Verifying Functional Correctness of Message-Passing Programs with Separation Logic

Separation Logic meets Session Types

Jonas Kastberg Hinrichsen

Jesper Bengtson Robbert Krebbers Jules Jacobs Daniël Louwrink
Léon Gondelman Mário Pereira Amin Timany Lars Birkedal

May 14. 2024, Uppsala University

Why: Formal Verification of Message Passing

Rigid validation needed

- Critical infrastructure based on concurrency
- Concurrency notoriously hard

Why: Formal Verification of Message Passing

Rigid validation needed

- Critical infrastructure based on concurrency
- Concurrency notoriously hard

Concurrency beyond reach of testing

- Covering all possible schedulings is infeasible

Why: Formal Verification of Message Passing

Rigid validation needed

- Critical infrastructure based on concurrency
- Concurrency notoriously hard

Concurrency beyond reach of testing

- Covering all possible schedulings is infeasible

Formal verification to the rescue

- Considers all possible executions at once

Why: Formal Verification of Message Passing

Rigid validation needed

- Critical infrastructure based on concurrency
- Concurrency notoriously hard

Concurrency beyond reach of testing

- Covering all possible schedulings is infeasible

Formal verification to the rescue

- Considers all possible executions at once

Verifying strong guarantees often require manual interaction

- Manual interaction demands good abstractions

Why: Formal Verification of Message Passing

Rigid validation needed

- Critical infrastructure based on concurrency
- Concurrency notoriously hard

Concurrency beyond reach of testing

- Covering all possible schedulings is infeasible

Formal verification to the rescue

- Considers all possible executions at once

Verifying strong guarantees often require manual interaction

- Manual interaction demands good abstractions

Message passing is a good (and necessary) abstraction

- **Good:** Used in shared memory (Go)
- **Necessary:** Inherent to distributed systems (TCP)

What: Message Passing Concurrency in Shared Memory

Shared-memory message passing concurrency:

- Structured approach to concurrent programming
- Threads act as services or clients
- Used in Go, Scala, C#, and more

What: Message Passing Concurrency in Shared Memory

Shared-memory message passing concurrency:

- Structured approach to concurrent programming
- Threads act as services or clients
- Used in Go, Scala, C#, and more

Bi-directional asynchronous session channels:

`new_chan`()	Create channel and return two endpoints c1 and c2
c.**send**(v)	Send value v over endpoint c
c.**recv**()	Receive and return next inbound value on endpoint c

What: Message Passing Concurrency in Shared Memory

Shared-memory message passing concurrency:

- Structured approach to concurrent programming
- Threads act as services or clients
- Used in Go, Scala, C#, and more

Bi-directional asynchronous session channels:

$\texttt{new\_chan}\,()$	Create channel and return two endpoints c1 and c2
$c.\texttt{send}(v)$	Send value v over endpoint c
$c.\texttt{recv}()$	Receive and return next inbound value on endpoint c

Example Program:

$$
\begin{array}{l}
\texttt{let}\,(c_1, c_2) = \texttt{new\_chan}\,()\;\texttt{in} \\
\left(\begin{array}{l}
c_1.\texttt{send}(40); \\
\texttt{let}\,y = c_1.\texttt{recv}()\;\texttt{in} \\
\texttt{assert}(y = 42)
\end{array}
\middle\|
\begin{array}{l}
\texttt{let}\,x = c_2.\texttt{recv}()\;\texttt{in} \\
c_2.\texttt{send}(x+2) \\
\\
\end{array}\right)
\end{array}
$$

How: Separation Logic Meets Session Types

Example Program:

$$\begin{array}{l} \texttt{let}\,(c_1, c_2) = \texttt{new\_chan}\,()\,\texttt{in} \\ \left(\begin{array}{l} c_1.\texttt{send}(40); \\ \texttt{let}\,y = c_1.\texttt{recv}()\,\texttt{in} \\ \texttt{assert}(y = 42) \end{array} \middle\| \begin{array}{l} \texttt{let}\,x = c_2.\texttt{recv}()\,\texttt{in} \\ c_2.\texttt{send}(x + 2) \\ \end{array}\right) \end{array}$$

Goal: Prove crash-freedom (safety) in presence of asserts (functional correctness)

How: Separation Logic Meets Session Types

Example Program:

$$
\begin{array}{l}
\mathtt{let}\,(c_1, c_2) = \mathtt{new\_chan}\,()\ \mathtt{in} \\
\left(\begin{array}{l||l}
c_1.\mathtt{send}(40); & \mathtt{let}\ x = c_2.\mathtt{recv}()\ \mathtt{in} \\
\mathtt{let}\ y = c_1.\mathtt{recv}()\ \mathtt{in} & c_2.\mathtt{send}(x + 2) \\
\mathtt{assert}(y = 42) &
\end{array}\right)
\end{array}
$$

Goal: Prove crash-freedom (safety) in presence of asserts (functional correctness)

Safety	**Functional correctness**
Type systems	Separation logics

How: Separation Logic Meets Session Types

Example Program:

$$
\begin{array}{l}
\mathtt{let}\,(c_1, c_2) = \mathtt{new\_chan}\,()\;\mathtt{in} \\
\left(\begin{array}{l}
c_1.\mathtt{send}(40); \\
\mathtt{let}\, y = c_1.\mathtt{recv}()\;\mathtt{in} \\
\mathtt{assert}(y = 42)
\end{array}
\;\middle\|\;
\begin{array}{l}
\mathtt{let}\, x = c_2.\mathtt{recv}()\;\mathtt{in} \\
c_2.\mathtt{send}(x + 2) \\
\\
\end{array}\right)
\end{array}
$$

Goal: Prove crash-freedom (safety) in presence of asserts (functional correctness)

Safety	**Functional correctness**
Type systems	Separation logics
Session types	???

How: Separation Logic Meets Session Types

Example Program:

$$
\begin{array}{l}
\texttt{let}\,(c_1, c_2) = \texttt{new\_chan}\,()\ \texttt{in} \\
\left(\begin{array}{l} c_1.\texttt{send}(40); \\ \texttt{let}\,y = c_1.\texttt{recv}()\ \texttt{in} \\ \texttt{assert}(y = 42) \end{array} \middle\| \begin{array}{l} \texttt{let}\,x = c_2.\texttt{recv}()\ \texttt{in} \\ c_2.\texttt{send}(x + 2) \\ \end{array}\right)
\end{array}
$$

Goal: Prove crash-freedom (safety) in presence of asserts (functional correctness)

Safety	**Functional correctness**
Type systems	Separation logics
Session types	???
$c_1 : \text{chan}\,(\mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\textbf{end})$	???

