Actris: Session-Type Based Reasoning in Separation Logic

Jonas Kastberg Hinrichsen
IT University of Copenhagen

18 November 2020
SWS Seminar, Radboud University

The actor model and message passing

Principled way of writing concurrent programs

The actor model and message passing

Principled way of writing concurrent programs

- Isolation of concurrent behaviour

The actor model and message passing

Principled way of writing concurrent programs

- Isolation of concurrent behaviour
- Threads as services and clients

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#

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

`new_chan` (), `send` c v, `recv` c

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

```
new_chan (), send c v, recv c
```

Example:

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

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

`new_chan` (), `send` c v, `recv` c

Example:

```
let (c, c') = new_chan () in
fork {let x = recv c' in send c' (x + 2)} ;
send c 40; 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

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

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

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 reasoning in **Iris's concurrent separation logic**

Key Idea

Protocols akin to **session types** for reasoning in **Iris's concurrent separation logic**

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

- Type system for channel endpoints
- Example: $\mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\texttt{end}$
- Ensures safety and session fidelity

Key Idea

Protocols akin to **session types** for reasoning in **Iris's concurrent separation logic**

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

- Type system for channel endpoints
- Example: $\mathbf{!}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\mathtt{end}$
- Ensures safety and session fidelity

Iris's concurrent separation logic [Jung et al., POPL'15]

- Logic for reasoning about concurrent programs with mutable state
- Example: $\{\ell \mapsto v\}\,\ell \leftarrow w\,\{\ell \mapsto w\}$
- Supports high-level concurrency reasoning mechanisms
- 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
- **Choice:** diverging protocols
- **Manifest sharing:** concurrent sharing of channel endpoints
- **Subprotocols:** weakening mechanism for added flexibility

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
- **Choice:** diverging protocols
- **Manifest sharing:** concurrent sharing of channel endpoints
- **Subprotocols:** weakening mechanism for added flexibility

Actris

joint work with

Jesper Bengtson, IT University of Copenhagen
Robbert Krebbers, Radboud University

Tour of Actris - Goal

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

$$\begin{aligned} e \in \mathrm{Expr} ::= {} & v \mid x \mid \texttt{rec}\, f\ x = e \mid e_1(e_2) \mid \texttt{fork}\ \{e\} \mid e_1 \mid\mid e_2 \mid \texttt{ref}\ (e) \mid !\, e \mid e_1 \leftarrow e_2 \mid \\ & \texttt{new\_chan}\ () \mid \texttt{send}\ e_1\ e_2 \mid \texttt{recv}\ e \mid \ldots \end{aligned}$$

Tour of Actris - Goal

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

$$\begin{aligned} e \in \mathrm{Expr} ::= {} & v \mid x \mid \texttt{rec}\, f\ x = e \mid e_1(e_2) \mid \texttt{fork}\ \{e\} \mid e_1 \mid\mid e_2 \mid \texttt{ref}\ (e) \mid !\, e \mid e_1 \leftarrow e_2 \mid \\ & \texttt{new\_chan}\ () \mid \texttt{send}\ e_1\ e_2 \mid \texttt{recv}\ e \mid \ldots \end{aligned}$$

Example program:

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

Tour of Actris - Goal

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

$$e \in \mathrm{Expr} ::= v \mid x \mid \texttt{rec}\ f\ x = e \mid e_1(e_2) \mid \texttt{fork}\ \{e\} \mid e_1 \mid\mid e_2 \mid \texttt{ref}\ (e) \mid !\, e \mid e_1 \leftarrow e_2 \mid$$
$$\texttt{new\_chan}\ () \mid \texttt{send}\ e_1\ e_2 \mid \texttt{recv}\ e \mid \ldots$$

Example program:

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

Goal: prove that returned value is 42

Session types

Symbols

$$S ::= \mathbf{!}A.\,S \mid \mathbf{?}A.\,S \mid \mathtt{end} \mid \ldots$$

Example

$$\mathbf{!}\mathrm{Z}.\,\mathbf{?}\mathrm{Z}.\,\mathtt{end}$$

Duality

$$\begin{aligned} \overline{\mathbf{!}A.\,S} &= \mathbf{?}A.\,\overline{S} \\ \overline{\mathbf{?}A.\,S} &= \mathbf{!}A.\,\overline{S} \\ \overline{\mathtt{end}} &= \mathtt{end} \end{aligned}$$

Usage

$$c : \mathtt{chan}\ S$$

Rules

$$\mathtt{new\_chan}\ () : \mathtt{chan}\ S \times \mathtt{chan}\ \overline{S}$$

$$\mathtt{send} : (\mathtt{chan}\ (\mathbf{!}A.\,S) \times A) \multimap \mathtt{chan}\ S$$

$$\mathtt{recv} : \mathtt{chan}\ (\mathbf{?}A.\,S) \multimap (A \times \mathtt{chan}\ S)$$

Tour of Actris - Type checked

Example program:

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

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:

$$\begin{aligned}c\ &:\ \mathtt{chan}\ (\mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\mathtt{end}) \qquad \mathtt{and}\\ c'\ &:\ \mathtt{chan}\ (\mathbf{?}\mathbb{Z}.\,\mathbf{!}\mathbb{Z}.\,\mathtt{end})\end{aligned}$$

Tour of Actris - Type checked

Example program:

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

Session types:

$$\begin{array}{lcl} c & : & \texttt{chan}\,(\mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\texttt{end}) \qquad \texttt{and}\\ c' & : & \texttt{chan}\,(\mathbf{?}\mathbb{Z}.\,\mathbf{!}\mathbb{Z}.\,\texttt{end})\end{array}$$

Properties obtained:

- ☑ Safety / session fidelity
- ☒ Functional correctness

Dependent separation protocols - Definitions

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

Dependent separation protocols - Rules

	Dependent separation protocols	**Session types**
New	$\{\mathrm{True}\}$ $\texttt{new\_chan}\ ()$ $\{(c, c').\ c \rightarrowtail prot * c' \rightarrowtail \overline{prot}\}$	$\texttt{new\_chan}\ () : \texttt{chan}\ S \times \texttt{chan}\ \overline{S}$
Send	$\{c \rightarrowtail \mathbf{!}\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\ prot * P[\vec{t}/\vec{x}]\}$ $\texttt{send}\ c\ (v[\vec{t}/\vec{x}])$ $\{c \rightarrowtail prot[\vec{t}/\vec{x}]\}$	$\texttt{send} : (\texttt{chan}\ (\mathbf{!}A.\ S) \times A) \multimap \texttt{chan}\ S$
Recv	$\{c \rightarrowtail \mathbf{?}\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\ prot\}$ $\texttt{recv}\ c$ $\{w.\ \exists \vec{y}.\ (w = v[\vec{y}/\vec{x}]) *$ $c \rightarrowtail prot[\vec{y}/\vec{x}] * P[\vec{y}/\vec{x}]\}$	$\texttt{recv} :\ \texttt{chan}\ (\mathbf{?}A.\ S) \multimap (A \times \texttt{chan}\ S)$

Tour of Actris - Verified

Example program:

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

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:

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

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:

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

Properties obtained:

- ✓ Safety / session fidelity
- ✓ Functional correctness

Tour of Actris - References

Example program:

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

Tour of Actris - References

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:

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

Tour of Actris - References

Example program:

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

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

Dependent separation protocols:

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

Tour of Actris - References

Example program:

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

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

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

Dependent separation protocols:

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

Example - Locks

Example program:

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

Example - 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:

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

Example - 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:

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

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

Example - 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:

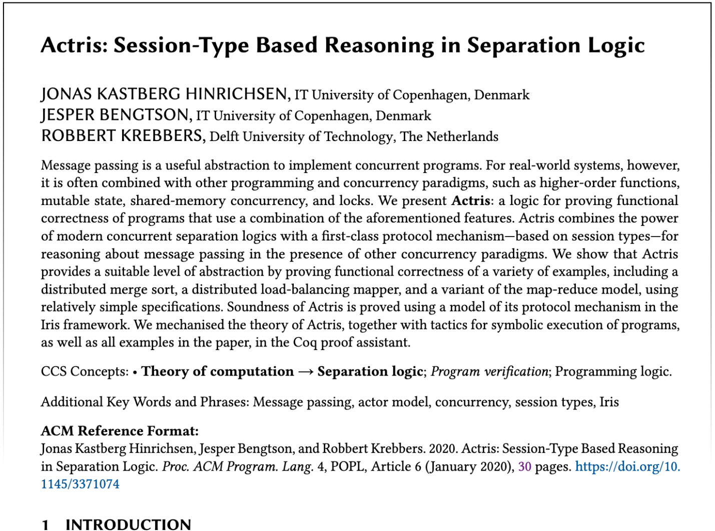
$$\texttt{lock\_prot}\ (n : \mathbb{N}) \triangleq \texttt{if}\ n = 0\ \texttt{then}\ \mathrm{end}\ \texttt{else}\ \mathbf{?}\langle 21\rangle.\,\texttt{lock\_prot}\ (n-1)$$

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

Proof:

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

Paper [POPL'20]: https://itu.dk/people/jkas/papers/actris_popl.pdf
Mechanisation: https://gitlab.mpi-sws.org/iris/actris/-/tree/popl20



Actris: Session-Type Based Reasoning in Separation Logic

JONAS KASTBERG HINRICHSEN, IT University of Copenhagen, Denmark
JESPER BENGTSON, IT University of Copenhagen, Denmark
ROBBERT KREBBERS, Delft University of Technology, The Netherlands

Message passing is a useful abstraction to implement concurrent programs. For real-world systems, however, it is often combined with other programming and concurrency paradigms, such as higher-order functions, mutable state, shared-memory concurrency, and locks. We present **Actris**: a logic for proving functional correctness of programs that use a combination of the aforementioned features. Actris combines the power of modern concurrent separation logics with a first-class protocol mechanism—based on session types—for reasoning about message passing in the presence of other concurrency paradigms. We show that Actris provides a suitable level of abstraction by proving functional correctness of a variety of examples, including a distributed merge sort, a distributed load-balancing mapper, and a variant of the map-reduce model, using relatively simple specifications. Soundness of Actris is proved using a model of its protocol mechanism in the Iris framework. We mechanised the theory of Actris, together with tactics for symbolic execution of programs, as well as all examples in the paper, in the Coq proof assistant.

CCS Concepts: • **Theory of computation → Separation logic**; *Program verification*; Programming logic.

Additional Key Words and Phrases: Message passing, actor model, concurrency, session types, Iris

ACM Reference Format:
Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. 2020. Actris: Session-Type Based Reasoning in Separation Logic. *Proc. ACM Program. Lang.* 4, POPL, Article 6 (January 2020), 30 pages. https://doi.org/10.1145/3371074

1 INTRODUCTION

Actris 2.0

joint work with

Jesper Bengtson, IT University of Copenhagen
Robbert Krebbers, Radboud University

Problem 1 - Lack of expressivity

Actris 1.0 does not take advantage of the asynchronous semantics

Problem 1 - Lack of expressivity

Actris 1.0 does not take advantage of the asynchronous semantics

- Bi-directional buffers allows messages in transit in both directions

Problem 1 - Lack of expressivity

Actris 1.0 does not take advantage of the asynchronous semantics

- Bi-directional buffers allows messages in transit in both directions

Example Program

