Verified Message-Passing Concurrency in Iris

Separation Logic Meets Session Types

Jonas Kastberg Hinrichsen
Aarhus University

Jules Jacobs
Cornell University

Robbert Krebbers
Radboud University Nijmegen

Tutorial Timeline

Part 1: 14:00 – 15:30

- Introduction (10 min)
- Layered implementation of session channels (10 min)
- Basic concurrent separation logic and one-shot protocols (30 min)
- **Break** (10 min)
- Dependent separation protocols (30 min)

Break (30 min)

Part 2: 16:00 – 17:30

- Iris invariants and ghost state (30 min)
- **Break** (10 min)
- Supervised Coq hacking (50 min)

Message Passing Concurrency

Shared-memory message passing concurrency:

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

Message Passing Concurrency

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 session channels:

new ()	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

Message Passing Concurrency

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 session channels:

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

Example program:

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

Safety and Functional Correctness

Example Program:

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

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

Safety and Functional Correctness

Example Program:

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

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

Safety	Functional correctness
Type systems	Program logics

Safety and Functional Correctness

Example Program:

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

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

Safety	**Functional correctness**
Type systems	Program logics
Session types	Dependent separation protocols

Safety and Functional Correctness

Example Program:

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

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

Safety	Functional correctness
Type systems	Program logics
Session types	Dependent separation protocols
$c_1 : \mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\textbf{end}$	$c_1 \rightarrowtail \mathbf{!}\,\langle 40 \rangle.\,\mathbf{?}\langle 42 \rangle.\,\textbf{end}$

! is send, **?** is receive

Safety and Functional Correctness

Example Program:

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

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

Safety	**Functional correctness**
Type systems	Program logics
Session types	Dependent separation protocols
$c_1 : \mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\mathbf{end}$	$c_1 \rightarrowtail \mathbf{!}\,\langle 40\rangle.\,\mathbf{?}\langle 42\rangle.\,\mathbf{end}$
Automatic checking	Interactive proofs

! is send, **?** is receive

Safety and Functional Correctness

Example Program:

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

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

Safety	Functional correctness
Type systems	Program logics
Session types	Dependent separation protocols
$c_1 : \mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\textbf{end}$	$c_1 \rightarrowtail \mathbf{!}\,\langle 40 \rangle.\,\mathbf{?}\langle 42 \rangle.\,\textbf{end}$
Automatic checking	Interactive proofs

! is send, **?** is receive

Iris: Higher-order concurrent separation logic mechanized in Coq

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

- **Separation logic:** Modular reasoning about stateful programs

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

- **Separation logic:** Modular reasoning about stateful programs
- **Higher-order:** Supports high-level abstractions

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

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

Iris and Actris

Iris: Higher-order concurrent separation logic mechanized in Coq

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

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 reasoning about stateful programs
- **Higher-order:** Supports high-level abstractions
- **Concurrent:** Reasoning about (fine-grained) concurrency
- **Mechanized in Coq:** Validation in the Coq proof assistant with tactic support

Actris: Session type-based extension of Iris

- **Session type-based:** Reasoning about message-passing concurrency via dependent separation protocols
- **MiniActris:** Layered minimalistic version of Actris from first principles (ICFP'23 Functional Pearl)

Learning Goals of this Tutorial

After this tutorial you will be able to:

- Design layers of abstractions in concurrent separation logic
- Verify sample programs using these abstractions
- Verify these abstractions using the Iris methodology
- Mechanize these results using the Iris Proof Mode in Coq

Overview of Abstraction Layers

Layer	**Reasoning principles / specifications**
#1 Iris's HeapLang	Basic concurrent separation logic Iris invariants and ghost state
#2 One-shot channels	One-shot protocols
#3 Functional session channels	Dependent separation protocols
#4 Session channels	Dependent separation protocols

Layered implementation of session channels ala. MiniActris

Tutorial Timeline

Part 1: 14:00 – 15:30

- ~~Introduction (10 min)~~
- Layered implementation of session channels (10 min)
- Basic concurrent separation logic and one-shot protocols (30 min)
- **Break** (10 min)
- Dependent separation protocols (30 min)

Break (30 min)

Part 2: 16:00 – 17:30

- Iris invariants and ghost state (30 min)
- **Break** (10 min)
- Supervised Coq hacking (50 min)

Overview of Abstraction Layers

Layer	Reasoning principles / specifications
#1 Iris's HeapLang	Basic concurrent separation logic Iris invariants and ghost state
#2 One-shot channels	One-shot protocols
#3 Functional session channels	Dependent separation protocols
#4 Session channels	Dependent separation protocols

Layer #1: Iris's Heap Lang

Untyped OCaml-like language with

- Mutable references
- Higher-order recursive functions
- Parallel composition-based concurrency
- Assert statements

$$
\begin{aligned}
v, w \in \mathsf{Val} ::= {} & z \mid \mathbf{true} \mid \mathbf{false} \mid () \mid \ell \mid && (z \in \mathbb{Z}, \ell \in \mathsf{Loc}) \\
& \mathbf{rec}\ f\ x = e \mid (v, w) \mid \mathbf{Some}\, v \mid \mathbf{None} \\
e \in \mathsf{Expr} ::= {} & v \mid x \mid e_1\ e_2 \mid \\
& \mathbf{fst}\, e \mid \mathbf{snd}\, e \mid \\
& (\mathbf{match}\, e\, \mathbf{with}\ \mathbf{Some}\, v \Rightarrow e_1 \mid \mathbf{None} \Rightarrow e_2\ \mathbf{end}) \mid \\
& \mathbf{ref}\, e \mid \mathbf{free}\, e \mid {!\,e} \mid e_1 \leftarrow e_2 \mid \\
& (e_1 \parallel e_2) \mid \mathbf{assert}(e) \mid \ldots
\end{aligned}
$$