! is send, **?** is receive

How: Separation Logic Meets Session Types

Example Program:

$$\begin{array}{l}
\texttt{let}\,(c_1, c_2) = \texttt{new\_chan}\,()\,\texttt{in} \\
\left(\begin{array}{l||l}
c_1.\texttt{send}(40); & \texttt{let}\,x = c_2.\texttt{recv}()\,\texttt{in} \\
\texttt{let}\,y = c_1.\texttt{recv}()\,\texttt{in} & c_2.\texttt{send}(x + 2) \\
\texttt{assert}(y = 42) &
\end{array}\right)
\end{array}$$

Goal: Prove crash-freedom (safety) in presence of asserts (functional correctness)

Safety	**Functional correctness**
Type systems	Separation logics
Session types	Dependent separation protocols
$c_1 : \mathsf{chan}\,(\mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\mathbf{end})$	$c_1 \rightarrowtail \mathbf{!}\,\langle 40\rangle.\,\mathbf{?}\langle 42\rangle.\,\mathbf{end}$

! is send, **?** is receive

How: Separation Logic Meets Session Types

Example Program:

$$\begin{array}{l} \texttt{let}\,(c_1, c_2) = \texttt{new\_chan}\,()\ \texttt{in} \\ \left(\begin{array}{l} c_1.\texttt{send}(40); \\ \texttt{let}\ y = c_1.\texttt{recv}()\ \texttt{in} \\ \texttt{assert}(y = 42) \end{array} \middle\| \begin{array}{l} \texttt{let}\ x = c_2.\texttt{recv}()\ \texttt{in} \\ c_2.\texttt{send}(x + 2) \\ \end{array} \right) \end{array}$$

Goal: Prove crash-freedom (safety) in presence of asserts (functional correctness)

Safety	**Functional correctness**
Type systems	Separation logics
Session types	Dependent separation protocols
$c_1 : \text{chan}\,(!\mathbb{Z}.\,?\mathbb{Z}.\,\textbf{end})$	$c_1 \rightarrowtail \,!\,\langle 40\rangle.\,?\langle 42\rangle.\,\textbf{end}$

! is send, **?** is receive

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

- **Separation logic:** Modular verification of stateful programs

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

- **Separation logic:** Modular verification of stateful programs
- **Higher-order:** Enables high-level abstractions

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

- **Separation logic:** Modular verification of stateful programs
- **Higher-order:** Enables high-level abstractions
- **Concurrent:** Reasoning about (fine-grained) concurrency

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

- **Separation logic:** Modular verification of stateful programs
- **Higher-order:** Enables high-level abstractions
- **Concurrent:** Reasoning about (fine-grained) concurrency
- **Mechanized in Coq:** Validation in the Coq proof assistant with tactic support

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

- **Separation logic:** Modular verification of stateful programs
- **Higher-order:** Enables high-level abstractions
- **Concurrent:** Reasoning about (fine-grained) concurrency
- **Mechanized in Coq:** Validation in the Coq proof assistant with tactic support
- 100+ publications since 2015: https://iris-project.org/

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

- **Separation logic:** Modular verification of stateful programs
- **Higher-order:** Enables high-level abstractions
- **Concurrent:** Reasoning about (fine-grained) concurrency
- **Mechanized in Coq:** Validation in the Coq proof assistant with tactic support
- 100+ publications since 2015: https://iris-project.org/

Actris: Session type-based extension of Iris

- **Session type-based:** Reasoning about message-passing concurrency via dependent separation protocols

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

- **Separation logic:** Modular verification of stateful programs
- **Higher-order:** Enables high-level abstractions
- **Concurrent:** Reasoning about (fine-grained) concurrency
- **Mechanized in Coq:** Validation in the Coq proof assistant with tactic support
- 100+ publications since 2015: https://iris-project.org/

Actris: Session type-based extension of Iris

- **Session type-based:** Reasoning about message-passing concurrency via dependent separation protocols
- Verified distributed merge-sort, distributed mapper, map-reduce, remote procedure calls, and more

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

- **Separation logic:** Modular verification of stateful programs
- **Higher-order:** Enables high-level abstractions
- **Concurrent:** Reasoning about (fine-grained) concurrency
- **Mechanized in Coq:** Validation in the Coq proof assistant with tactic support
- 100+ publications since 2015: https://iris-project.org/

Actris: Session type-based extension of Iris

- **Session type-based:** Reasoning about message-passing concurrency via dependent separation protocols
- Verified distributed merge-sort, distributed mapper, map-reduce, remote procedure calls, and more
- Also applied to distributed systems

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

- **Separation logic:** Modular verification of stateful programs
- **Higher-order:** Enables high-level abstractions
- **Concurrent:** Reasoning about (fine-grained) concurrency
- **Mechanized in Coq:** Validation in the Coq proof assistant with tactic support
- 100+ publications since 2015: https://iris-project.org/

Actris: Session type-based extension of Iris

- **Session type-based:** Reasoning about message-passing concurrency via dependent separation protocols
- Verified distributed merge-sort, distributed mapper, map-reduce, remote procedure calls, and more
- Also applied to distributed systems
- 6 publications since 2020: https://iris-project.org/actris/

Roadmap of This Talk

Separation Logic

- Safety and functional correctness
- Modular verification
- Verification of example program

Actris

- Reasoning methodology for message passing
- Demonstration of select Actris features

Beyond this talk

- Sample Actris features
- The Actris line of work

Separation Logic

[O'Hearn, Reynolds, Yang 2001]

Language Under Consideration

HeapLang: Untyped OCaml-like language

$$\begin{aligned} v, w \in \mathsf{Val} &::= z \mid \mathtt{true} \mid \mathtt{false} \mid () \mid \ell & (z \in \mathbb{Z}, \ell \in \mathsf{Loc}) \\ e \in \mathsf{Expr} &::= v \mid x \mid e_1\ e_2 \mid \\ &\quad\ \mathtt{ref}\, e \mid {!\,e} \mid e_1 \leftarrow e_2 \mid \\ &\quad\ (e_1 \parallel e_2) \mid \mathtt{assert}(e) \ldots \end{aligned}$$

Language Under Consideration

HeapLang: Untyped OCaml-like language

$$
\begin{aligned}
v, w \in \mathsf{Val} &::= z \mid \mathbf{true} \mid \mathbf{false} \mid () \mid \ell && (z \in \mathbb{Z}, \ell \in \mathsf{Loc}) \\
e \in \mathsf{Expr} &::= v \mid x \mid e_1\ e_2 \mid \\
&\quad\ \mathbf{ref}\, e \mid !\, e \mid e_1 \leftarrow e_2 \mid \\
&\quad\ (e_1 \parallel e_2) \mid \mathbf{assert}(e) \ldots
\end{aligned}
$$

Example program:

$$
\begin{aligned}
&\mathbf{let}\ \ell_1 = \mathbf{ref}\, 0\ \mathbf{in} \\
&\mathbf{let}\ \ell_2 = \mathbf{ref}\, 0\ \mathbf{in} \\
&\big(\ell_1 \leftarrow !\,\ell_1 + 2 \ \big\|\ \ell_2 \leftarrow !\,\ell_2 + 2\big)\,; \\
&\mathbf{assert}(!\,\ell_1 + !\,\ell_2 = 4)
\end{aligned}
$$

Language Under Consideration

HeapLang: Untyped OCaml-like language

$$
\begin{aligned}
v, w \in \mathsf{Val} &::= z \mid \mathtt{true} \mid \mathtt{false} \mid () \mid \ell && (z \in \mathbb{Z}, \ell \in \mathsf{Loc}) \\
e \in \mathsf{Expr} &::= v \mid x \mid e_1\ e_2 \mid \\
&\qquad \mathtt{ref}\, e \mid \,!\, e \mid e_1 \leftarrow e_2 \mid \\
&\qquad (e_1 \parallel e_2) \mid \mathtt{assert}(e) \ldots
\end{aligned}
$$

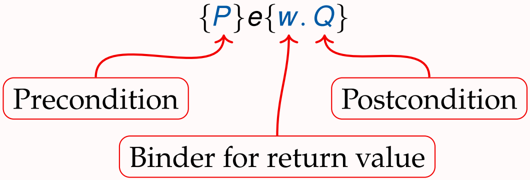
Example program:

$$
\begin{array}{l}
\mathtt{let}\ \ell_1 = \mathtt{ref}\, 0\ \mathtt{in} \\
\mathtt{let}\ \ell_2 = \mathtt{ref}\, 0\ \mathtt{in} \\
\big(\ell_1 \leftarrow\, !\, \ell_1 + 2 \,\big\|\, \ell_2 \leftarrow\, !\, \ell_2 + 2\big)\, ; \\
\mathtt{assert}(!\, \ell_1 + !\, \ell_2 = 4)
\end{array}
$$

Goal: Program does not crash

Hoare Triples

Hoare triples for partial functional correctness:



$$\{P\}e\{w.\,Q\}$$

Precondition

Binder for return value

Postcondition

If the initial state satisfies P, then:

- **Safety:** e does not crash
- **Postcondition validity:** if e terminates with value v, then the final state satisfies $Q[v/w]$

Separation Logic

Separation logic propositions assert ownership of resources

The points-to connective $\ell \mapsto v$

- Provides the knowledge that location ℓ has value v, and
- Provides exclusive ownership of ℓ

Separating conjunction $P * Q$ captures that the state consists of disjoint parts satisfying P and Q.