```
let (c, c') = new_chan () in
fork {send c' 20; let x = recv c' in send c' (x + 2)} ;
send c 20;
let x = recv c in
let y = recv c in x + y
```

Problem 1 - Lack of expressivity

Actris 1.0 does not take advantage of the asynchronous semantics

- Bi-directional buffers allows messages in transit in both directions

Example Program

```
let (c, c') = new_chan () in
fork {send c' 20;  let x = recv c' in send c' (x + 2)} ;
send c 20;
let x = recv c in
let y = recv c in x + y
```

Dependent separation protocols needed for verification

$$c \rightarrowtail \mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,\mathbf{?}\langle 20\rangle\{\mathrm{True}\}.\,\mathbf{?}\langle x+2\rangle\{\mathrm{True}\}.\,\mathrm{end} \qquad \texttt{and}$$
$$c' \rightarrowtail \mathbf{!}\,\langle 20\rangle\{\mathrm{True}\}.\,\mathbf{?}(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,\mathbf{!}\,\langle x+2\rangle\{\mathrm{True}\}.\,\mathrm{end}$$

Problem 1 - Lack of expressivity

Actris 1.0 does not take advantage of the asynchronous semantics

- Bi-directional buffers allows messages in transit in both directions

Example Program

```
let (c, c') = new_chan () in
fork {send c' 20;  let x = recv c' in send c' (x + 2)} ;
send c 20;
let x = recv c in
let y = recv c in x + y
```

Dependent separation protocols needed for verification

$$c \rightarrowtail \mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,\mathbf{?}\langle 20\rangle\{\mathrm{True}\}.\,\mathbf{?}\langle x+2\rangle\{\mathrm{True}\}.\,\mathrm{end} \qquad \texttt{and}$$
$$c' \rightarrowtail \mathbf{!}\,\langle 20\rangle\{\mathrm{True}\}.\,\mathbf{?}(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,\mathbf{!}\,\langle x+2\rangle\{\mathrm{True}\}.\,\mathrm{end}$$

Actris 1.0 requires protocols to be strictly dual

- Every send matched by a receive and vice versa

Problem 2 - Lack of extensionality

Problem 2 - Lack of extensionality

1. **Protocols that differ only syntactically cannot interact**

Problem 2 - Lack of extensionality

1. **Protocols that differ only syntactically cannot interact**

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Problem 2 - Lack of extensionality

1. **Protocols that differ only syntactically cannot interact**

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

2. **Protocols cannot send more or receive less**

Problem 2 - Lack of extensionality

1. **Protocols that differ only syntactically cannot interact**

$$\mathbf{!}\,(x : \mathbb{Z})(y : \mathbb{Z})\,\langle (x, y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle (y, x)\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(y : \mathbb{Z})(x : \mathbb{Z})\,\langle (x, y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle (y, x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

2. **Protocols cannot send more or receive less**

$$\mathbf{!}\,(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle |\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(v : \mathsf{Val})(\vec{w} : \mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle |\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Problem 2 - Lack of extensionality

1. **Protocols that differ only syntactically cannot interact**

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

2. **Protocols cannot send more or receive less**

$$\mathbf{!}\,(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(v:\mathsf{Val})(\vec{w}:\mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

3. **Protocols cannot send and recover a “frame”**

Problem 2 - Lack of extensionality

1. **Protocols that differ only syntactically cannot interact**

$$\begin{array}{l} \mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and} \\ \mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end} \end{array}$$

2. **Protocols cannot send more or receive less**

$$\begin{array}{l} \mathbf{!}\,(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and} \\ \mathbf{?}(v:\mathsf{Val})(\vec{w}:\mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end} \end{array}$$

3. **Protocols cannot send and recover a "frame"** $\dfrac{\{P\}\,e\,\{Q\}}{\{P * R\}\,e\,\{Q * R\}}$

Problem 2 - Lack of extensionality

1. **Protocols that differ only syntactically cannot interact**

$$\mathbf{!}\,(x : \mathbb{Z})(y : \mathbb{Z})\,\langle (x, y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle (y, x)\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(y : \mathbb{Z})(x : \mathbb{Z})\,\langle (x, y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle (y, x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

2. **Protocols cannot send more or receive less**

$$\mathbf{!}\,(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle |\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(v : \mathsf{Val})(\vec{w} : \mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle |\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

3. **Protocols cannot send and recover a "frame"** $\dfrac{\{P\}\,e\,\{Q\}}{\{P * R\}\,e\,\{Q * R\}}$

$$\mathbf{!}\,(\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle \ell\rangle\{\texttt{is\_int\_llist}\ \ell\ \vec{x}\}.\,\mathbf{?}\langle ()\rangle\{\texttt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(\ell : \mathsf{Loc})(\vec{w} : \mathsf{List}\ \mathsf{Val})\,\langle \ell\rangle\{\texttt{is\_llist}\ \ell\ \vec{w}\}.\,\mathbf{!}\,\langle ()\rangle\{\texttt{is\_llist}\ \ell\ (\mathit{rev}\ \vec{w})\}.\,\mathsf{end}$$

Key Idea

Integrate **asynchronous session subtyping** with **Actris**

Key Idea

Integrate **asynchronous session subtyping** with **Actris**

Asynchronous session subtyping [Mostrous et al., Inf.Comput'2015]

- Swapping: $\mathbf{?}A.\,\mathbf{!}B.\,S <: \mathbf{!}B.\,\mathbf{?}A.\,S$

  Example: $\mathbf{?}\mathit{Cat}.\,\mathbf{!}\mathit{Dog}.\,\texttt{end} <: \mathbf{!}\mathit{Dog}.\,\mathbf{?}\mathit{Cat}.\,\texttt{end}$

Key Idea

Integrate **asynchronous session subtyping** with **Actris**

Asynchronous session subtyping [Mostrous et al., Inf.Comput'2015]

- Swapping: $\mathbf{?}A.\,\mathbf{!}B.\,S <: \mathbf{!}B.\,\mathbf{?}A.\,S$

  Example: $\mathbf{?}\mathit{Cat}.\,\mathbf{!}\mathit{Dog}.\,\mathtt{end} <: \mathbf{!}\mathit{Dog}.\,\mathbf{?}\mathit{Cat}.\,\mathtt{end}$

- Contra and covariance of send / receive: $\dfrac{B <: A \qquad S <: T}{\mathbf{!}A.\,S <: \mathbf{!}B.\,S} \quad \dfrac{A <: B \qquad S <: T}{\mathbf{?}A.\,S <: \mathbf{?}B.\,S}$

  Example: $\mathbf{!}\mathit{Cat}.\,\mathbf{?}\mathit{Cat}.\,\mathtt{end} <: \mathbf{!}\mathit{MaineCoon}.\,\mathbf{?}\mathit{Animal}.\,\mathtt{end}$

Key Idea

Integrate **asynchronous session subtyping** with **Actris**

Asynchronous session subtyping [Mostrous et al., Inf.Comput'2015]

- Swapping: $\mathbf{?}A.\,\mathbf{!}B.\,S <: \mathbf{!}B.\,\mathbf{?}A.\,S$

  Example: $\mathbf{?}\mathit{Cat}.\,\mathbf{!}\mathit{Dog}.\,\texttt{end} <: \mathbf{!}\mathit{Dog}.\,\mathbf{?}\mathit{Cat}.\,\texttt{end}$

- Contra and covariance of send / receive: $\dfrac{B <: A \qquad S <: T}{\mathbf{!}A.\,S <: \mathbf{!}B.\,S}$ $\quad \dfrac{A <: B \qquad S <: T}{\mathbf{?}A.\,S <: \mathbf{?}B.\,S}$

  Example: $\mathbf{!}\mathit{Cat}.\,\mathbf{?}\mathit{Cat}.\,\texttt{end} <: \mathbf{!}\mathit{MaineCoon}.\,\mathbf{?}\mathit{Animal}.\,\texttt{end}$

- Subsumption: $\dfrac{A <: B \qquad \Gamma \vdash e : B}{\Gamma \vdash e : A}$ $\quad \dfrac{S <: T}{\texttt{chan}\ S <: \texttt{chan}\ T}$

Problem 1 - Type Checked

Example program

$$\begin{array}{l}
\texttt{let}\,(c, c') = \texttt{new\_chan}\;()\;\texttt{in} \\