Example Program – Sequential

Simple sequential program:

$$\begin{aligned}
&\texttt{ref_prog} \triangleq \\
&\quad \textbf{let}\ \ell = \textbf{ref}\ \textbf{None}\ \textbf{in} \\
&\quad \ell \leftarrow \textbf{Some}\ 42; \\
&\quad \textbf{let}\ x = !\,\ell\ \textbf{in} \\
&\quad \textbf{free}\ \ell; \\
&\quad \textbf{assert}(x = \textbf{Some}\ 42)
\end{aligned}$$

The **assert** statement halts the program if the condition does not reduce to **true**

One-shot channel implementation:

$$
\begin{aligned}
\mathtt{new1}\,() &\triangleq \mathtt{ref}\ \mathtt{None} \\
\mathtt{send1}\,c\,v &\triangleq c \leftarrow \mathtt{Some}\,v \\
\mathtt{recv1}\,c &\triangleq \mathtt{let}\,x = !\,c\,\mathtt{in} \\
&\quad\ \mathtt{match}\,x\,\mathtt{with} \\
&\quad\quad \mathtt{None} \quad \Rightarrow \mathtt{recv1}\,c \\
&\quad\ \mid \mathtt{Some}\,v \Rightarrow \mathtt{free}\,c;\ v \\
&\quad\ \mathtt{end}
\end{aligned}
$$

Layer #2: One-Shot Channels

One-shot channel implementation:

$$
\begin{aligned}
\texttt{new1}\,() &\triangleq \texttt{ref}\ \texttt{None} \\
\texttt{send1}\,c\,v &\triangleq c \leftarrow \texttt{Some}\,v \\
\texttt{recv1}\,c &\triangleq \texttt{let}\ x = {!}\,c\ \texttt{in} \\
&\qquad \texttt{match}\ x\ \texttt{with} \\
&\qquad\ \ \texttt{None} \quad \Rightarrow \texttt{recv1}\,c \\
&\qquad \mid \texttt{Some}\,v \Rightarrow \texttt{free}\,c;\ v \\
&\qquad \texttt{end}
\end{aligned}
$$

Concurrent program that uses one-shot channels:

$$
\begin{aligned}
&\texttt{oneshot\_prog} \triangleq \\
&\quad \texttt{let}\ c = \texttt{new1}\,()\ \texttt{in} \\
&\quad \left(\texttt{send1}\,c\,42 \;\middle\|\; \begin{array}{l} \texttt{let}\ x = \texttt{recv1}\,c\ \texttt{in} \\ \texttt{assert}(x = 42) \end{array} \right)
\end{aligned}
$$

Example Programs – Reference Passing

Passing references over one-shot channels:

$$
\begin{array}{l}
\texttt{oneshot\_ref\_prog} \triangleq \\
\quad \textbf{let}\, c = \textbf{new1}\,()\,\textbf{in} \\
\quad \left(\begin{array}{l} \textbf{let}\,\ell = \textbf{ref}\,42\,\textbf{in} \\ \textbf{send1}\,c\,\ell \\ \end{array} \;\middle\|\; \begin{array}{l} \textbf{let}\,\ell = \textbf{recv1}\,c\,\textbf{in} \\ \textbf{let}\,x = !\,\ell\,\textbf{in}\,\textbf{free}\,\ell; \\ \textbf{assert}(x = 42) \end{array}\right)
\end{array}
$$

Passing one-shot channels over one-shot channels:

$$
\begin{array}{l}
\texttt{oneshot\_chan\_prog} \triangleq \\
\quad \mathbf{let}\, c = \mathbf{new1}\,()\;\mathbf{in} \\
\quad \left(\begin{array}{l}
\mathbf{let}\,\ell = \mathbf{ref}\,40\;\mathbf{in} \\
\mathbf{let}\,c' = \mathbf{new1}\,()\;\mathbf{in}\;\mathbf{send1}\,c\,(\ell, c'); \\
\mathbf{recv1}\,c'; \\
\mathbf{let}\,x = !\,\ell\;\mathbf{in}\;\mathbf{free}\,\ell; \\
\mathbf{assert}(x = 42)
\end{array}
\;\middle\|\;
\begin{array}{l}
\mathbf{let}\,(\ell, c') = \mathbf{recv1}\,c\;\mathbf{in} \\
\ell \leftarrow (!\,\ell + 2); \\
\mathbf{send1}\,c'\,() \\
\\
\\
\end{array}\right)
\end{array}
$$

Layer #3: Functional Session Channels

Implementation (inspired by Kobayashi et al., Dardha et al.):

$$\mathbf{new}_{\mathbf{fun}}\,() \triangleq \mathbf{new1}\,()$$

$$\mathbf{send}\,c\,v \triangleq \mathbf{let}\,c' = \mathbf{new1}\,()\,\mathbf{in}\,\mathbf{send1}\,c\,(v, c');\; c' \qquad \mathbf{close}\,c \triangleq \mathbf{send1}\,c\,()$$

$$\mathbf{recv}\,c \triangleq \mathbf{recv1}\,c \qquad \mathbf{wait}\,c \triangleq \mathbf{recv1}\,c$$