Separation Logic

Separation logic propositions assert ownership of resources

The points-to connective $\ell \mapsto v$

- Provides the knowledge that location ℓ has value v, and
- Provides exclusive ownership of ℓ

Separating conjunction $P * Q$ captures that the state consists of disjoint parts satisfying P and Q.

Enables modular reasoning, through disjointness:

$$\frac{\{P\}\, e\, \{w.\, Q\}}{\{P * R\}\, e\, \{w.\, Q * R\}}\ \textsc{Ht-frame}$$

Hoare Triples for Seperation Logic

Hoare triples for references:

HT-ALLOC
$\{\mathsf{True}\}\ \mathtt{\textbf{ref}}\, v\ \{\ell.\, \ell \mapsto v\}$

HT-LOAD
$\{\ell \mapsto v\}\ !\,\ell\ \{w.\, w = v * \ell \mapsto v\}$

HT-STORE
$\{\ell \mapsto v\}\ \ell \leftarrow w\ \{\ell \mapsto w\}$

Hoare Triples for Seperation Logic

Hoare triples for references:

Ht-alloc
$\{\mathsf{True}\}\ \mathbf{ref}\, v\ \{\ell.\, \ell \mapsto v\}$

Ht-load
$\{\ell \mapsto v\}\ !\,\ell\ \{w.\, w = v * \ell \mapsto v\}$

Ht-store
$\{\ell \mapsto v\}\ \ell \leftarrow w\ \{\ell \mapsto w\}$

Hoare triples for structural expressions:

Ht-let
$$\frac{\{P\}\ e_1\ \{w_1.\, Q\} \qquad \forall w_1.\, \{Q\}\ e_2[w_1/x]\ \{w_2.\, R\}}{\{P\}\ \mathbf{let}\, x = e_1\ \mathbf{in}\, e_2\ \{w_2.\, R\}}$$

Ht-assert
$$\frac{\{P\}\ e\ \{w.\, w = \mathbf{true} * Q\}}{\{P\}\ \mathbf{assert}(e)\ \{Q\}}$$

Ht-seq
$$\frac{\{P\}\ e_1\ \{w_1.\, Q\} \qquad \forall w_1.\, \{Q\}\ e_2\ \{w_2.R\}}{\{P\}\ e_1 ; e_2\ \{w_2.\, R\}}$$

Ht-par
$$\frac{\{P_1\}\ e_1\ \{Q_1\} \qquad \{P_2\}\ e_2\ \{Q_2\}}{\{P_1 * P_2\}\ (e_1 \parallel e_2)\ \{Q_1 * Q_2\}}$$

Example Program - Verified

$$
\begin{array}{l}
\mathbf{let}\ \ell_1 = \mathbf{ref}\ 0\ \mathbf{in} \\
\mathbf{let}\ \ell_2 = \mathbf{ref}\ 0\ \mathbf{in} \\
\left(\ell_1 \leftarrow\ !\,\ell_1 + 2 \parallel \ell_2 \leftarrow\ !\,\ell_2 + 2\right); \\
\mathbf{assert}(!\,\ell_1 + !\,\ell_2 = 4)
\end{array}
$$

Example Program - Verified

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{\textbf{let}}\ \ell_1 = \mathtt{\textbf{ref}}\, 0\ \mathtt{\textbf{in}} \\
\mathtt{\textbf{let}}\ \ell_2 = \mathtt{\textbf{ref}}\, 0\ \mathtt{\textbf{in}} \\
\left(\ell_1 \leftarrow !\,\ell_1 + 2 \;\middle\|\; \ell_2 \leftarrow !\,\ell_2 + 2\right); \\
\mathtt{\textbf{assert}}(!\,\ell_1 + !\,\ell_2 = 4) \\
\{\mathsf{True}\}
\end{array}
$$

Example Program - Verified

$\{\mathsf{True}\}$
$\mathtt{let}\ \ell_1 = \mathtt{ref}\,0\ \mathtt{in}$ // HT-LET, HT-ALLOC
$\{\ell_1 \mapsto 0\}$
$\mathtt{let}\ \ell_2 = \mathtt{ref}\,0\ \mathtt{in}$
$\big(\ell_1 \leftarrow\ !\,\ell_1 + 2 \ \big\|\ \ell_2 \leftarrow\ !\,\ell_2 + 2\big)\ ;$
$\mathtt{assert}(!\,\ell_1 + !\,\ell_2 = 4)$
$\{\mathsf{True}\}$

Example Program - Verified

$$
\begin{array}{ll}
\{\mathsf{True}\} & \\
\mathbf{let}\ \ell_1 = \mathbf{ref}\,0\ \mathbf{in} & //\ \textsc{Ht-let}, \textsc{Ht-alloc} \\
\{\ell_1 \mapsto 0\} & \\
\mathbf{let}\ \ell_2 = \mathbf{ref}\,0\ \mathbf{in} & //\ \textsc{Ht-let}, \textsc{Ht-alloc}, \textsc{Ht-frame} \\
\{\ell_1 \mapsto 0 * \ell_2 \mapsto 0\} & \\
\big(\ell_1 \leftarrow\, !\,\ell_1 + 2 \;\big\|\; \ell_2 \leftarrow\, !\,\ell_2 + 2\big)\,; & \\
\mathbf{assert}(!\,\ell_1 + !\,\ell_2 = 4) & \\
\{\mathsf{True}\} &
\end{array}
$$

Example Program - Verified

$\{\mathsf{True}\}$

$\mathbf{let}\ \ell_1 = \mathbf{ref}\,0\ \mathbf{in}$ // HT-LET, HT-ALLOC

$\{\ell_1 \mapsto 0\}$

$\mathbf{let}\ \ell_2 = \mathbf{ref}\,0\ \mathbf{in}$ // HT-LET, HT-ALLOC, HT-FRAME

$\{\ell_1 \mapsto 0 * \ell_2 \mapsto 0\}$

$$\left(\begin{array}{l} \{\ell_1 \mapsto 0\} \\ \ell_1 \leftarrow\, !\,\ell_1 + 2 \end{array} \middle\| \begin{array}{l} \{\ell_2 \mapsto 0\} \\ \ell_2 \leftarrow\, !\,\ell_2 + 2 \end{array}\right);$$