\texttt{fork}\;\{\texttt{send}\;c'\;20;\;\texttt{let}\,x = \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\,; \\
\texttt{send}\;c\;20; \\
\texttt{let}\,x = \texttt{recv}\;c\;\texttt{in} \\
\texttt{let}\,y = \texttt{recv}\;c\;\texttt{in}\,x + y
\end{array}$$

Problem 1 - Type Checked

Example program

$$\begin{array}{l}\texttt{let}\,(c, c') = \texttt{new\_chan}\,()\,\texttt{in} \\ \texttt{fork}\ \{\texttt{send}\ c'\ 20;\ \texttt{let}\,x = \texttt{recv}\ c'\,\texttt{in}\,\texttt{send}\ c'\ (x+2)\}\,; \\ \texttt{send}\ c\ 20; \\ \texttt{let}\,x = \texttt{recv}\ c\ \texttt{in} \\ \texttt{let}\,y = \texttt{recv}\ c\ \texttt{in}\,x + y\end{array}$$

Expected session types

$$\begin{array}{rl} c : & \mathtt{chan}\ (\mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\mathtt{end}) \quad \mathtt{and} \\ c' : & \mathtt{chan}\ (\mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\mathbf{!}\mathbb{Z}.\,\mathtt{end}) \end{array}$$

Problem 1 - Type Checked

Example program

$$
\begin{aligned}
&\texttt{let}\,(c, c') = \texttt{new\_chan}\;()\;\texttt{in}\\
&\texttt{fork}\;\{\texttt{send}\;c'\;20;\;\texttt{let}\,x = \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\,;\\
&\texttt{send}\;c\;20;\\
&\texttt{let}\,x = \texttt{recv}\;c\;\texttt{in}\\
&\texttt{let}\,y = \texttt{recv}\;c\;\texttt{in}\,x + y
\end{aligned}
$$

Expected session types

$$
\begin{aligned}
c &: \mathtt{chan}\;(\mathbf{!}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\mathtt{end}) \quad \mathtt{and}\\
c' &: \mathtt{chan}\;(\mathbf{!}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\mathbf{!}\mathsf{Z}.\,\mathtt{end})
\end{aligned}
$$

Dual session types

$$
\begin{aligned}
c &: \mathtt{chan}\;(\mathbf{!}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\mathtt{end}) \quad \mathtt{and}\\
c' &: \mathtt{chan}\;(\mathbf{?}\mathsf{Z}.\,\mathbf{!}\mathsf{Z}.\,\mathbf{!}\mathsf{Z}.\,\mathtt{end})
\end{aligned}
$$

Problem 1 - Type Checked

Example program

$$\begin{array}{l}
\texttt{let}\,(c, c') = \texttt{new\_chan}\;()\;\texttt{in} \\
\texttt{fork}\;\{\texttt{send}\;c'\;20;\;\texttt{let}\,x = \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\,; \\
\texttt{send}\;c\;20; \\
\texttt{let}\,x = \texttt{recv}\;c\;\texttt{in} \\
\texttt{let}\,y = \texttt{recv}\;c\;\texttt{in}\,x + y
\end{array}$$

Expected session types

$$\begin{array}{rl}
c : \mathtt{chan}\;(\mathbf{!}\mathrm{Z}.\,\mathbf{?}\mathrm{Z}.\,\mathbf{?}\mathrm{Z}.\,\mathtt{end}) & \text{and} \\
c' : \mathtt{chan}\;(\mathbf{!}\mathrm{Z}.\,\mathbf{?}\mathrm{Z}.\,\mathbf{!}\mathrm{Z}.\,\mathtt{end}) &
\end{array}$$

Dual session types

$$\begin{array}{rl}
c : \mathtt{chan}\;(\mathbf{!}\mathrm{Z}.\,\mathbf{?}\mathrm{Z}.\,\mathbf{?}\mathrm{Z}.\,\mathtt{end}) & \text{and} \\
c' : \mathtt{chan}\;(\mathbf{?}\mathrm{Z}.\,\mathbf{!}\mathrm{Z}.\,\mathbf{!}\mathrm{Z}.\,\mathtt{end}) &
\end{array}$$

Subtype relation of c'

$$\begin{array}{rl}
& \mathbf{?}\mathrm{Z}.\,\mathbf{!}\mathrm{Z}.\,\mathbf{!}\mathrm{Z}.\,\mathtt{end} \\
<: & \mathbf{!}\mathrm{Z}.\,\mathbf{?}\mathrm{Z}.\,\mathbf{!}\mathrm{Z}.\,\mathtt{end}
\end{array}$$

Problem 2.2 - Type Checked

Protocols cannot send more or receive less

- Corresponding session types

$$\mathbf{!}(\texttt{List}\ \mathsf{Z}).\,\mathbf{?}\mathsf{Z}.\,\texttt{end} \quad \texttt{and}$$
$$\mathbf{?}(\texttt{List any}).\,\mathbf{!}\mathsf{Z}.\,\texttt{end}$$

Problem 2.2 - Type Checked

Protocols cannot send more or receive less

- Corresponding session types

$$\mathbf{!}(\texttt{List}\ \mathsf{Z}).\,\mathbf{?}\mathsf{Z}.\,\texttt{end} \quad \texttt{and}$$
$$\mathbf{?}(\texttt{List any}).\,\mathbf{!}\mathsf{Z}.\,\texttt{end}$$

- Instantiate dual session types

$$\mathbf{!}(\texttt{List}\ \mathsf{Z}).\,\mathbf{?}\mathsf{Z}.\,\texttt{end} \quad \texttt{and}$$
$$\mathbf{?}(\texttt{List}\ \mathsf{Z}).\,\mathbf{!}\mathsf{Z}.\,\texttt{end}$$

Problem 2.2 - Type Checked

Protocols cannot send more or receive less

- Corresponding session types

$$\mathbf{!}(\mathtt{List}\ \mathsf{Z}).\,\mathbf{?}\mathsf{Z}.\,\mathtt{end} \quad \mathtt{and}$$
$$\mathbf{?}(\mathtt{List}\ \mathtt{any}).\,\mathbf{!}\mathsf{Z}.\,\mathtt{end}$$

- Instantiate dual session types

$$\mathbf{!}(\mathtt{List}\ \mathsf{Z}).\,\mathbf{?}\mathsf{Z}.\,\mathtt{end} \quad \mathtt{and}$$
$$\mathbf{?}(\mathtt{List}\ \mathsf{Z}).\,\mathbf{!}\mathsf{Z}.\,\mathtt{end}$$

- Show subtyping of service type:

$$\begin{aligned} &\mathbf{?}(\mathtt{List}\ \mathsf{Z}).\,\mathbf{!}\mathsf{Z}.\,\mathtt{end} \\ <:\ &\mathbf{?}(\mathtt{List}\ \mathtt{any}).\,\mathbf{!}\mathsf{Z}.\,\mathtt{end} \end{aligned}$$

Problem 2.2 - Type Checked

Protocols cannot send more or receive less

- Corresponding session types

$$\mathbf{!}(\texttt{List}\ \mathsf{Z}).\,\mathbf{?}\mathsf{Z}.\,\texttt{end} \quad \texttt{and}$$
$$\mathbf{?}(\texttt{List any}).\,\mathbf{!}\mathsf{Z}.\,\texttt{end}$$

- Instantiate dual session types

$$\mathbf{!}(\texttt{List}\ \mathsf{Z}).\,\mathbf{?}\mathsf{Z}.\,\texttt{end} \quad \texttt{and}$$
$$\mathbf{?}(\texttt{List}\ \mathsf{Z}).\,\mathbf{!}\mathsf{Z}.\,\texttt{end}$$

- Show subtyping of service type:

$$\begin{aligned} &\mathbf{?}(\texttt{List}\ \mathsf{Z}).\,\mathbf{!}\mathsf{Z}.\,\texttt{end} \qquad \texttt{List}\ \mathsf{Z} <: \texttt{List any} \\ <:\ &\mathbf{?}(\texttt{List any}).\,\mathbf{!}\mathsf{Z}.\,\texttt{end} \end{aligned}$$

Problem 2.2 - Type Checked

Protocols cannot send more or receive less

- Corresponding session types

$$\mathbf{!}(\texttt{List}\ \mathsf{Z}).\,\mathbf{?}\mathsf{Z}.\,\texttt{end} \quad \texttt{and}$$
$$\mathbf{?}(\texttt{List any}).\,\mathbf{!}\mathsf{Z}.\,\texttt{end}$$

- Instantiate dual session types

$$\mathbf{!}(\texttt{List}\ \mathsf{Z}).\,\mathbf{?}\mathsf{Z}.\,\texttt{end} \quad \texttt{and}$$
$$\mathbf{?}(\texttt{List}\ \mathsf{Z}).\,\mathbf{!}\mathsf{Z}.\,\texttt{end}$$

- Show subtyping of service type:

$$\begin{aligned} &\mathbf{?}(\texttt{List}\ \mathsf{Z}).\,\mathbf{!}\mathsf{Z}.\,\texttt{end} \qquad \texttt{List}\ \mathsf{Z} <: \texttt{List any} \qquad \mathsf{Z} <: \texttt{any} \\ <:\ &\mathbf{?}(\texttt{List any}).\,\mathbf{!}\mathsf{Z}.\,\texttt{end} \end{aligned}$$

Subprotocols

	Subprotocols	**Subtyping**
Swap	$\mathbf{?}\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,\mathbf{!}\,\vec{y}:\vec{\sigma}\,\langle w\rangle\{Q\}.\,\mathit{prot} \sqsubseteq \mathbf{!}\,\vec{y}:\vec{\sigma}\,\langle w\rangle\{Q\}.\,\mathbf{?}\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,\mathit{prot}$	$\mathbf{?}A.\,\mathbf{!}B.\,S <: \mathbf{!}B.\,\mathbf{?}A.\,S$
Send	$\dfrac{\forall \vec{y}:\vec{\sigma}.\, Q \mathrel{-\!\!*} \exists \vec{x}:\vec{\tau}.\, P * (v_1 = v_2) * \triangleright(\mathit{prot}_1 \sqsubseteq \mathit{prot}_2)}{\mathbf{!}\,\vec{x}:\vec{\tau}\,\langle v_1\rangle\{P\}.\,\mathit{prot}_1 \sqsubseteq \mathbf{!}\,\vec{y}:\vec{\sigma}\,\langle v_2\rangle\{Q\}.\,\mathit{prot}_2}$	$\dfrac{B <: A \qquad S <: T}{\mathbf{!}A.\,S <: \mathbf{!}B.\,T}$
Recv	$\dfrac{\forall \vec{x}:\vec{\tau}.\, P \mathrel{-\!\!*} \exists \vec{y}:\vec{\sigma}.\, Q * (v_1 = v_2) * \triangleright(\mathit{prot}_1 \sqsubseteq \mathit{prot}_2)}{\mathbf{?}\vec{x}:\vec{\tau}\,\langle v_1\rangle\{P\}.\,\mathit{prot}_1 \sqsubseteq \mathbf{?}\vec{y}:\vec{\sigma}\,\langle v_2\rangle\{Q\}.\,\mathit{prot}_2}$	$\dfrac{A <: B \qquad S <: T}{\mathbf{?}A.\,S <: \mathbf{?}B.\,T}$
Sub.	$\dfrac{c \rightarrowtail \mathit{prot}_1 \qquad \mathit{prot}_1 \sqsubseteq \mathit{prot}_2}{c \rightarrowtail \mathit{prot}_2}$	$\dfrac{A <: B \qquad \Gamma \vdash e : B}{\Gamma \vdash e : A}$