Recovering the one-shot channel example:

$$\begin{array}{l}
\texttt{ses_fun_ref_prog} \triangleq \\
\quad \mathbf{let}\,c = \mathbf{new}_{\mathbf{fun}}\,()\,\mathbf{in} \\
\quad \left(\begin{array}{l|l}
\mathbf{let}\,\ell = \mathbf{ref}\,40\,\mathbf{in} & \mathbf{let}\,(\ell, c') = \mathbf{recv}\,c\,\mathbf{in} \\
\mathbf{let}\,c' = \mathbf{send}\,c\,\ell\,\mathbf{in} & \ell \leftarrow (!\,\ell + 2); \\
\mathbf{wait}\,c'; & \mathbf{close}\,c' \\
\mathbf{let}\,x = !\,\ell\,\mathbf{in}\,\mathbf{free}\,\ell; & \\
\mathbf{assert}(x = 42) &
\end{array}\right)
\end{array}$$

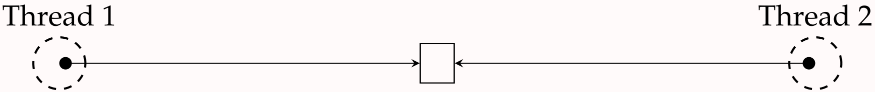
Implementation (inspired by Kobayashi et al., Dardha et al.):

$$\mathtt{new_{fun}}\,() \triangleq \mathtt{new1}\,()$$

$$\mathtt{send}\,c\,v \triangleq \mathtt{let}\,c' = \mathtt{new1}\,()\,\mathtt{in}\,\mathtt{send1}\,c\,(v, c');\ c' \qquad \mathtt{close}\,c \triangleq \mathtt{send1}\,c\,()$$

$$\mathtt{recv}\,c \triangleq \mathtt{recv1}\,c \qquad \mathtt{wait}\,c \triangleq \mathtt{recv1}\,c$$

Emerging polarized bi-directional linked list:



$\mathtt{let}\,c_1 = \mathtt{send}\,c_1\,0\,\mathtt{in}$
$\mathtt{let}\,c_1 = \mathtt{send}\,c_1\,1\,\mathtt{in}$
$\mathtt{let}\,c_1 = \mathtt{send}\,c_1\,2\,\mathtt{in}$
$\mathtt{let}\,(c_1, \_) = \mathtt{recv}\,c_1\,\mathtt{in}$

$\mathtt{let}\,(c_2, \_) = \mathtt{recv}\,c_2\,\mathtt{in}$
$\mathtt{let}\,(c_2, \_) = \mathtt{recv}\,c_2\,\mathtt{in}$
$\mathtt{let}\,(c_2, \_) = \mathtt{recv}\,c_2\,\mathtt{in}$
$\mathtt{let}\,c_2 = \mathtt{send}\,c_2\,3\,\mathtt{in}$

Layer #3: Functional Session Channels – Intuition

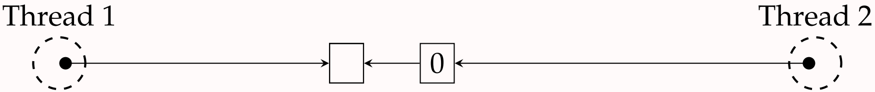
Implementation (inspired by Kobayashi et al., Dardha et al.):

$$\mathtt{new_{fun}}\;() \triangleq \mathtt{new1}\;()$$

$$\mathtt{send}\;c\;v \triangleq \mathtt{let}\;c' = \mathtt{new1}\;()\;\mathtt{in}\;\mathtt{send1}\;c\;(v, c');\;\; c' \qquad \mathtt{close}\;c \triangleq \mathtt{send1}\;c\;()$$

$$\mathtt{recv}\;c \triangleq \mathtt{recv1}\;c \qquad \mathtt{wait}\;c \triangleq \mathtt{recv1}\;c$$

Emerging polarized bi-directional linked list:



$\mathtt{let}\;c_1 = \mathtt{send}\;c_1\;0\;\mathtt{in}$ $\qquad$ $\mathtt{let}\;(c_2, \_) = \mathtt{recv}\;c_2\;\mathtt{in}$

$\mathtt{let}\;c_1 = \mathtt{send}\;c_1\;1\;\mathtt{in}$ $\qquad$ $\mathtt{let}\;(c_2, \_) = \mathtt{recv}\;c_2\;\mathtt{in}$

$\mathtt{let}\;c_1 = \mathtt{send}\;c_1\;2\;\mathtt{in}$ $\qquad$ $\mathtt{let}\;(c_2, \_) = \mathtt{recv}\;c_2\;\mathtt{in}$

$\mathtt{let}\;(c_1, \_) = \mathtt{recv}\;c_1\;\mathtt{in}$ $\qquad$ $\mathtt{let}\;c_2 = \mathtt{send}\;c_2\;3\;\mathtt{in}$

Layer #3: Functional Session Channels – Intuition

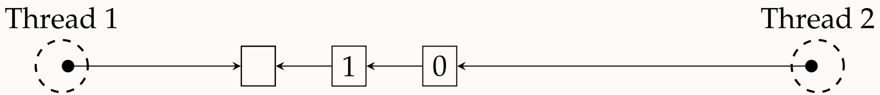
Implementation (inspired by Kobayashi et al., Dardha et al.):

$$\texttt{new}_{\texttt{fun}}\,() \triangleq \texttt{new1}\,()$$

$$\texttt{send}\,c\,v \triangleq \texttt{let}\,c' = \texttt{new1}\,()\,\texttt{in}\,\texttt{send1}\,c\,(v, c');\; c' \qquad \texttt{close}\,c \triangleq \texttt{send1}\,c\,()$$

$$\texttt{recv}\,c \triangleq \texttt{recv1}\,c \qquad \texttt{wait}\,c \triangleq \texttt{recv1}\,c$$