// HT-SEQ, HT-PAR

$\mathbf{assert}(!\,\ell_1 + !\,\ell_2 = 4)$

$\{\mathsf{True}\}$

Example Program - Verified

$$
\begin{array}{ll}
\{\mathsf{True}\} & \\
\mathtt{let}\ \ell_1 = \mathtt{ref}\ 0\ \mathtt{in} & /\!/\ \textsc{Ht-let}, \textsc{Ht-alloc} \\
\{\ell_1 \mapsto 0\} & \\
\mathtt{let}\ \ell_2 = \mathtt{ref}\ 0\ \mathtt{in} & /\!/\ \textsc{Ht-let}, \textsc{Ht-alloc}, \textsc{Ht-frame} \\
\{\ell_1 \mapsto 0 * \ell_2 \mapsto 0\} & \\
\left(\begin{array}{l|l}
\{\ell_1 \mapsto 0\} & \{\ell_2 \mapsto 0\} \\
\ell_1 \leftarrow {!}\,\ell_1 + 2 & \ell_2 \leftarrow {!}\,\ell_2 + 2 \\
\{\ell_1 \mapsto 2\} & \{\ell_2 \mapsto 2\}
\end{array}\right); & /\!/\ \textsc{Ht-seq}, \textsc{Ht-par}, \textsc{Ht-load}, \textsc{Ht-store} \\
\mathtt{assert}({!}\,\ell_1 + {!}\,\ell_2 = 4) & \\
\{\mathsf{True}\} &
\end{array}
$$

Example Program - Verified

$$
\begin{array}{ll}
\{\mathsf{True}\} & \\
\mathtt{let}\ \ell_1 = \mathtt{ref}\ 0\ \mathtt{in} & /\!/\ \textsc{Ht-let}, \textsc{Ht-alloc} \\
\{\ell_1 \mapsto 0\} & \\
\mathtt{let}\ \ell_2 = \mathtt{ref}\ 0\ \mathtt{in} & /\!/\ \textsc{Ht-let}, \textsc{Ht-alloc}, \textsc{Ht-frame} \\
\{\ell_1 \mapsto 0 * \ell_2 \mapsto 0\} & \\
\left(\begin{array}{l} \{\ell_1 \mapsto 0\} \\ \ell_1 \leftarrow\, !\,\ell_1 + 2 \\ \{\ell_1 \mapsto 2\} \end{array} \middle\| \begin{array}{l} \{\ell_2 \mapsto 0\} \\ \ell_2 \leftarrow\, !\,\ell_2 + 2 \\ \{\ell_2 \mapsto 2\} \end{array}\right); & /\!/\ \textsc{Ht-seq}, \textsc{Ht-par}, \textsc{Ht-load}, \textsc{Ht-store} \\
\{\ell_1 \mapsto 2 * \ell_2 \mapsto 2\} & \\
\mathtt{assert}(!\,\ell_1 + !\,\ell_2 = 4) & \\
\{\mathsf{True}\} &
\end{array}
$$

Example Program - Verified

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\ \ell_1 = \mathtt{ref}\,0\ \mathtt{in} \qquad // \textsc{Ht-let}, \textsc{Ht-alloc} \\
\{\ell_1 \mapsto 0\} \\
\mathtt{let}\ \ell_2 = \mathtt{ref}\,0\ \mathtt{in} \qquad // \textsc{Ht-let}, \textsc{Ht-alloc}, \textsc{Ht-frame} \\
\{\ell_1 \mapsto 0 * \ell_2 \mapsto 0\} \\
\left(\begin{array}{l} \{\ell_1 \mapsto 0\} \\ \ell_1 \leftarrow\ !\,\ell_1 + 2 \\ \{\ell_1 \mapsto 2\} \end{array} \middle\| \begin{array}{l} \{\ell_2 \mapsto 0\} \\ \ell_2 \leftarrow\ !\,\ell_2 + 2 \\ \{\ell_2 \mapsto 2\} \end{array}\right); \qquad // \textsc{Ht-seq}, \textsc{Ht-par}, \textsc{Ht-load}, \textsc{Ht-store} \\
\{\ell_1 \mapsto 2 * \ell_2 \mapsto 2\} \\
\mathtt{assert}(!\,\ell_1 + !\,\ell_2 = 4) \qquad // \textsc{Ht-load}, \textsc{Ht-assert} \\
\{\mathsf{True}\}
\end{array}
$$

But What About Channels?

Example Program:

$$\begin{array}{l}\mathtt{let}\,(c_1, c_2) = \mathtt{new\_chan}\,()\;\mathtt{in} \\ \left(\begin{array}{l} c_1.\mathtt{send}(40); \\ \mathtt{let}\,y = c_1.\mathtt{recv}()\;\mathtt{in} \\ \mathtt{assert}(y = 42) \end{array} \;\middle\|\; \begin{array}{l} \mathtt{let}\,x = c_2.\mathtt{recv}()\;\mathtt{in} \\ c_2.\mathtt{send}(x + 2) \\ \end{array}\right)\end{array}$$

Goal: Program does not crash

But What About Channels?

Example Program:

$$\begin{array}{l} \mathbf{let}\ (c_1, c_2) = \mathbf{new\_chan}\,()\ \mathbf{in} \\ \left(\begin{array}{l} c_1.\mathbf{send}(40); \\ \mathbf{let}\ y = c_1.\mathbf{recv}()\ \mathbf{in} \\ \mathbf{assert}(y = 42) \end{array} \middle\| \begin{array}{l} \mathbf{let}\ x = c_2.\mathbf{recv}()\ \mathbf{in} \\ c_2.\mathbf{send}(x + 2) \\ \end{array}\right) \end{array}$$

Goal: Program does not crash
Sub-Goal: Hoare triples for channel primitives

Ht-new
$\{???\}\ \mathbf{new\_chan}\,()\ \{???\}$

Ht-send
$\{???\}\ c.\mathbf{send}(v)\ \{???\}$

Ht-recv
$\{???\}\ c.\mathbf{recv}()\ \{???\}$

But What About Channels?

Example Program:

$$\begin{array}{l} \mathbf{let}\,(c_1, c_2) = \mathbf{new\_chan}\,()\,\mathbf{in} \\ \left(\begin{array}{l} c_1.\mathbf{send}(40); \\ \mathbf{let}\,y = c_1.\mathbf{recv}()\,\mathbf{in} \\ \mathbf{assert}(y = 42) \end{array} \middle\| \begin{array}{l} \mathbf{let}\,x = c_2.\mathbf{recv}()\,\mathbf{in} \\ c_2.\mathbf{send}(x + 2) \\ \\ \end{array}\right) \end{array}$$

Goal: Program does not crash
Sub-Goal: Hoare triples for channel primitives

Ht-new
$\{???\}\ \mathbf{new\_chan}\,()\ \{???\}$

Ht-send
$\{???\}\ c.\mathbf{send}(v)\ \{???\}$

Ht-recv
$\{???\}\ c.\mathbf{recv}()\ \{???\}$

Key Idea: Separate channel endpoint ownership à la Session Types

Actris

Hinrichsen et al.

Actris

Channel Endpoint Ownership: $c \rightarrowtail p$

Actris

Channel Endpoint Ownership: $c \rightarrowtail p$

Dependent Separation Protocols: $\mathbf{!}\,\langle v \rangle.\, p \mid \mathbf{?}\langle v \rangle.\, p \mid \textbf{end}$

Actris

Channel Endpoint Ownership: $c \rightarrowtail p$

Dependent Separation Protocols: $\mathbf{!}\,\langle v \rangle.\, p \mid \mathbf{?}\langle v \rangle.\, p \mid \textbf{end}$

Example: $\mathbf{!}\,\langle 40 \rangle.\, \mathbf{?}\langle 42 \rangle.\, \textbf{end}$

Actris

Channel Endpoint Ownership: $c \rightarrowtail p$

Dependent Separation Protocols: $\mathbf{!}\,\langle v \rangle.\, p \mid \mathbf{?}\langle v \rangle.\, p \mid \textbf{end}$

Example: $\mathbf{!}\,\langle 40 \rangle.\, \mathbf{?}\langle 42 \rangle.\, \textbf{end}$

Rules:

Ht-send

$$\{c \rightarrowtail \mathbf{!}\,\langle v \rangle.\, p\}\; c.\mathtt{send}(v)\; \{c \rightarrowtail p\}$$

Ht-recv

$$\{c \rightarrowtail \mathbf{?}\langle v \rangle.\, p\}\; c.\mathtt{recv}()\; \{w.\, w = v * c \rightarrowtail p\}$$

Actris

Channel Endpoint Ownership: $c \rightarrowtail p$

Dependent Separation Protocols: $\mathbf{!}\,\langle v \rangle.\, p \mid \mathbf{?}\langle v \rangle.\, p \mid \mathbf{end}$

Example: $\mathbf{!}\,\langle 40 \rangle.\, \mathbf{?}\langle 42 \rangle.\, \mathbf{end}$

Duality: $\overline{\mathbf{!}\,\langle v \rangle.\, p} = \mathbf{?}\langle v \rangle.\, \overline{p}$ $\quad \overline{\mathbf{?}\langle v \rangle.\, p} = \mathbf{!}\,\langle v \rangle.\, \overline{p}$ $\quad \overline{\mathbf{end}} = \mathbf{end}$

Rules:

Ht-send
$\{c \rightarrowtail \mathbf{!}\,\langle v \rangle.\, p\}\; c.\mathbf{send}(v)\; \{c \rightarrowtail p\}$

Ht-recv
$\{c \rightarrowtail \mathbf{?}\langle v \rangle.\, p\}\; c.\mathtt{recv}()\; \{w.\, w = v * c \rightarrowtail p\}$

Actris

Channel Endpoint Ownership: $c \rightarrowtail p$

Dependent Separation Protocols: $\mathbf{!}\,\langle v \rangle .\, p \mid \mathbf{?}\langle v \rangle .\, p \mid \mathbf{end}$

Example: $\mathbf{!}\,\langle 40 \rangle .\, \mathbf{?}\langle 42 \rangle .\, \mathbf{end}$

Duality: $\overline{\mathbf{!}\,\langle v \rangle .\, p} = \mathbf{?}\langle v \rangle .\, \overline{p} \qquad \overline{\mathbf{?}\langle v \rangle .\, p} = \mathbf{!}\,\langle v \rangle .\, \overline{p} \qquad \overline{\mathbf{end}} = \mathbf{end}$

Rules:

Ht-send
$\{c \rightarrowtail \mathbf{!}\,\langle v \rangle .\, p\}\; c.\mathtt{\textbf{send}}(v)\; \{c \rightarrowtail p\}$

Ht-recv
$\{c \rightarrowtail \mathbf{?}\langle v \rangle .\, p\}\; c.\mathtt{\textbf{recv}}()\; \{w.\, w = v * c \rightarrowtail p\}$

Ht-new
$\{\mathsf{True}\}\; \mathtt{\textbf{new\_chan}}\,()\; \{(c_1, c_2).\, c_1 \rightarrowtail p * c_2 \rightarrowtail \overline{p}\}$

Example Channel Program - Verified

Example Program:

$$\begin{array}{l}\mathtt{let}\,(c_1, c_2) = \mathtt{new\_chan}\,()\,\mathtt{in}\\ \left(\begin{array}{l||l} c_1.\mathtt{send}(40); & \mathtt{let}\,x = c_2.\mathtt{recv}()\,\mathtt{in}\\ \mathtt{let}\,y = c_1.\mathtt{recv}()\,\mathtt{in} & c_2.\mathtt{send}(x+2)\\ \mathtt{assert}(y = 42) & \end{array}\right)\end{array}$$

Example Channel Program - Verified

Example Program:

$$
\begin{array}{l}
\texttt{let}\,(c_1, c_2) = \texttt{new\_chan}\,()\,\texttt{in} \\
\left(\begin{array}{l}
c_1.\texttt{send}(40); \\
\texttt{let}\,y = c_1.\texttt{recv}()\,\texttt{in} \\
\texttt{assert}(y = 42)
\end{array}
\;\middle\|\;
\begin{array}{l}
\texttt{let}\,x = c_2.\texttt{recv}()\,\texttt{in} \\
c_2.\texttt{send}(x + 2) \\
\\
\end{array}\right)
\end{array}
$$

Protocols:

$$c_1 \rightarrowtail \mathbf{!}\,\langle 40\rangle.\,\mathbf{?}\langle 42\rangle.\,\mathbf{end}$$

$$c_2 \rightarrowtail \mathbf{?}\langle 40\rangle.\,\mathbf{!}\,\langle 42\rangle.\,\mathbf{end}$$

Example Channel Program - Verified

Example Program:

$$
\begin{array}{l}
\texttt{let}\,(c_1, c_2) = \texttt{new\_chan}\,()\,\texttt{in} \\
\left(\begin{array}{l||l}
c_1.\texttt{send}(40); & \texttt{let}\,x = c_2.\texttt{recv}()\,\texttt{in} \\
\texttt{let}\,y = c_1.\texttt{recv}()\,\texttt{in} & c_2.\texttt{send}(x + 2) \\
\texttt{assert}(y = 42) &
\end{array}\right)
\end{array}
$$

Protocols:

$$
\begin{array}{l}
c_1 \rightarrowtail \mathbf{!}\,\langle 40 \rangle.\,\mathbf{?}\langle 42 \rangle.\,\mathbf{end} \\
c_2 \rightarrowtail \mathbf{?}\langle 40 \rangle.\,\mathbf{!}\,\langle 42 \rangle.\,\mathbf{end}
\end{array}
$$

Goal complete: Program verified safe to execute for any scheduling

Example Channel Program - Verified

Example Program:

$$\texttt{let}\,(c_1, c_2) = \texttt{new\_chan}\,()\,\texttt{in}$$
$$\left(\begin{array}{l} c_1.\texttt{send}(40); \\ \texttt{let}\; y = c_1.\texttt{recv}()\;\texttt{in} \\ \texttt{assert}(y = 42) \end{array} \middle\| \begin{array}{l} \texttt{let}\; x = c_2.\texttt{recv}()\;\texttt{in} \\ c_2.\texttt{send}(x + 2) \\ \end{array}\right)$$

Protocols:

$$c_1 \rightarrowtail \,!\,\langle 40\rangle.\, ?\langle 42\rangle.\, \textbf{end}$$
$$c_2 \rightarrowtail \,?\langle 40\rangle.\, !\,\langle 42\rangle.\, \textbf{end}$$

Goal complete: Program verified safe to execute for any scheduling
Problem: Protocols too restrictive; right thread works for any integer

Actris with Quantifiers

Dependent Separation Protocols: $\mathbf{!}\,(\vec{x}:\vec{\tau})\langle v\rangle.\,p \mid \mathbf{?}\,(\vec{x}:\vec{\tau})\langle v\rangle.\,p \mid \mathbf{end}$

Example: $\mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle.\,\mathbf{?}\langle x+2\rangle.\,\mathbf{end}$

Duality: $\overline{\mathbf{!}\,(\vec{x}:\vec{\tau})\langle v\rangle.\,p} = \mathbf{?}\,(\vec{x}:\vec{\tau})\langle v\rangle.\,\overline{p}$ $\qquad$ $\overline{\mathbf{?}\,(\vec{x}:\vec{\tau})\langle v\rangle.\,p} = \mathbf{!}\,(\vec{x}:\vec{\tau})\langle v\rangle.\,\overline{p}$

Rules:

Ht-send

$$\{c \rightarrowtail \mathbf{!}\,(\vec{x}:\vec{\tau})\langle v\rangle.\,p\}\; c.\mathbf{send}(v[\vec{t}/\vec{x}])\; \{c \rightarrowtail p[\vec{t}/\vec{x}]\}$$