Problem 1 - Verified

Example program

$$
\begin{array}{l}
\texttt{let}\,(c, c') = \texttt{new\_chan}\ ()\ \texttt{in} \\
\texttt{fork}\ \{\texttt{send}\ c'\ 20;\ \texttt{let}\,x = \texttt{recv}\ c'\ \texttt{in}\,\texttt{send}\ c'\ (x+2)\}\,; \\
\texttt{send}\ c\ 20; \\
\texttt{let}\,x = \texttt{recv}\ c\ \texttt{in} \\
\texttt{let}\,y = \texttt{recv}\ c\ \texttt{in}\,x + y
\end{array}
$$

Problem 1 - Verified

Example program

$$\begin{array}{l}
\texttt{let}\,(c, c') = \texttt{new\_chan}\;()\;\texttt{in} \\
\texttt{fork}\;\{\texttt{send}\;c'\;20;\;\texttt{let}\,x = \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\,; \\
\texttt{send}\;c\;20; \\
\texttt{let}\,x = \texttt{recv}\;c\;\texttt{in} \\
\texttt{let}\,y = \texttt{recv}\;c\;\texttt{in}\,x+y
\end{array}$$

Dual dependent sepration protocols

$$\begin{array}{rl}
c \rightarrowtail & \mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle 20\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle x+2\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and} \\
c' \rightarrowtail & \mathbf{?}(x:\mathbb{Z})\,\langle x\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle 20\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle x+2\rangle\{\mathsf{True}\}.\,\mathsf{end}
\end{array}$$

Problem 1 - Verified

Example program

$$\begin{array}{l}
\texttt{let}\,(c, c') = \texttt{new\_chan}\ ()\ \texttt{in} \\
\texttt{fork}\ \{\texttt{send}\ c'\ 20;\ \texttt{let}\ x = \texttt{recv}\ c'\ \texttt{in}\ \texttt{send}\ c'\ (x+2)\}\,; \\
\texttt{send}\ c\ 20; \\
\texttt{let}\ x = \texttt{recv}\ c\ \texttt{in} \\
\texttt{let}\ y = \texttt{recv}\ c\ \texttt{in}\ x + y
\end{array}$$

Dual dependent sepration protocols

$$\begin{array}{rl}
c \rightarrowtail\ !\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,?\langle 20\rangle\{\mathrm{True}\}.\,?\langle x+2\rangle\{\mathrm{True}\}.\,\mathrm{end} & \texttt{and} \\
c' \rightarrowtail\ ?(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,!\,\langle 20\rangle\{\mathrm{True}\}.\,!\,\langle x+2\rangle\{\mathrm{True}\}.\,\mathrm{end} &
\end{array}$$

Subprotocol relation of c'

$$\begin{array}{rl}
& ?(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,!\,\langle 20\rangle\{\mathrm{True}\}.\,!\,\langle x+2\rangle\{\mathrm{True}\}.\,\mathrm{end} \\
\sqsubseteq & !\,\langle 20\rangle\{\mathrm{True}\}.\,?(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,!\,\langle x+2\rangle\{\mathrm{True}\}.\,\mathrm{end}
\end{array}$$

Problem 2.1 - Verified

Protocols that differ only syntactically can interact

Problem 2.1 - Verified

Protocols that differ only syntactically <u>can</u> interact

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle (x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle (y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle (x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle (y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Problem 2.1 - Verified

Protocols that differ only syntactically can interact

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad \texttt{and}$$
$$\mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Instantiate dual protocols (from client perspective)

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad \texttt{and}$$
$$\mathbf{?}(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Problem 2.1 - Verified

Protocols that differ only syntactically <u>can</u> interact

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Instantiate dual protocols (from client perspective)

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Show subprotocol relation (using recv subprotocol rule)

$$\mathbf{?}(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$
$$\sqsubseteq\ \mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Problem 2.1 - Verified

Protocols that differ only syntactically <u>can</u> interact

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad \texttt{and}$$
$$\mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Instantiate dual protocols (from client perspective)

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad \texttt{and}$$
$$\mathbf{?}(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Show subprotocol relation (using recv subprotocol rule)

$$\begin{aligned}&\mathbf{?}(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}\\ \sqsubseteq\ &\mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}\end{aligned}$$

$$\forall(x:\mathbb{Z})(y:\mathbb{Z}).\ \mathsf{True} \mathrel{-\!\!*}$$

Problem 2.1 - Verified

Protocols that differ only syntactically <u>can</u> interact

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad \texttt{and}$$
$$\mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Instantiate dual protocols (from client perspective)

$$\mathbf{!}\,(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{?}\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad \texttt{and}$$
$$\mathbf{?}(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Show subprotocol relation (using recv subprotocol rule)

$$\begin{aligned}&\mathbf{?}(x:\mathbb{Z})(y:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}\\ \sqsubseteq\;&\mathbf{?}(y:\mathbb{Z})(x:\mathbb{Z})\,\langle(x,y)\rangle\{\mathsf{True}\}.\,\mathbf{!}\,\langle(y,x)\rangle\{\mathsf{True}\}.\,\mathsf{end}\end{aligned}$$

$$\forall(x:\mathbb{Z})(y:\mathbb{Z}).\ \mathsf{True} \mathrel{-\!\!*} \exists.\,(y:\mathbb{Z})(x:\mathbb{Z}).\mathsf{True} * \ldots$$

Problem 2.2 - Verified

Protocols <u>can</u> send more or receive less

Problem 2.2 - Verified

Protocols <u>can</u> send more or receive less

$$\mathbf{!}\,(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(v : \mathsf{Val})(\vec{w} : \mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Problem 2.2 - Verified

Protocols <u>can</u> send more or receive less

$$\mathbf{!}\,(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad \texttt{and}$$
$$\mathbf{?}(v:\mathsf{Val})(\vec{w}:\mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Instantiate dual protocols (from client perspective)

$$\mathbf{!}\,(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad \texttt{and}$$
$$\mathbf{?}(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{!}\,\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Problem 2.2 - Verified

Protocols <u>can</u> send more or receive less

$$\mathbf{!}\,(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(v : \mathsf{Val})(\vec{w} : \mathsf{List\ Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Instantiate dual protocols (from client perspective)

$$\mathbf{!}\,(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{!}\,\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Show subprotocol relation (using recv subprotocol rule)

$$\begin{aligned}&\mathbf{?}(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{!}\,\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\\ \sqsubseteq\ &\mathbf{?}(v : \mathsf{Val})(\vec{w} : \mathsf{List\ Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\end{aligned}$$

Problem 2.2 - Verified

Protocols can send more or receive less

$$\mathbf{!}\,(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(v : \mathsf{Val})(\vec{w} : \mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Instantiate dual protocols (from client perspective)

$$\mathbf{!}\,(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end} \quad \texttt{and}$$