Emerging polarized bi-directional linked list:



$\texttt{let}\,c_1 = \texttt{send}\,c_1\,0\,\texttt{in}$
$\texttt{let}\,c_1 = \texttt{send}\,c_1\,1\,\texttt{in}$
$\texttt{let}\,c_1 = \texttt{send}\,c_1\,2\,\texttt{in}$
$\texttt{let}\,(c_1, \_) = \texttt{recv}\,c_1\,\texttt{in}$

$\texttt{let}\,(c_2, \_) = \texttt{recv}\,c_2\,\texttt{in}$
$\texttt{let}\,(c_2, \_) = \texttt{recv}\,c_2\,\texttt{in}$
$\texttt{let}\,(c_2, \_) = \texttt{recv}\,c_2\,\texttt{in}$
$\texttt{let}\,c_2 = \texttt{send}\,c_2\,3\,\texttt{in}$

Layer #3: Functional Session Channels – Intuition

Implementation (inspired by Kobayashi et al., Dardha et al.):

$$\mathbf{new_{fun}}\,() \triangleq \mathbf{new1}\,()$$

$$\mathbf{send}\,c\,v \triangleq \mathbf{let}\,c' = \mathbf{new1}\,()\,\mathbf{in}\,\mathbf{send1}\,c\,(v, c');\; c' \qquad \mathbf{close}\,c \triangleq \mathbf{send1}\,c\,()$$

$$\mathbf{recv}\,c \triangleq \mathbf{recv1}\,c \qquad \mathbf{wait}\,c \triangleq \mathbf{recv1}\,c$$

Emerging polarized bi-directional linked list:

$\mathbf{let}\,c_1 = \mathbf{send}\,c_1\,0\,\mathbf{in}$ $\qquad$ $\mathbf{let}\,(c_2, \_) = \mathbf{recv}\,c_2\,\mathbf{in}$

$\mathbf{let}\,c_1 = \mathbf{send}\,c_1\,1\,\mathbf{in}$ $\qquad$ $\mathbf{let}\,(c_2, \_) = \mathbf{recv}\,c_2\,\mathbf{in}$

$\mathbf{let}\,c_1 = \mathbf{send}\,c_1\,2\,\mathbf{in}$ $\qquad$ $\mathbf{let}\,(c_2, \_) = \mathbf{recv}\,c_2\,\mathbf{in}$

$\mathbf{let}\,(c_1, \_) = \mathbf{recv}\,c_1\,\mathbf{in}$ $\qquad$ $\mathbf{let}\,c_2 = \mathbf{send}\,c_2\,3\,\mathbf{in}$

Layer #3: Functional Session Channels – Intuition

Implementation (inspired by Kobayashi et al., Dardha et al.):

$$\mathtt{new_{fun}}\,() \triangleq \mathtt{new1}\,()$$

$$\mathtt{send}\,c\,v \triangleq \mathtt{let}\ c' = \mathtt{new1}\,()\ \mathtt{in}\ \mathtt{send1}\,c\,(v, c');\ c' \qquad \mathtt{close}\,c \triangleq \mathtt{send1}\,c\,()$$

$$\mathtt{recv}\,c \triangleq \mathtt{recv1}\,c \qquad \mathtt{wait}\,c \triangleq \mathtt{recv1}\,c$$

Emerging polarized bi-directional linked list:

$\mathtt{let}\ c_1 = \mathtt{send}\,c_1\ 0\ \mathtt{in}$

$\mathtt{let}\ c_1 = \mathtt{send}\,c_1\ 1\ \mathtt{in}$

$\mathtt{let}\ c_1 = \mathtt{send}\,c_1\ 2\ \mathtt{in}$

$\mathtt{let}\ (c_1, \_) = \mathtt{recv}\,c_1\ \mathtt{in}$

$\mathtt{let}\ (c_2, \_) = \mathtt{recv}\,c_2\ \mathtt{in}$

$\mathtt{let}\ (c_2, \_) = \mathtt{recv}\,c_2\ \mathtt{in}$

$\mathtt{let}\ (c_2, \_) = \mathtt{recv}\,c_2\ \mathtt{in}$

$\mathtt{let}\ c_2 = \mathtt{send}\,c_2\ 3\ \mathtt{in}$

Layer #3: Functional Session Channels – Intuition

Implementation (inspired by Kobayashi et al., Dardha et al.):

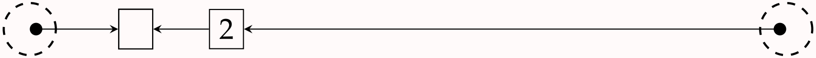
$$\mathtt{new_{fun}}\,() \triangleq \mathtt{new1}\,()$$

$$\mathtt{send}\,c\,v \triangleq \mathtt{let}\,c' = \mathtt{new1}\,()\,\mathtt{in}\,\mathtt{send1}\,c\,(v, c');\ c' \qquad \mathtt{close}\,c \triangleq \mathtt{send1}\,c\,()$$

$$\mathtt{recv}\,c \triangleq \mathtt{recv1}\,c \qquad \mathtt{wait}\,c \triangleq \mathtt{recv1}\,c$$