Ht-recv

$$\{c \rightarrowtail \mathbf{?}\,(\vec{x}:\vec{\tau})\langle v\rangle.\,p\}\; c.\mathbf{recv}()\; \{w.\, \exists(\vec{t}:\vec{\tau}).\, w = v[\vec{t}/\vec{x}] * c \rightarrowtail p[\vec{t}/\vec{x}]\}$$

Example Channel Program - Quantifiers

Example Program:

$$\begin{array}{l}
\mathbf{let}\,(c_1, c_2) = \mathbf{new\_chan}\,()\,\mathbf{in} \\
\left(\begin{array}{l} c_1.\mathbf{send}(40); \\ \mathbf{let}\, y = c_1.\mathbf{recv}()\,\mathbf{in} \\ \mathbf{assert}(y = 42) \end{array} \middle\| \begin{array}{l} \mathbf{let}\, x = c_2.\mathbf{recv}()\,\mathbf{in} \\ c_2.\mathbf{send}(x + 2) \\ \end{array}\right)
\end{array}$$

Example Channel Program - Quantifiers

Example Program:

$$\texttt{let}\,(c_1, c_2) = \texttt{new\_chan}\,()\ \texttt{in}$$
$$\left(\begin{array}{l} c_1.\texttt{send}(40); \\ \texttt{let}\, y = c_1.\texttt{recv}()\ \texttt{in} \\ \texttt{assert}(y = 42) \end{array} \middle\| \begin{array}{l} \texttt{let}\, x = c_2.\texttt{recv}()\ \texttt{in} \\ c_2.\texttt{send}(x + 2) \\ \ \end{array}\right)$$

Protocols:

$$c_1 \rightarrowtail\, !\,(x : \mathbb{Z})\,\langle x \rangle.\, ?\langle x + 2 \rangle.\, \textbf{end}$$
$$c_2 \rightarrowtail\, ?(x : \mathbb{Z})\,\langle x \rangle.\, !\,\langle x + 2 \rangle.\, \textbf{end}$$

Example Channel Program - Quantifiers

Example Program:

$$\begin{array}{l}\texttt{let}\,(c_1, c_2) = \texttt{new\_chan}\,()\,\texttt{in}\\ \left(\begin{array}{l} c_1.\texttt{send}(40);\\ \texttt{let}\,y = c_1.\texttt{recv}()\,\texttt{in}\\ \texttt{assert}(y = 42) \end{array} \;\middle\|\; \begin{array}{l} \texttt{let}\,x = c_2.\texttt{recv}()\,\texttt{in}\\ c_2.\texttt{send}(x+2)\\ \end{array}\right)\end{array}$$

Protocols:

$$c_1 \rightarrowtail\, !\,(x : \mathbb{Z})\,\langle x\rangle.\, ?\langle x+2\rangle.\,\mathbf{end}$$

$$c_2 \rightarrowtail\, ?(x : \mathbb{Z})\,\langle x\rangle.\, !\,\langle x+2\rangle.\,\mathbf{end}$$

Goal complete: Right thread now modularly compose with arbitrary clients

Example Reference Program

Example Program:

$$\mathtt{let}\,(c_1, c_2) = \mathtt{new\_chan}\,()\ \mathtt{in}$$
$$\left(\begin{array}{l||l} \mathtt{let}\ \ell = \mathtt{ref}\ 40\ \mathtt{in} & \mathtt{let}\ \ell = c_2.\mathtt{recv}()\ \mathtt{in} \\ c_1.\mathtt{send}(\ell); & \ell \leftarrow\ !\,\ell + 2; \\ c_1.\mathtt{recv}(); & c_2.\mathtt{send}() \\ \mathtt{assert}(!\,\ell = 42) & \end{array}\right)$$

$\ell \mapsto v$: Ownership of reference ℓ pointing to v

$\{\mathsf{True}\}\ \mathtt{ref}\,v\ \{\ell.\ \ell \mapsto v\}$ $\quad$ $\{\ell \mapsto v\}\ !\,v\ \{w.\ w = v * \ell \mapsto v\}$ $\quad$ $\{\ell \mapsto v\}\ \ell \leftarrow w\ \{\ell \mapsto w\}$

Example Reference Program

Example Program:

$$\begin{array}{l}\mathbf{let}\,(c_1, c_2) = \mathbf{new\_chan}\,()\ \mathbf{in} \\ \left(\begin{array}{l}\mathbf{let}\,\ell = \mathbf{ref}\,40\ \mathbf{in} \\ c_1.\mathbf{send}(\ell); \\ c_1.\mathbf{recv}(); \\ \mathbf{assert}(!\,\ell = 42)\end{array} \middle\| \begin{array}{l}\mathbf{let}\,\ell = c_2.\mathbf{recv}()\ \mathbf{in} \\ \ell \leftarrow !\,\ell + 2; \\ c_2.\mathbf{send}() \\ \\ \end{array}\right)\end{array}$$

Protocols?

$$c_1 \rightarrowtail \mathbf{!}\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle.\,\mathbf{?}\langle () \rangle.\,\mathbf{end}$$

$$c_2 \rightarrowtail \mathbf{?}(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle.\,\mathbf{!}\,\langle () \rangle.\,\mathbf{end}$$

$\ell \mapsto v$: Ownership of reference ℓ pointing to v

$\{\mathsf{True}\}\ \mathbf{ref}\,v\ \{\ell.\,\ell \mapsto v\}$ $\quad \{\ell \mapsto v\}\ !\,v\ \{w.\,w = v * \ell \mapsto v\}$ $\quad \{\ell \mapsto v\}\ \ell \leftarrow w\ \{\ell \mapsto w\}$

Example Reference Program

Example Program:

$$\begin{array}{l}
\mathtt{let}\,(c_1, c_2) = \mathtt{new\_chan}\,()\,\mathtt{in}\\
\left(\begin{array}{l} \mathtt{let}\,\ell = \mathtt{ref}\,40\,\mathtt{in}\\ c_1.\mathtt{send}(\ell);\\ c_1.\mathtt{recv}();\\ \mathtt{assert}(!\,\ell = 42)\end{array} \middle\| \begin{array}{l} \mathtt{let}\,\ell = c_2.\mathtt{recv}()\,\mathtt{in}\\ \ell \leftarrow !\,\ell + 2;\\ c_2.\mathtt{send}()\\ \\ \end{array}\right)
\end{array}$$

Protocols?

$$c_1 \rightarrowtail \mathbf{!}\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle.\,\mathbf{?}\langle () \rangle.\,\mathbf{end}$$

$$c_2 \rightarrowtail \mathbf{?}(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle.\,\mathbf{!}\,\langle () \rangle.\,\mathbf{end}$$

Problem: Implicit transfer of control not possible to capture

---

$\ell \mapsto v$: Ownership of reference ℓ pointing to v

$\{\mathrm{True}\}\ \mathtt{ref}\,v\ \{\ell.\ \ell \mapsto v\}\quad \{\ell \mapsto v\}\ !\,v\ \{w.\ w = v * \ell \mapsto v\}\quad \{\ell \mapsto v\}\ \ell \leftarrow w\ \{\ell \mapsto w\}$

Example Reference Program

Example Program:

$$\begin{array}{l}
\mathtt{let}\,(c_1, c_2) = \mathtt{new\_chan}\,()\;\mathtt{in} \\
\left(\begin{array}{l||l}
\mathtt{let}\;\ell = \mathtt{ref}\,40\;\mathtt{in} & \mathtt{let}\;\ell = c_2.\mathtt{recv}()\;\mathtt{in} \\
c_1.\mathtt{send}(\ell); & \ell \leftarrow {!}\,\ell + 2; \\
c_1.\mathtt{recv}(); & c_2.\mathtt{send}() \\
\mathtt{assert}({!}\,\ell = 42) &
\end{array}\right)
\end{array}$$