$$\mathbf{?}(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{!}\,\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Show subprotocol relation (using recv subprotocol rule)

$$\begin{aligned}&\mathbf{?}(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{!}\,\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\\ \sqsubseteq\ &\mathbf{?}(v : \mathsf{Val})(\vec{w} : \mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\end{aligned}$$

$$\forall(v : \mathsf{Val})(\vec{x} : \mathsf{List}\ \mathbb{Z}).\ \texttt{is\_int\_list}\ v\ \vec{x} \mathrel{-\!\!*}$$

Problem 2.2 - Verified

Protocols can send more or receive less

$$\mathbf{!}\,(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad\texttt{and}$$
$$\mathbf{?}(v:\mathsf{Val})(\vec{w}:\mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Instantiate dual protocols (from client perspective)

$$\mathbf{!}\,(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad\texttt{and}$$
$$\mathbf{?}(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{!}\,\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Show subprotocol relation (using recv subprotocol rule)

$$\begin{aligned}&\mathbf{?}(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{!}\,\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\\ \sqsubseteq\ &\mathbf{?}(v:\mathsf{Val})(\vec{w}:\mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\end{aligned}$$

$$\forall(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z}).\,\texttt{is\_int\_list}\ v\ \vec{x} \mathrel{-\!\!*} \exists(v:\mathsf{Val})(\vec{w}:\mathsf{List}\ \mathsf{Val}).\,\texttt{is\_list}\ v\ \vec{w} * \ldots$$

Problem 2.2 - Verified

Protocols <u>can</u> send more or receive less

$$\mathbf{!}\,(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad\texttt{and}$$
$$\mathbf{?}(v:\mathsf{Val})(\vec{w}:\mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Instantiate dual protocols (from client perspective)

$$\mathbf{!}\,(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{?}\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\quad\texttt{and}$$
$$\mathbf{?}(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{!}\,\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Show subprotocol relation (using recv subprotocol rule)

$$\begin{aligned}&\mathbf{?}(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z})\,\langle v\rangle\{\texttt{is\_int\_list}\ v\ \vec{x}\}.\,\mathbf{!}\,\langle|\vec{x}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\\ \sqsubseteq\ &\mathbf{?}(v:\mathsf{Val})(\vec{w}:\mathsf{List}\ \mathsf{Val})\,\langle v\rangle\{\texttt{is\_list}\ v\ \vec{w}\}.\,\mathbf{!}\,\langle|\vec{w}|\rangle\{\mathsf{True}\}.\,\mathsf{end}\end{aligned}$$

$$\forall(v:\mathsf{Val})(\vec{x}:\mathsf{List}\ \mathbb{Z}).\,\texttt{is\_int\_list}\ v\ \vec{x} \mathrel{-\!\!*} \exists(v:\mathsf{Val})(\vec{w}:\mathsf{List}\ \mathsf{Val}).\,\texttt{is\_list}\ v\ \vec{w} * \ldots$$

$$\texttt{is\_int\_list}\ v\ \vec{x} \mathrel{*\!\!*} (\exists(\vec{w}:\mathsf{List}\ \mathsf{Val}).\,\texttt{is\_list}\ v\ \vec{w} * \mathop{\Huge *}_{(x,w)\in(\vec{x},\vec{w})}.x = w)$$

Problem 2.3 - Verified

Protocols <u>can</u> send and recover a “frame”

Problem 2.3 - Verified

Protocols <u>can</u> send and recover a “frame”

$\mathbf{!}\,(\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle \ell \rangle \{\texttt{is\_int\_llist}\ \ell\ \vec{x}\}.\,\mathbf{?}\langle () \rangle \{\texttt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\,\mathsf{end}$ `and`
$\mathbf{?}(\ell : \mathsf{Loc})(\vec{w} : \mathsf{List\ Val})\,\langle \ell \rangle \{\texttt{is\_llist}\ \ell\ \vec{w}\}.\,\mathbf{!}\,\langle () \rangle \{\texttt{is\_llist}\ \ell\ (\mathit{rev}\ \vec{w})\}.\,\mathsf{end}$

Problem 2.3 - Verified

Protocols <u>can</u> send and recover a "frame"

$\mathbf{!}\,(\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle \ell \rangle \{\texttt{is\_int\_llist}\ \ell\ \vec{x}\}.\,\mathbf{?}\langle () \rangle \{\texttt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\,\mathsf{end}$ $\texttt{and}$
$\mathbf{?}(\ell : \mathsf{Loc})(\vec{w} : \mathsf{List\ Val})\,\langle \ell \rangle \{\texttt{is\_llist}\ \ell\ \vec{w}\}.\,\mathbf{!}\,\langle () \rangle \{\texttt{is\_llist}\ \ell\ (\mathit{rev}\ \vec{w})\}.\,\mathsf{end}$

Instantiate dual protocols (from client perspective)

$\mathbf{!}\,(\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle \ell \rangle \{\texttt{is\_int\_llist}\ \ell\ \vec{x}\}.\,\mathbf{?}\langle () \rangle \{\texttt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\,\mathsf{end}$ $\texttt{and}$
$\mathbf{?}(\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle \ell \rangle \{\texttt{is\_int\_llist}\ \ell\ \vec{x}\}.\,\mathbf{!}\,\langle () \rangle \{\texttt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\,\mathsf{end}$

Problem 2.3 - Verified

Protocols can send and recover a "frame"

$! (\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\ \langle \ell \rangle \{\mathtt{is\_int\_llist}\ \ell\ \vec{x}\}.\ ?\langle () \rangle \{\mathtt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\ \mathsf{end}$ and
$?(\ell : \mathsf{Loc})(\vec{w} : \mathsf{List\ Val})\ \langle \ell \rangle \{\mathtt{is\_llist}\ \ell\ \vec{w}\}.\ !\ \langle () \rangle \{\mathtt{is\_llist}\ \ell\ (\mathit{rev}\ \vec{w})\}.\ \mathsf{end}$

Instantiate dual protocols (from client perspective)

$! (\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\ \langle \ell \rangle \{\mathtt{is\_int\_llist}\ \ell\ \vec{x}\}.\ ?\langle () \rangle \{\mathtt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\ \mathsf{end}$ and
$?(\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\ \langle \ell \rangle \{\mathtt{is\_int\_llist}\ \ell\ \vec{x}\}.\ !\ \langle () \rangle \{\mathtt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\ \mathsf{end}$

Show subprotocol relation (using recv and send subprotocol rules)

$$\begin{aligned} & ?(\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\ \langle \ell \rangle \{\mathtt{is\_int\_llist}\ \ell\ \vec{x}\}.\ !\ \langle () \rangle \{\mathtt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\ \mathsf{end} \\ \sqsubseteq\ & ?(\ell : \mathsf{Loc})(\vec{v} : \mathsf{List\ Val})\ \langle \ell \rangle \{\mathtt{is\_llist}\ \ell\ \vec{v}\}.\ !\ \langle () \rangle \{\mathtt{is\_llist}\ \ell\ (\mathit{rev}\ \vec{v})\}.\ \mathsf{end} \end{aligned}$$

Problem 2.3 - Verified

Protocols <u>can</u> send and recover a "frame"

$\mathbf{!}\,(\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle \ell \rangle\{\texttt{is\_int\_llist}\ \ell\ \vec{x}\}.\,\mathbf{?}\langle () \rangle\{\texttt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\,\mathsf{end}$ and

$\mathbf{?}(\ell : \mathsf{Loc})(\vec{w} : \mathsf{List\ Val})\,\langle \ell \rangle\{\texttt{is\_llist}\ \ell\ \vec{w}\}.\,\mathbf{!}\,\langle () \rangle\{\texttt{is\_llist}\ \ell\ (\mathit{rev}\ \vec{w})\}.\,\mathsf{end}$

Instantiate dual protocols (from client perspective)

$\mathbf{!}\,(\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle \ell \rangle\{\texttt{is\_int\_llist}\ \ell\ \vec{x}\}.\,\mathbf{?}\langle () \rangle\{\texttt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\,\mathsf{end}$ and

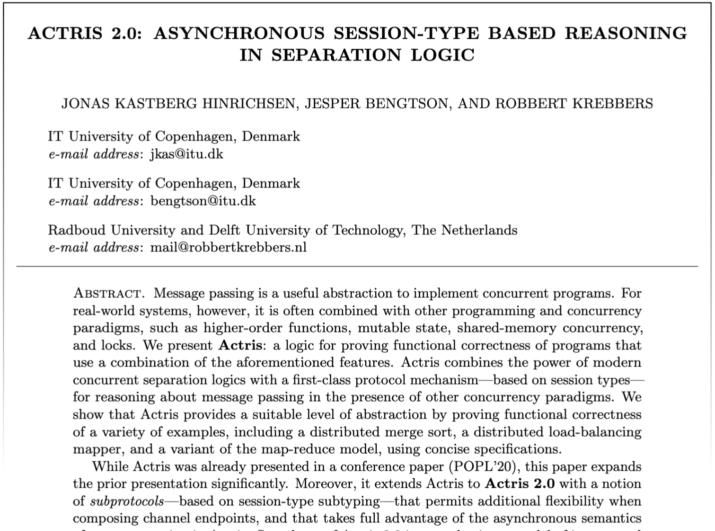
$\mathbf{?}(\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle \ell \rangle\{\texttt{is\_int\_llist}\ \ell\ \vec{x}\}.\,\mathbf{!}\,\langle () \rangle\{\texttt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\,\mathsf{end}$

Show subprotocol relation (using recv and send subprotocol rules)