Emerging polarized bi-directional linked list:

Thread 1 | Thread 2



$\mathtt{let}\,c_1 = \mathtt{send}\,c_1\,0\,\mathtt{in}$

$\mathtt{let}\,c_1 = \mathtt{send}\,c_1\,1\,\mathtt{in}$

$\mathtt{let}\,c_1 = \mathtt{send}\,c_1\,2\,\mathtt{in}$

$\mathtt{let}\,(c_1, \_) = \mathtt{recv}\,c_1\,\mathtt{in}$

$\mathtt{let}\,(c_2, \_) = \mathtt{recv}\,c_2\,\mathtt{in}$

$\mathtt{let}\,(c_2, \_) = \mathtt{recv}\,c_2\,\mathtt{in}$

$\mathtt{let}\,(c_2, \_) = \mathtt{recv}\,c_2\,\mathtt{in}$

$\mathtt{let}\,c_2 = \mathtt{send}\,c_2\,3\,\mathtt{in}$

Layer #3: Functional Session Channels – Intuition

Implementation (inspired by Kobayashi et al., Dardha et al.):

$$\mathtt{new}_{\mathtt{fun}}\,() \triangleq \mathtt{new1}\,()$$

$$\mathtt{send}\,c\,v \triangleq \mathtt{let}\,c' = \mathtt{new1}\,()\,\mathtt{in}\,\mathtt{send1}\,c\,(v, c');\; c' \qquad \mathtt{close}\,c \triangleq \mathtt{send1}\,c\,()$$

$$\mathtt{recv}\,c \triangleq \mathtt{recv1}\,c \qquad \mathtt{wait}\,c \triangleq \mathtt{recv1}\,c$$

Emerging polarized bi-directional linked list:

$\mathtt{let}\,c_1 = \mathtt{send}\,c_1\,0\,\mathtt{in}$
$\mathtt{let}\,c_1 = \mathtt{send}\,c_1\,1\,\mathtt{in}$
$\mathtt{let}\,c_1 = \mathtt{send}\,c_1\,2\,\mathtt{in}$
$\mathtt{let}\,(c_1, \_) = \mathtt{recv}\,c_1\,\mathtt{in}$

$\mathtt{let}\,(c_2, \_) = \mathtt{recv}\,c_2\,\mathtt{in}$
$\mathtt{let}\,(c_2, \_) = \mathtt{recv}\,c_2\,\mathtt{in}$
$\mathtt{let}\,(c_2, \_) = \mathtt{recv}\,c_2\,\mathtt{in}$
$\mathtt{let}\,c_2 = \mathtt{send}\,c_2\,3\,\mathtt{in}$

Layer #3: Functional Session Channels – Intuition

Implementation (inspired by Kobayashi et al., Dardha et al.):

$$\mathbf{new}_{\mathbf{fun}}\,() \triangleq \mathbf{new1}\,()$$

$$\mathbf{send}\,c\,v \triangleq \mathbf{let}\;c' = \mathbf{new1}\,()\;\mathbf{in}\;\mathbf{send1}\,c\,(v, c');\; c' \qquad \mathbf{close}\,c \triangleq \mathbf{send1}\,c\,()$$

$$\mathbf{recv}\,c \triangleq \mathbf{recv1}\,c \qquad \mathbf{wait}\,c \triangleq \mathbf{recv1}\,c$$

Emerging polarized bi-directional linked list:

$\mathbf{let}\;c_1 = \mathbf{send}\,c_1\,0\;\mathbf{in}$ $\qquad$ $\mathbf{let}\;(c_2, \_) = \mathbf{recv}\,c_2\;\mathbf{in}$

$\mathbf{let}\;c_1 = \mathbf{send}\,c_1\,1\;\mathbf{in}$ $\qquad$ $\mathbf{let}\;(c_2, \_) = \mathbf{recv}\,c_2\;\mathbf{in}$

$\mathbf{let}\;c_1 = \mathbf{send}\,c_1\,2\;\mathbf{in}$ $\qquad$ $\mathbf{let}\;(c_2, \_) = \mathbf{recv}\,c_2\;\mathbf{in}$

$\mathbf{let}\;(c_1, \_) = \mathbf{recv}\,c_1\;\mathbf{in}$ $\qquad$ $\mathbf{let}\;c_2 = \mathbf{send}\,c_2\,3\;\mathbf{in}$

Layer #3: Functional Session Channels – Intuition

Implementation (inspired by Kobayashi et al., Dardha et al.):

$$\mathbf{new}_{\mathbf{fun}}\,() \triangleq \mathbf{new1}\,()$$

$$\mathbf{send}\,c\,v \triangleq \mathbf{let}\,c' = \mathbf{new1}\,()\,\mathbf{in}\,\mathbf{send1}\,c\,(v, c');\; c' \qquad \mathbf{close}\,c \triangleq \mathbf{send1}\,c\,()$$

$$\mathbf{recv}\,c \triangleq \mathbf{recv1}\,c \qquad \mathbf{wait}\,c \triangleq \mathbf{recv1}\,c$$

Emerging polarized bi-directional linked list:

$\mathbf{let}\,c_1 = \mathbf{send}\,c_1\,0\,\mathbf{in}$ | $\mathbf{let}\,(c_2, \_) = \mathbf{recv}\,c_2\,\mathbf{in}$