Protocols?

$$c_1 \rightarrowtail \mathbf{!}\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle.\,\mathbf{?}\langle () \rangle.\,\mathbf{end}$$

$$c_2 \rightarrowtail \mathbf{?}(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle.\,\mathbf{!}\,\langle () \rangle.\,\mathbf{end}$$

Problem: Implicit transfer of control not possible to capture

Key Idea: Resources in protocols

$\ell \mapsto v$: Ownership of reference ℓ pointing to v

$\{\mathrm{True}\}\;\mathtt{ref}\,v\;\{\ell.\,\ell \mapsto v\}\quad \{\ell \mapsto v\}\;{!}\,v\;\{w.\,w = v * \ell \mapsto v\}\quad \{\ell \mapsto v\}\;\ell \leftarrow w\;\{\ell \mapsto w\}$

Actris with Resources

Dependent Separation Protocols: $\mathbf{!}\,(\vec{x}:\vec{\tau})\langle v\rangle\{P\}.\,p \mid \mathbf{?}\,(\vec{x}:\vec{\tau})\langle v\rangle\{P\}.\,p \mid \mathbf{end}$

Example: $\mathbf{!}\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell\rangle\{\ell \mapsto x\}.\,\mathbf{?}\langle ()\rangle\{\ell \mapsto (x+2)\}.\,\mathbf{end}$

Duality: $\overline{\mathbf{!}\,(\vec{x}:\vec{\tau})\langle v\rangle\{P\}.\,p} = \mathbf{?}\,(\vec{x}:\vec{\tau})\langle v\rangle\{P\}.\,\overline{p}$ $\qquad$ $\overline{\mathbf{?}\,(\vec{x}:\vec{\tau})\langle v\rangle\{P\}.\,p} = \mathbf{!}\,(\vec{x}:\vec{\tau})\langle v\rangle\{P\}.\,\overline{p}$

Rules:

Ht-send

$$\left\{c \rightarrowtail \mathbf{!}\,(\vec{x}:\vec{\tau})\langle v\rangle\{P\}.\,p * P[\vec{t}/\vec{x}]\right\}\; c.\mathbf{send}(v[\vec{t}/\vec{x}])\; \left\{c \rightarrowtail p[\vec{t}/\vec{x}]\right\}$$

Ht-recv

$$\{c \rightarrowtail \mathbf{?}\,(\vec{x}:\vec{\tau})\langle v\rangle\{P\}.\,p\}\; c.\mathbf{recv}()\; \left\{w.\, \exists(\vec{t} : \vec{\tau}).\, w = v[\vec{t}/\vec{x}] * c \rightarrowtail p[\vec{t}/\vec{x}] * P[\vec{t}/\vec{x}]\right\}$$

Example Reference Program - Verified

Example Program:

$$\begin{array}{l}
\mathbf{let}\,(c_1, c_2) = \mathbf{new\_chan}\,()\,\mathbf{in} \\
\left(\begin{array}{l}
\mathbf{let}\,\ell = \mathbf{ref}\,40\,\mathbf{in} \\
c_1.\mathbf{send}(\ell); \\
c_1.\mathbf{recv}(); \\
\mathbf{assert}(!\,\ell = 42)
\end{array}
\middle\|
\begin{array}{l}
\mathbf{let}\,\ell = c_2.\mathbf{recv}()\,\mathbf{in} \\
\ell \leftarrow !\,\ell + 2; \\
c_2.\mathbf{send}() \\
\\
\end{array}\right)
\end{array}$$

Protocols:

$$c_1 \rightarrowtail \mathbf{!}\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{?}\langle () \rangle \{\ell \mapsto (x+2)\}.\,\mathbf{end}$$

$$c_2 \rightarrowtail \mathbf{?}(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,\langle () \rangle \{\ell \mapsto (x+2)\}.\,\mathbf{end}$$

$\ell \mapsto v$: Ownership of reference ℓ pointing to v

$\{\mathrm{True}\}\ \mathbf{ref}\,v\ \{\ell.\,\ell \mapsto v\} \quad \{\ell \mapsto v\}\ !\,v\ \{w.\,w = v * \ell \mapsto v\} \quad \{\ell \mapsto v\}\ \ell \leftarrow w\ \{\ell \mapsto w\}$

Beyond This Talk

And Much More

Sample of additional Actris features:

Exchanging channels: $\mathbf{!}\,(c : \mathsf{Chan}, p : \mathsf{iProto})\,\langle c \rangle \{c \rightarrowtail \mathbf{!}\,\langle 42 \rangle .\, p\} .\, \mathbf{end}$

And Much More

Sample of additional Actris features:

Exchanging channels: $\mathbf{!}\,(c : \mathsf{Chan}, p : \mathsf{iProto})\,\langle c\rangle\{c \rightarrowtail \mathbf{!}\,\langle 42\rangle.\,p\}.\,\mathbf{end}$

Recursion: $\mu(p : \mathsf{iProto}).\,\mathbf{!}\,(c : \mathsf{Chan})\,\langle c\rangle\{c \rightarrowtail \mathbf{!}\,\langle 42\rangle.\,p\}.\,p$

And Much More

Sample of additional Actris features:

Exchanging channels: $\mathbf{!}\,(c : \mathsf{Chan}, p : \mathsf{iProto})\,\langle c\rangle\{c \rightarrowtail \mathbf{!}\,\langle 42\rangle.\,p\}.\,\mathbf{end}$

Recursion: $\mu(p : \mathsf{iProto}).\,\mathbf{!}\,(c : \mathsf{Chan})\,\langle c\rangle\{c \rightarrowtail \mathbf{!}\,\langle 42\rangle.\,p\}.\,p$

Exchanging closures: $\mathbf{!}\,(f : \mathsf{Val}, \Phi : \mathsf{Val} \to \mathsf{iProp})\,\langle f\rangle\{(\{\mathsf{True}\}\, f\, ()\, \{w.\Phi\, w\})\}.$
$\mathbf{?}(w : \mathsf{Val})\,\langle w\rangle\{\Phi\, w\}.\,\mathbf{end}$

And Much More

Sample of additional Actris features:

Exchanging channels: $\mathbf{!}\,(c : \mathsf{Chan}, p : \mathsf{iProto})\ \langle c\rangle\{c \rightarrowtail \mathbf{!}\,\langle 42\rangle .\, p\}.\ \mathbf{end}$

Recursion: $\mu(p : \mathsf{iProto}).\ \mathbf{!}\,(c : \mathsf{Chan})\ \langle c\rangle\{c \rightarrowtail \mathbf{!}\,\langle 42\rangle .\, p\}.\, p$

Exchanging closures: $\mathbf{!}\,(f : \mathsf{Val}, \Phi : \mathsf{Val} \to \mathsf{iProp})\ \langle f\rangle\{(\{\mathsf{True}\}\ f\ ()\ \{w.\Phi\ w\})\}.$
$\mathbf{?}(w : \mathsf{Val})\ \langle w\rangle\{\Phi\ w\}.\ \mathbf{end}$

Verified programs:

- Distributed merge-sort
- Distributed load-balancing mapper
- Shared-memory Map-Reduce
- Remote procedure calls
- Distributed locks

And Much More

Sample of additional Actris features:

Exchanging channels: $\mathbf{!}\,(c : \mathsf{Chan}, p : \mathsf{iProto})\,\langle c\rangle\{c \rightarrowtail \mathbf{!}\,\langle 42\rangle\,.\,p\}\,.\,\mathbf{end}$