$$\begin{aligned} &\mathbf{?}(\ell : \mathsf{Loc})(\vec{x} : \mathsf{List}\ \mathbb{Z})\,\langle \ell \rangle\{\texttt{is\_int\_llist}\ \ell\ \vec{x}\}.\,\mathbf{!}\,\langle () \rangle\{\texttt{is\_int\_llist}\ \ell\ (\mathit{rev}\ \vec{x})\}.\,\mathsf{end} \\ \sqsubseteq\ &\mathbf{?}(\ell : \mathsf{Loc})(\vec{v} : \mathsf{List\ Val})\,\langle \ell \rangle\{\texttt{is\_llist}\ \ell\ \vec{v}\}.\,\mathbf{!}\,\langle () \rangle\{\texttt{is\_llist}\ \ell\ (\mathit{rev}\ \vec{v})\}.\,\mathsf{end} \end{aligned}$$

$$\texttt{is\_int\_llist}\ \ell\ \vec{x} \twoheadrightarrow\!\!\twoheadleftarrow (\exists(\vec{v} : \texttt{List Val}).\,\texttt{is\_llist}\ \ell\ \vec{v} * \mathop{\Huge *}_{(x,v)\in(\vec{x},\vec{v})}.x = v)$$

Draft [LMCS]: https://itu.dk/people/jkas/papers/actris_lmcs.pdf
Mechanisation: https://gitlab.mpi-sws.org/iris/actris/-/tree/lmcs



ACTRIS 2.0: ASYNCHRONOUS SESSION-TYPE BASED REASONING IN SEPARATION LOGIC

JONAS KASTBERG HINRICHSEN, JESPER BENGTSON, AND ROBBERT KREBBERS

IT University of Copenhagen, Denmark
e-mail address: jkas@itu.dk

IT University of Copenhagen, Denmark
e-mail address: bengtson@itu.dk

Radboud University and Delft University of Technology, The Netherlands
e-mail address: mail@robbertkrebbers.nl

Abstract. Message passing is a useful abstraction to implement concurrent programs. For real-world systems, however, it is often combined with other programming and concurrency paradigms, such as higher-order functions, mutable state, shared-memory concurrency, and locks. We present **Actris**: a logic for proving functional correctness of programs that use a combination of the aforementioned features. Actris combines the power of modern concurrent separation logics with a first-class protocol mechanism—based on session types—for reasoning about message passing in the presence of other concurrency paradigms. We show that Actris provides a suitable level of abstraction by proving functional correctness of a variety of examples, including a distributed merge sort, a distributed load-balancing mapper, and a variant of the map-reduce model, using concise specifications.

While Actris was already presented in a conference paper (POPL'20), this paper expands the prior presentation significantly. Moreover, it extends Actris to **Actris 2.0** with a notion of *subprotocols*—based on session-type subtyping—that permits additional flexibility when composing channel endpoints, and that takes full advantage of the asynchronous semantics

Semantic Session Typing

joint work with

Daniël Louwrink, Universty of Amsterdam
Jesper Bengtson, IT University of Copenhagen
Robbert Krebbers, Radboud University

Problem

Consider the following program:

$$\lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Problem

Consider the following program:

$$\lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is it safe?

Problem

Consider the following program:

$$\lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is it safe? Yes

Problem

Consider the following program:

$$\lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is it safe? Yes Order of receives does not matter

Problem

Consider the following program:

$$\lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is it safe? Yes Order of receives does not matter
Is it typeable?

Problem

Consider the following program:

$$\lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is it safe?	Yes	Order of receives does not matter
Is it typeable?	No	

Problem

Consider the following program:

$$\lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is it safe?	Yes	Order of receives does not matter
Is it typeable?	No	It violates the ownership discipline

Problem

Consider the following program:

$$\lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\textbf{?}\mathsf{Z}.\, \textbf{?}\mathsf{Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is it safe?	Yes	Order of receives does not matter
Is it typeable?	No	It violates the ownership discipline
Really?		

Problem

Consider the following program:

$$\lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is it safe?	Yes	Order of receives does not matter
Is it typeable?	No	It violates the ownership discipline
Really?	Well...	

Problem

Consider the following program:

$$\lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is it safe?	Yes	Order of receives does not matter
Is it typeable?	No	It violates the ownership discipline
Really?	Well...	It could be added as an ad-hoc rule

Syntactic Typing and its short-comings

Adding ad-hoc typing rules is infeasible

Syntactic Typing and its short-comings

Adding ad-hoc typing rules is infeasible in a **syntactic type system**

Syntactic Typing and its short-comings

Adding ad-hoc typing rules is infeasible in a **syntactic type system**

- **Types** defined as a closed inductive definition

Syntactic Typing and its short-comings

Adding ad-hoc typing rules is infeasible in a **syntactic type system**

- **Types** defined as a closed inductive definition
- **Rules** defined as a closed inductive relation

Syntactic Typing and its short-comings

Adding ad-hoc typing rules is infeasible in a **syntactic type system**

- **Types** defined as a closed inductive definition
- **Rules** defined as a closed inductive relation
- **Soundness** proven as **progress** and **preservation**

Syntactic Typing and its short-comings

Adding ad-hoc typing rules is infeasible in a **syntactic type system**

- **Types** defined as a closed inductive definition
- **Rules** defined as a closed inductive relation
- **Soundness** proven as **progress** and **preservation** using induction over the relation

Syntactic Typing and its short-comings

Adding ad-hoc typing rules is infeasible in a **syntactic type system**

- **Types** defined as a closed inductive definition
- **Rules** defined as a closed inductive relation
- **Soundness** proven as **progress** and **preservation** using induction over the relation

Supporting $\lambda c.\, (\texttt{recv}\ c \mid\mid \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$

Syntactic Typing and its short-comings

Adding ad-hoc typing rules is infeasible in a **syntactic type system**

- **Types** defined as a closed inductive definition
- **Rules** defined as a closed inductive relation
- **Soundness** proven as **progress** and **preservation** using induction over the relation

Supporting $\lambda c.\, (\texttt{recv}\ c \mid\mid \texttt{recv}\ c) : \mathtt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \mathtt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$

- Requires adding ad-hoc rule for it (and all reductions to satisfy **preservation**)

Syntactic Typing and its short-comings

Adding ad-hoc typing rules is infeasible in a **syntactic type system**

- **Types** defined as a closed inductive definition
- **Rules** defined as a closed inductive relation
- **Soundness** proven as **progress** and **preservation** using induction over the relation

Supporting $\lambda c.\,(\texttt{recv}\ c \mid\mid \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\mathsf{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$

- Requires adding ad-hoc rule for it (and all reductions to satisfy **preservation**)
- Must reprove **progress** and **preservation** for any such addition

Syntactic Typing and its short-comings

Adding ad-hoc typing rules is infeasible in a **syntactic type system**

- **Types** defined as a closed inductive definition
- **Rules** defined as a closed inductive relation