$\mathbf{let}\,c_1 = \mathbf{send}\,c_1\,1\,\mathbf{in}$ | $\mathbf{let}\,(c_2, \_) = \mathbf{recv}\,c_2\,\mathbf{in}$

$\mathbf{let}\,c_1 = \mathbf{send}\,c_1\,2\,\mathbf{in}$ | $\mathbf{let}\,(c_2, \_) = \mathbf{recv}\,c_2\,\mathbf{in}$

$\mathbf{let}\,(c_1, \_) = \mathbf{recv}\,c_1\,\mathbf{in}$ | $\mathbf{let}\,c_2 = \mathbf{send}\,c_2\,3\,\mathbf{in}$

Layer #4: Session Channels

Session channel implementation:

$$\mathtt{new}\,() \triangleq \mathtt{let}\, c = \mathtt{new_{fun}}\,()\,\mathtt{in}\,(\mathtt{ref}\, c, \mathtt{ref}\, c)$$

$$c.\mathtt{send}(v) \triangleq c \leftarrow \mathtt{send}\,(!\, c)\, v \qquad c.\mathtt{close}() \triangleq \mathtt{close}\,(!\, c);\, \mathtt{free}\, c$$

$$c.\mathtt{recv}() \triangleq \mathtt{let}\,(v, c') = \mathtt{recv}\; !\, c\;\mathtt{in}\, c \leftarrow c';\, v \qquad c.\mathtt{wait}() \triangleq \mathtt{wait}\,(!\, c);\, \mathtt{free}\, c$$

Session channel implementation:

$$\mathbf{new}\,() \triangleq \mathbf{let}\, c = \mathbf{new}_{\mathbf{fun}}\,()\,\mathbf{in}\,(\mathbf{ref}\, c, \mathbf{ref}\, c)$$

$$c.\mathbf{send}(v) \triangleq c \leftarrow \mathbf{send}\,(!\,c)\,v \qquad c.\mathbf{close}() \triangleq \mathbf{close}\,(!\,c); \mathbf{free}\, c$$

$$c.\mathbf{recv}() \triangleq \mathbf{let}\,(v, c') = \mathbf{recv}\,!\,c\,\mathbf{in}\, c \leftarrow c'; v \qquad c.\mathbf{wait}() \triangleq \mathbf{wait}\,(!\,c); \mathbf{free}\, c$$

Session channel example:

$$\begin{array}{l}
\texttt{ses_ref_prog} \triangleq \\
\quad \mathbf{let}\,(c_1, c_2) = \mathbf{new}\,()\,\mathbf{in} \\
\quad \left(\begin{array}{l|l}
\mathbf{let}\,\ell = \mathbf{ref}\,40\,\mathbf{in} & \mathbf{let}\,\ell = c_2.\mathbf{recv}()\,\mathbf{in} \\
c_1.\mathbf{send}(\ell); c_1.\mathbf{wait}(); & \ell \leftarrow (!\,\ell + 2); c_2.\mathbf{close}() \\
\mathbf{let}\, x = !\,\ell\,\mathbf{in}\,\mathbf{free}\,\ell; & \\
\mathbf{assert}(x = 42) &
\end{array}\right)
\end{array}$$

Layer #4: Session Channels

Session channel implementation:

$$\mathbf{new}\,() \triangleq \mathbf{let}\, c = \mathbf{new_{fun}}\,()\, \mathbf{in}\, (\mathbf{ref}\, c, \mathbf{ref}\, c)$$

$$c.\mathbf{send}(v) \triangleq c \leftarrow \mathbf{send}\,(!\, c)\, v \qquad c.\mathbf{close}() \triangleq \mathbf{close}\,(!\, c); \mathbf{free}\, c$$

$$c.\mathbf{recv}() \triangleq \mathbf{let}\,(v, c') = \mathbf{recv}\; !\, c\, \mathbf{in}\, c \leftarrow c'; v \qquad c.\mathbf{wait}() \triangleq \mathbf{wait}\,(!\, c); \mathbf{free}\, c$$

Session channel example:

$$\begin{array}{l} \texttt{ses\_ref\_prog} \triangleq \\ \quad \mathbf{let}\,(c_1, c_2) = \mathbf{new}\,()\, \mathbf{in} \\ \quad \left(\begin{array}{l} \mathbf{let}\, \ell = \mathbf{ref}\, 40\, \mathbf{in} \\ c_1.\mathbf{send}(\ell); c_1.\mathbf{wait}(); \\ \mathbf{let}\, x = !\, \ell\, \mathbf{in}\, \mathbf{free}\, \ell; \\ \mathbf{assert}(x = 42) \end{array} \middle\| \begin{array}{l} \mathbf{let}\, \ell = c_2.\mathbf{recv}()\, \mathbf{in} \\ \ell \leftarrow (!\, \ell + 2); c_2.\mathbf{close}() \\ \\ \\ \end{array} \right) \end{array}$$

Goal: Verify this example and all its dependencies in Iris

Questions?

