$\mathbf{!}\, \vec{x}{:}\vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot} \triangleq \texttt{inj}_2\, (\texttt{true}, \lambda\, w\, (f : \blacktriangleright \mathrm{iProto} \to \mathrm{iProp}).\, \exists (\vec{x}{:}\vec{\tau}).\, (v = w) * \triangleright P * f(\mathrm{next}\ \mathit{prot}))$$

$$\mathbf{?}\, \vec{x}{:}\vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot} \triangleq \texttt{inj}_2\, (\texttt{false}, \lambda\, w\, (f : \blacktriangleright \mathrm{iProto} \to \mathrm{iProp}).\, \exists (\vec{x}{:}\vec{\tau}).\, (v = w) * \triangleright P * f(\mathrm{next}\ \mathit{prot}))$$

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

Future Work

- Semantic model of session types via logical relations

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

- Multi-party Session Types [Honda et al., POPL'08]
- Communication between distributed systems with logical marshalling

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-complete programs including a variant of Map-Reduce
- A full mechanization of all of the above in Coq with tactic support
- A paper on Actris has been submitted to POPL'20.