- **Soundness** proven as **progress** and **preservation** using induction over the relation

Supporting $\lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\texttt{end}) \multimap (\mathbb{Z} \times \mathbb{Z})$

- Requires adding ad-hoc rule for it (and all reductions to satisfy **preservation**)
- Must reprove **progress** and **preservation** for any such addition
- Resulting proof effort is infeasible

Goal: Type system where ad-hoc rules can be added

Solution: Semantic Type System!

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

- **Types** defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

- **Types** defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- **Judgement** defined as safety-capturing evaluation: $\Gamma \vDash e : A$

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

- **Types** defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- **Judgement** defined as safety-capturing evaluation: $\Gamma \vDash e : A$

  e does not get *stuck*

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

- **Types** defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- **Judgement** defined as safety-capturing evaluation: $\Gamma \vDash e : A$

  e does not get *stuck* `and` if e reduces to a value v, $A\,v$ holds.

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

- **Types** defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- **Judgement** defined as safety-capturing evaluation: $\Gamma \vDash e : A$

  e does not get *stuck* `and` if e reduces to a value v, $A\, v$ holds.
- **Rules** are proven as lemmas

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

- **Types** defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- **Judgement** defined as safety-capturing evaluation: $\Gamma \vDash e : A$

  e does not get *stuck* `and` if e reduces to a value v, $A\,v$ holds.
- **Rules** are proven as lemmas: $\vDash i : \mathsf{Z}$

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

- **Types** defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- **Judgement** defined as safety-capturing evaluation: $\Gamma \vDash e : A$

  e does not get *stuck* `and` if e reduces to a value v, $A\,v$ holds.
- **Rules** are proven as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

- **Types** defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- **Judgement** defined as safety-capturing evaluation: $\Gamma \vDash e : A$

  e does not get *stuck* `and` if e reduces to a value v, $A\,v$ holds.
- **Rules** are proven as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$
- **Soundness** is a consequence of the judgement definition

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

- **Types** defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- **Judgement** defined as safety-capturing evaluation: $\Gamma \vDash e : A$

  e does not get *stuck* and if e reduces to a value v, $A\,v$ holds.
- **Rules** are proven as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$
- **Soundness** is a consequence of the judgement definition

Supporting $\lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

- **Types** defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- **Judgement** defined as safety-capturing evaluation: $\Gamma \vDash e : A$

  e does not get *stuck* and if e reduces to a value v, $A\, v$ holds.
- **Rules** are proven as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$
- **Soundness** is a consequence of the judgement definition

Supporting $\lambda c.\, (\texttt{recv}\ c \mid\mid \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$

- Requires adding ad-hoc rule for it

Semantic Typing

A **semantic type system** is defined in terms of the language semantics:

- **Types** defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- **Judgement** defined as safety-capturing evaluation: $\Gamma \vDash e : A$

  e does not get *stuck* `and` if e reduces to a value v, $A\,v$ holds.
- **Rules** are proven as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$
- **Soundness** is a consequence of the judgement definition

Supporting $\lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$

- Requires adding ad-hoc rule for it
- Requires logical interpretation of session types

Key Idea

Semantic Typing

Semantic Typing [Milner, Princeton Proof-Carrying Code project, RustBelt Project]

- Supports adding ad-hoc rules

Key Idea

Semantic Typing using **Iris**

Semantic Typing [Milner, Princeton Proof-Carrying Code project, RustBelt Project]

- Supports adding ad-hoc rules

Iris [Iris project]

- Semantic type system for similar language (modulo message passing)
  https://gitlab.mpi-sws.org/iris/tutorial-popl20
- Mechanised in **Coq**

Key Idea

Semantic Typing using **Iris** and **Actris**

Semantic Typing [Milner, Princeton Proof-Carrying Code project, RustBelt Project]

- Supports adding ad-hoc rules

Iris [Iris project]

- Semantic type system for similar language (modulo message passing)
  https://gitlab.mpi-sws.org/iris/tutorial-popl20
- Mechanised in **Coq**

Actris [Hinrichsen et al., POPL'20]

- **Dependent separation protocols:** Session type-style logical protocols
- Mechanised in **Coq**

Semantic Session Types

Session types as dependent separation protocols:

$$\begin{aligned} \mathsf{Type}_{\blacklozenge} &\triangleq \mathsf{iProto} \\ \mathbf{!}A.\,S &\triangleq \mathbf{!}\,(v : \mathsf{Val})\,\langle v\rangle\{A\,v\}.\,S \\ \mathbf{?}A.\,S &\triangleq \mathbf{?}(v : \mathsf{Val})\,\langle v\rangle\{A\,v\}.\,S \\ \mathtt{end} &\triangleq \mathsf{end} \end{aligned} \qquad \begin{aligned} \mathsf{Type}_{\bigstar} &\triangleq \mathsf{Val} \to \mathsf{iProp} \\ \mathtt{chan}\ S &\triangleq \lambda w.\, w \rightarrowtail S \end{aligned}$$

Semantic Session Types

Session types as dependent separation protocols:

$$\begin{aligned} \mathsf{Type}_{\blacklozenge} &\triangleq \mathsf{iProto} \\ \mathbf{!}A.\,S &\triangleq \mathbf{!}\,(v : \mathsf{Val})\,\langle v\rangle\{A\,v\}.\,S \\ \mathbf{?}A.\,S &\triangleq \mathbf{?}(v : \mathsf{Val})\,\langle v\rangle\{A\,v\}.\,S \\ \mathsf{end} &\triangleq \mathsf{end} \end{aligned} \qquad \begin{aligned} \mathsf{Type}_{\bigstar} &\triangleq \mathsf{Val} \to \mathsf{iProp} \\ \mathtt{chan}\ S &\triangleq \lambda w.\, w \rightarrowtail S \end{aligned}$$

Rules are proven as lemmas using the rules for dependent separation protocols

$$\begin{aligned} \Gamma &\vDash \texttt{new\_chan}\ () : \mathtt{chan}\ S \times \mathtt{chan}\ \overline{S} \dashv \Gamma \\ \Gamma, (c : \mathtt{chan}\ (\mathbf{!}A.\,S)), (x : A) &\vDash \texttt{send}\ c\ x : 1 \dashv \Gamma, (c : \mathtt{chan}\ S) \\ \Gamma, (c : \mathtt{chan}\ (\mathbf{?}A.\,S)) &\vDash \texttt{recv}\ c : A \dashv \Gamma, (c : \mathtt{chan}\ S) \end{aligned}$$

Typing the Untypeable Program

The rule:

$$\vDash \lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Typing the Untypeable Program

The rule:

$$\vDash \lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is just another lemma

Typing the Untypeable Program

The rule:

$$\vDash \lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is just another lemma proven by unfolding all type-level definitions

$$(c \rightarrowtail \mathbf{?}(v_1 : \mathsf{Val})\, \langle v_1 \rangle \{v_1 \in \mathbb{Z}\}.\, \mathbf{?}(v_2 : \mathsf{Val})\, \langle v_2 \rangle \{v_2 \in \mathbb{Z}\}.\, \mathsf{end}) \mathrel{-\!\!*}$$
$$\mathsf{wp}\ (\texttt{recv}\ c \parallel \texttt{recv}\ c)\ \{v.\, \exists v_1, v_2.\, (v = (v_1, v_2)) * \triangleright(v_1 \in \mathbb{Z}) * \triangleright(v_2 \in \mathbb{Z})\}$$