Basic concurrent separation logic and one-shot protocols

Tutorial Timeline

Part 1: 14:00 – 15:30

- ~~Introduction (10 min)~~
- ~~Layered implementation of session channels (10 min)~~
- Basic concurrent separation logic and one-shot protocols (30 min)
- **Break** (10 min)
- Dependent separation protocols (30 min)

Break (30 min)

Part 2: 16:00 – 17:30

- Iris invariants and ghost state (30 min)
- **Break** (10 min)
- Supervised Coq hacking (50 min)

Overview of Abstraction Layers

Layer	Reasoning principles / specifications
#1 Iris's HeapLang	Basic concurrent separation logic Iris invariants and ghost state
#2 One-shot channels	One-shot protocols
#3 Functional session channels	Dependent separation protocols
#4 Session channels	Dependent separation protocols

Hoare Triples

Hoare triples for partial functional correctness:

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

Precondition

Postcondition

Binder for return value

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 $Q[v/w]$

We often write $\{P\}\, e\, \{Q\} \triangleq \{P\}\, e\, \{w.\, w = () * Q\}$

Separation Logic [O'Hearn, Reynolds, Yang 2001]

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 [O'Hearn, Reynolds, Yang 2001]

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:

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

Sample of Separation Logic Rules

Heap manipulation:

Ht-alloc

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

Ht-load

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

Ht-store

$$\{\ell \mapsto v\}\ \ell \leftarrow w\ \{\ell \mapsto w\}$$

Ht-free

$$\{\ell \mapsto v\}\ \mathtt{free}\,\ell\ \{\mathsf{True}\}$$

Structural and general rules:

Ht-let

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

Ht-val

$$\{\mathsf{True}\}\ v\ \{w.\ w = v\}$$

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-assert

$$\frac{\{P\}\ e\ \{w.\ w = \mathtt{true} * Q\}}{\{P\}\ \mathtt{assert}(e)\ \{Q\}}$$

Simple Verification Example – Sequential Reference Program

$$
\begin{array}{l}
\mathtt{let}\ \ell = \mathtt{ref}\ \mathtt{None}\ \mathtt{in} \\
\ell \leftarrow \mathtt{Some}\ 42; \\
\mathtt{let}\ x = !\,\ell\ \mathtt{in} \\
\mathtt{free}\ \ell; \\
\mathtt{assert}\ (x = \mathtt{Some}\ 42)
\end{array}
$$

Simple Verification Example – Sequential Reference Program

$\{\mathsf{True}\}$
$\mathbf{let}\ \ell = \mathbf{ref}\ \mathbf{None}\ \mathbf{in}$
$\ell \leftarrow \mathbf{Some}\ 42;$
$\mathbf{let}\ x = !\,\ell\ \mathbf{in}$
$\mathbf{free}\ \ell;$
$\mathbf{assert}\ (x = \mathbf{Some}\ 42)$

Simple Verification Example – Sequential Reference Program

$\{\mathsf{True}\}$
$\texttt{let}\ \ell = \texttt{ref None in}$ $\quad$ // HT-LET, HT-ALLOC
$\{\ell \mapsto \texttt{None}\}$
$\ell \leftarrow \texttt{Some}\ 42;$
$\texttt{let}\ x = !\,\ell\ \texttt{in}$
$\texttt{free}\ \ell;$
$\texttt{assert}\,(x = \texttt{Some}\ 42)$

Simple Verification Example – Sequential Reference Program

$$
\begin{array}{ll}
\{\mathsf{True}\} & \\
\mathtt{let}\ \ell = \mathtt{ref}\ \mathtt{None}\ \mathtt{in} & //\ \textsc{Ht-let}, \textsc{Ht-alloc} \\
\{\ell \mapsto \mathtt{None}\} & \\
\ell \leftarrow \mathtt{Some}\ 42; & //\ \textsc{Ht-seq}, \textsc{Ht-store} \\
\{\ell \mapsto \mathtt{Some}\ 42\} & \\
\mathtt{let}\ x = !\,\ell\ \mathtt{in} & \\
\mathtt{free}\ \ell; & \\
\mathtt{assert}\ (x = \mathtt{Some}\ 42) &
\end{array}
$$

Simple Verification Example – Sequential Reference Program

$$
\begin{array}{ll}
\{\mathsf{True}\} & \\
\mathtt{let}\ \ell = \mathtt{ref}\ \mathtt{None}\ \mathtt{in} & //\ \textsc{Ht-let},\ \textsc{Ht-alloc} \\
\{\ell \mapsto \mathtt{None}\} & \\
\ell \leftarrow \mathtt{Some}\ 42; & //\ \textsc{Ht-seq},\ \textsc{Ht-store} \\
\{\ell \mapsto \mathtt{Some}\ 42\} & \\
\mathtt{let}\ x = !\,\ell\ \mathtt{in} & //\ \textsc{Ht-let},\ \textsc{Ht-load} \\
\{\ell \mapsto 42 * x = \mathtt{Some}\ 42\} & \\
\mathtt{free}\ \ell; & \\
\mathtt{assert}\,(x = \mathtt{Some}\ 42) &
\end{array}
$$

Simple Verification Example – Sequential Reference Program

$\{\mathsf{True}\}$
$\mathtt{let}\ \ell = \mathtt{ref}\ \mathtt{None}\ \mathtt{in}$ // HT-LET, HT-ALLOC
$\{\ell \mapsto \mathtt{None}\}$
$\ell \leftarrow \mathtt{Some}\ 42;$ // HT-SEQ, HT-STORE
$\{\ell \mapsto \mathtt{Some}\ 42\}$
$\mathtt{let}\ x = !\,\ell\ \mathtt{in}$ // HT-LET, HT-LOAD
$\{\ell \mapsto 42 * x = \mathtt{Some}\ 42\}$
$\mathtt{free}\ \ell;$ // HT-SEQ, HT-FREE
$\{x = \mathtt{Some}\ 42\}$
$\mathtt{assert}\ (x = \mathtt{Some}\ 42)$