Recursion: $\mu(p : \mathsf{iProto})\,.\,\mathbf{!}\,(c : \mathsf{Chan})\,\langle c\rangle\{c \rightarrowtail \mathbf{!}\,\langle 42\rangle\,.\,p\}\,.\,p$

Exchanging closures: $\mathbf{!}\,(f : \mathsf{Val}, \Phi : \mathsf{Val} \to \mathsf{iProp})\,\langle f\rangle\{(\{\mathsf{True}\}\ f\ ()\ \{w.\Phi\ w\})\}\,.$
$\mathbf{?}(w : \mathsf{Val})\,\langle w\rangle\{\Phi\ w\}\,.\,\mathbf{end}$

Verified programs:

- Distributed merge-sort
- Distributed load-balancing mapper
- Shared-memory Map-Reduce
- Remote procedure calls
- Distributed locks

Fully validated and mechanized in Iris in Coq up to operational semantics

The Actris Line of Work

[POPL'20] Actris (This talk)

- Dependent separation protocols
- Actris for shared memory message passing

The Actris Line of Work

[POPL'20] Actris (This talk)

- Dependent separation protocols
- Actris for shared memory message passing

[CPP'21] Semantic Session Type System [Distinguished Paper Award]

- Semantic session types: $!A.\,S \triangleq !\,(v : \mathsf{Val})\,\langle v \rangle \{A\ v\}.\,S$

The Actris Line of Work

[POPL'20] Actris (This talk)

- Dependent separation protocols
- Actris for shared memory message passing

[CPP'21] Semantic Session Type System [Distinguished Paper Award]

- Semantic session types: $!A.\, S \triangleq \,!\, (v : \mathsf{Val})\, \langle v \rangle \{A\ v\}.\, S$

[LMCS'22] Actris 2.0

- Language-parametric validation of Actris rules
- Subprotocols: $!\,(x : \mathbb{Z})\, \langle x \rangle.\, ?\langle x + 2 \rangle.\, \mathbf{end} \quad \sqsubseteq \quad !\, \langle 40 \rangle.\, ?\langle 42 \rangle.\, \mathbf{end}$

The Actris Line of Work

[POPL'20] Actris (This talk)

- Dependent separation protocols
- Actris for shared memory message passing

[CPP'21] Semantic Session Type System [Distinguished Paper Award]

- Semantic session types: $!A.\,S \triangleq \mathbf{!}\,(v : \mathsf{Val})\,\langle v \rangle \{A\ v\}.\,S$

[LMCS'22] Actris 2.0

- Language-parametric validation of Actris rules
- Subprotocols: $\mathbf{!}\,(x : \mathbb{Z})\,\langle x \rangle.\,\mathbf{?}\langle x + 2 \rangle.\,\mathbf{end} \quad \sqsubseteq \quad \mathbf{!}\,\langle 40 \rangle.\,\mathbf{?}\langle 42 \rangle.\,\mathbf{end}$

[ICFP'23a] Actris in Distributed Systems

The Actris Line of Work

[POPL'20] Actris (This talk)

- Dependent separation protocols
- Actris for shared memory message passing

[CPP'21] Semantic Session Type System [Distinguished Paper Award]

- Semantic session types: $!A.\,S \triangleq \mathbf{!}\,(v : \mathsf{Val})\,\langle v\rangle\{A\ v\}.\,S$

[LMCS'22] Actris 2.0

- Language-parametric validation of Actris rules
- Subprotocols: $\mathbf{!}\,(x : \mathbb{Z})\,\langle x\rangle.\,\mathbf{?}\langle x + 2\rangle.\,\mathbf{end} \quad \sqsubseteq \quad \mathbf{!}\,\langle 40\rangle.\,\mathbf{?}\langle 42\rangle.\,\mathbf{end}$

[ICFP'23a] Actris in Distributed Systems
[ICFP'23b] Minimalistic Actris (Session channels via one-shot channels)

The Actris Line of Work

[POPL'20] Actris (This talk)

- Dependent separation protocols
- Actris for shared memory message passing

[CPP'21] Semantic Session Type System [Distinguished Paper Award]

- Semantic session types: $!A.\, S \triangleq \mathbf{!}\,(v : \mathsf{Val})\,\langle v\rangle\{A\ v\}.\, S$

[LMCS'22] Actris 2.0

- Language-parametric validation of Actris rules
- Subprotocols: $\mathbf{!}\,(x : \mathbb{Z})\,\langle x\rangle.\,\mathbf{?}\langle x+2\rangle.\,\mathbf{end} \quad \sqsubseteq \quad \mathbf{!}\,\langle 40\rangle.\,\mathbf{?}\langle 42\rangle.\,\mathbf{end}$

[ICFP'23a] Actris in Distributed Systems
[ICFP'23b] Minimalistic Actris (Session channels via one-shot channels)
[POPL'24] Deadlock-freedom via Actris

The Actris Line of Work

[POPL'20] Actris (This talk)

- Dependent separation protocols
- Actris for shared memory message passing

[CPP'21] Semantic Session Type System [Distinguished Paper Award]

- Semantic session types: $!A.\,S \triangleq !\,(v : \mathsf{Val})\,\langle v\rangle\{A\;v\}.\,S$

[LMCS'22] Actris 2.0

- Language-parametric validation of Actris rules
- Subprotocols: $!\,(x : \mathbb{Z})\,\langle x\rangle.\,?\langle x+2\rangle.\,\mathbf{end} \quad \sqsubseteq \quad !\,\langle 40\rangle.\,?\langle 42\rangle.\,\mathbf{end}$

[ICFP'23a] Actris in Distributed Systems
[ICFP'23b] Minimalistic Actris (Session channels via one-shot channels)
[POPL'24] Deadlock-freedom via Actris

[Ongoing Work] Multiparty Actris: $!\,[i]\,(x : \mathbb{Z})\,\langle x\rangle.\,?[j]\,\langle x+2\rangle.\,\mathbf{end}$

The Actris Line of Work

[POPL'20] Actris (This talk)

- Dependent separation protocols
- Actris for shared memory message passing

[CPP'21] Semantic Session Type System [Distinguished Paper Award]

- Semantic session types: $!A.\,S \triangleq\ !\,(v : \mathsf{Val})\,\langle v \rangle \{A\ v\}.\,S$

[LMCS'22] Actris 2.0

- Language-parametric validation of Actris rules
- Subprotocols: $!\,(x : \mathbb{Z})\,\langle x \rangle.\,?\langle x + 2 \rangle.\,\mathbf{end} \quad \sqsubseteq \quad !\,\langle 40 \rangle.\,?\langle 42 \rangle.\,\mathbf{end}$

[ICFP'23a] Actris in Distributed Systems
[ICFP'23b] Minimalistic Actris (Session channels via one-shot channels)
[POPL'24] Deadlock-freedom via Actris

[Ongoing Work] Multiparty Actris: $!\,[i]\,(x : \mathbb{Z})\,\langle x \rangle.\,?[j]\,\langle x + 2 \rangle.\,\mathbf{end}$

$$
\begin{aligned}
&\mathbf{!}\,\langle \text{“Thank you”} \rangle \{ \mathtt{ActrisKnowledge} \}. \\
&\mu \mathit{rec}.\, \mathbf{?}(q : \mathtt{Question})\, \langle q \rangle \{ \mathtt{AboutActris}\; q \}. \\
&\qquad\quad \mathbf{!}\,(a : \mathtt{Answer})\, \langle a \rangle \{ \mathtt{Insightful}\; q\; a \}.\, \mathit{rec}
\end{aligned}
$$