Typing the Untypeable Program

The rule:

$$\vDash \lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\textbf{?}\mathsf{Z}.\, \textbf{?}\mathsf{Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is just another lemma proven by unfolding all type-level definitions

$$(c \rightarrowtail \textbf{?}(v_1 : \mathsf{Val})\, \langle v_1 \rangle \{v_1 \in \mathbb{Z}\}.\, \textbf{?}(v_2 : \mathsf{Val})\, \langle v_2 \rangle \{v_2 \in \mathbb{Z}\}.\, \mathsf{end}) \mathrel{-\!\!*}$$
$$\mathsf{wp}\, (\texttt{recv}\ c \parallel \texttt{recv}\ c)\, \{v.\, \exists v_1, v_2.\, (v = (v_1, v_2)) * \triangleright(v_1 \in \mathbb{Z}) * \triangleright(v_2 \in \mathbb{Z})\}$$

And then using Iris's ghost state machinery!

Typing the Untypeable Program

The rule:

$$\vDash \lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Is just another lemma proven by unfolding all type-level definitions

$$(c \rightarrowtail \mathbf{?}(v_1 : \mathsf{Val})\,\langle v_1\rangle\{v_1 \in \mathbb{Z}\}.\,\mathbf{?}(v_2 : \mathsf{Val})\,\langle v_2\rangle\{v_2 \in \mathbb{Z}\}.\,\mathsf{end}) \mathrel{-\!\!*}$$
$$\mathsf{wp}\,(\texttt{recv}\ c \parallel \texttt{recv}\ c)\,\{v.\,\exists v_1, v_2.\,(v = (v_1, v_2)) * \triangleright(v_1 \in \mathbb{Z}) * \triangleright(v_2 \in \mathbb{Z})\}$$

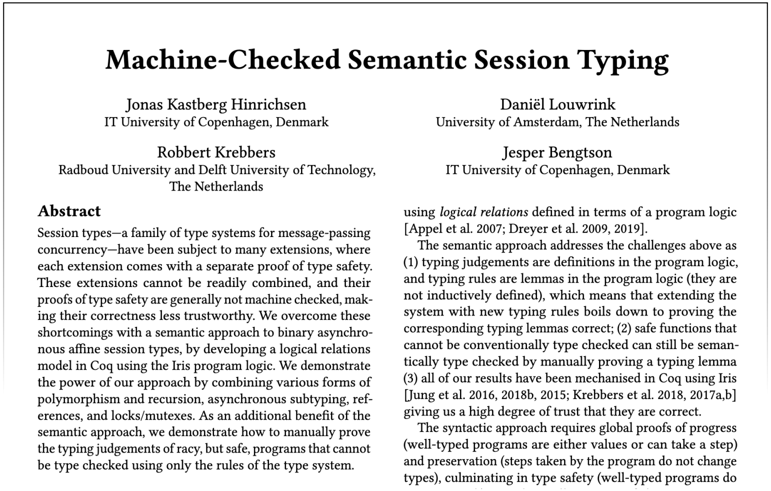
And then using Iris's ghost state machinery! Beyond the scope of this talk

Full Contributions

Semantic Session Type System

- Formal connection between dependent separation protocols and session types
- Rich extensible type system for session types
    - Term and session type equi-recursion
    - Term and session type polymorphism
    - Term and (asynchronous) session type subtyping
    - Unique and shared reference types, Copyable types, Lock types
- Full mechanisation in Coq
- Supports integration of safe yet untypeable programs

Draft [CPP'21]: https://itu.dk/people/jkas/papers/semantic_session_typing_cpp.pdf
Mechanisation: https://gitlab.mpi-sws.org/iris/actris/-/tree/cpp21



Machine-Checked Semantic Session Typing

Jonas Kastberg Hinrichsen
IT University of Copenhagen, Denmark

Daniël Louwrink
University of Amsterdam, The Netherlands

Robbert Krebbers
Radboud University and Delft University of Technology, The Netherlands

Jesper Bengtson
IT University of Copenhagen, Denmark

Abstract

Session types—a family of type systems for message-passing concurrency—have been subject to many extensions, where each extension comes with a separate proof of type safety. These extensions cannot be readily combined, and their proofs of type safety are generally not machine checked, making their correctness less trustworthy. We overcome these shortcomings with a semantic approach to binary asynchronous affine session types, by developing a logical relations model in Coq using the Iris program logic. We demonstrate the power of our approach by combining various forms of polymorphism and recursion, asynchronous subtyping, references, and locks/mutexes. As an additional benefit of the semantic approach, we demonstrate how to manually prove the typing judgements of racy, but safe, programs that cannot be type checked using only the rules of the type system.

using *logical relations* defined in terms of a program logic [Appel et al. 2007; Dreyer et al. 2009, 2019].

The semantic approach addresses the challenges above as (1) typing judgements are definitions in the program logic, and typing rules are lemmas in the program logic (they are not inductively defined), which means that extending the system with new typing rules boils down to proving the corresponding typing lemmas correct; (2) safe functions that cannot be conventionally type checked can still be semantically type checked by manually proving a typing lemma (3) all of our results have been mechanised in Coq using Iris [Jung et al. 2016, 2018b, 2015; Krebbers et al. 2018, 2017a,b] giving us a high degree of trust that they are correct.

The syntactic approach requires global proofs of progress (well-typed programs are either values or can take a step) and preservation (steps taken by the program do not change types), culminating in type safety (well-typed programs do

Soundness and implementation of Actris

Soundness of Actris

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

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

Implementation and model of Actris in Iris

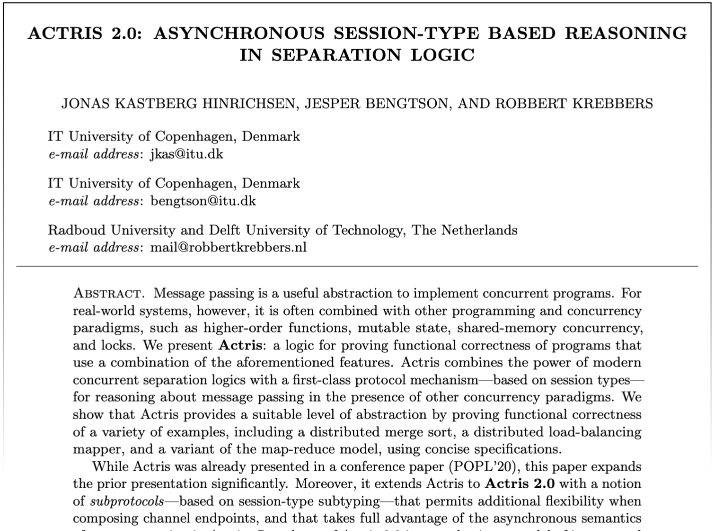
Approach:

- Define the type of *prot* with support from Iris's recursive domain equation solver
- Define operations and relations on *prot*, such as $\overline{prot}$ and $prot_1 \sqsubseteq prot_2$
- Implement `new_chan`, `send`, and `recv` as a library using lock-protected buffers
- Define $c \rightarrowtail prot$ using Iris's invariants and ghost state
- 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
- ✓ Small Coq development (∼5000 lines in total)
- ✓ Readily integrates with other concurrency mechanisms in Iris

Draft [LMCS]: https://itu.dk/people/jkas/papers/actris_lmcs.pdf
Mechanisation: https://gitlab.mpi-sws.org/iris/actris/-/tree/lmcs



ACTRIS 2.0: ASYNCHRONOUS SESSION-TYPE BASED REASONING IN SEPARATION LOGIC

JONAS KASTBERG HINRICHSEN, JESPER BENGTSON, AND ROBBERT KREBBERS

IT University of Copenhagen, Denmark
e-mail address: jkas@itu.dk

IT University of Copenhagen, Denmark
e-mail address: bengtson@itu.dk

Radboud University and Delft University of Technology, The Netherlands
e-mail address: mail@robbertkrebbers.nl

ABSTRACT. Message passing is a useful abstraction to implement concurrent programs. For real-world systems, however, it is often combined with other programming and concurrency paradigms, such as higher-order functions, mutable state, shared-memory concurrency, and locks. We present **Actris**: a logic for proving functional correctness of programs that use a combination of the aforementioned features. Actris combines the power of modern concurrent separation logics with a first-class protocol mechanism—based on session types—for reasoning about message passing in the presence of other concurrency paradigms. We show that Actris provides a suitable level of abstraction by proving functional correctness of a variety of examples, including a distributed merge sort, a distributed load-balancing mapper, and a variant of the map-reduce model, using concise specifications.

While Actris was already presented in a conference paper (POPL'20), this paper expands the prior presentation significantly. Moreover, it extends Actris to **Actris 2.0** with a notion of *subprotocols*—based on session-type subtyping—that permits additional flexibility when composing channel endpoints, and that takes full advantage of the asynchronous semantics

Ongoing and future work

Ongoing and future work

Ongoing work

- Multi-party dependent separation protocols (based on [Honda et al., POPL'08])
- Dependent separation protocols as specifications for TCP-based communication in distributed systems

Future Work

- Deadlock free communication (based on ongoing work by Jules Jacobs)
- Linearity of channels through Iron [Bizjak et al., POPL'19]

$$
\begin{array}{l}
\textbf{!}\,\langle \text{''}\textit{Thank you}\text{''}\rangle\{\texttt{ActrisKnowledge}\}.\\
\mu \mathit{rec}.\,\textbf{?}(q : \texttt{Question})\,\langle q\rangle\{\texttt{AboutActris}\ q\}.\\
\qquad \textbf{!}\,(a : \texttt{Answer})\,\langle a\rangle\{\texttt{Insightful}\ a\}.\,\mathit{rec}
\end{array}
$$