Simple Verification Example – Sequential Reference Program

$$
\begin{array}{ll}
\{\mathsf{True}\} & \\
\texttt{let}\, \ell = \texttt{ref None in} & //\ \textsc{Ht-let}, \textsc{Ht-alloc} \\
\{\ell \mapsto \texttt{None}\} & \\
\ell \leftarrow \texttt{Some}\, 42; & //\ \textsc{Ht-seq}, \textsc{Ht-store} \\
\{\ell \mapsto \texttt{Some}\, 42\} & \\
\texttt{let}\, x = !\,\ell\ \texttt{in} & //\ \textsc{Ht-let}, \textsc{Ht-load} \\
\{\ell \mapsto 42 * x = \texttt{Some}\, 42\} & \\
\texttt{free}\, \ell; & //\ \textsc{Ht-seq}, \textsc{Ht-free} \\
\{x = \texttt{Some}\, 42\} & \\
\texttt{assert}\, (x = \texttt{Some}\, 42) & //\ \textsc{Ht-assert} \\
\{\mathsf{True}\} &
\end{array}
$$

One-Shot Channel Specifications

Channel ownership $c \rightarrowtail p$

- Provides exclusive permission to use the channel c according to the protocol p

Protocols and duality:

$$\textbf{Protocols:} \quad p ::= (\mathsf{Send}, \Phi) \mid (\mathsf{Recv}, \Phi) \quad \text{where} \quad \Phi : \mathsf{Val} \to \mathsf{Prop}$$

$$\textbf{Duality:} \quad \overline{(\mathsf{Send}, \Phi)} \triangleq (\mathsf{Recv}, \Phi) \qquad \overline{(\mathsf{Recv}, \Phi)} \triangleq (\mathsf{Send}, \Phi)$$

One-Shot Channel Specifications

Channel ownership $c \rightarrowtail p$

- Provides exclusive permission to use the channel c according to the protocol p

Protocols and duality:

$$\textbf{Protocols:} \quad p ::= (\mathsf{Send}, \Phi) \mid (\mathsf{Recv}, \Phi) \quad \text{where} \quad \Phi : \mathsf{Val} \to \mathsf{Prop}$$

$$\textbf{Duality:} \quad \overline{(\mathsf{Send}, \Phi)} \triangleq (\mathsf{Recv}, \Phi) \qquad \overline{(\mathsf{Recv}, \Phi)} \triangleq (\mathsf{Send}, \Phi)$$

One-shot channel specifications:

Ht-new

$$\{\mathsf{True}\}\ \mathtt{new1}\ ()\ \{w.\ \exists c.\ w = c * c \rightarrowtail p * c \rightarrowtail \overline{p}\}$$

Ht-send

$$\{c \rightarrowtail (\mathsf{Send}, \Phi) * \Phi\ v\}\ \mathtt{send1}\ c\ v\ \{\mathsf{True}\}$$

Ht-recv

$$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}\ \mathtt{recv1}\ c\ \{w.\ \Phi\ w\}$$

Concurrency Specifications

Parallel composition rule:

$$\textsc{Ht-par}\quad \frac{\{P_1\}\, e_1\, \{w_1.Q_1\} \qquad \{P_2\}\, e_2\, \{w_2.Q_2\}}{\{P_1 * P_2\}\, (e_1 \parallel e_2)\, \{w.\, \exists w_1, w_2.\, w = (w_1, w_2) * Q_1 * Q_2\}}$$

One-Shot Channel Verification Examples – Basics

$$
\begin{array}{l}
\mathtt{let}\ c = \mathtt{new1}\,()\ \mathtt{in} \\
\left(\begin{array}{l} \mathtt{send1}\ c\ 42 \end{array} \middle\| \begin{array}{l} \mathtt{let}\ x = \mathtt{recv1}\ c\ \mathtt{in} \\ \mathtt{assert}(x = 42) \end{array}\right)
\end{array}
$$

One-Shot Channel Verification Examples – Basics

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\; c = \mathtt{new1}\,()\;\mathtt{in} \\
\left(\begin{array}{l} \mathtt{send1}\; c\; 42 \end{array} \;\middle\|\; \begin{array}{l} \mathtt{let}\; x = \mathtt{recv1}\; c\; \mathtt{in} \\ \mathtt{assert}(x = 42) \end{array}\right)
\end{array}
$$

One-Shot Channel Verification Examples – Basics

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

$$\mathtt{let}\, c = \mathtt{new1}\,()\,\mathtt{in}$$

$$\left(\mathtt{send1}\, c\, 42 \;\middle\|\; \begin{array}{l}\mathtt{let}\, x = \mathtt{recv1}\, c\, \mathtt{in} \\ \mathtt{assert}(x = 42)\end{array}\right)$$

One-shot protocol:

$$\mathtt{prot} \triangleq (\mathsf{Send}, \lambda w.\, w = 42)$$

One-Shot Channel Verification Examples – Basics

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\,()\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{prot} * c \rightarrowtail \overline{\mathtt{prot}}\} \\
\left(\begin{array}{l} \mathbf{send1}\ c\ 42 \end{array} \middle\| \begin{array}{l} \mathbf{let}\ x = \mathbf{recv1}\ c\ \mathbf{in} \\ \mathbf{assert}(x = 42) \end{array}\right)
\end{array}
$$

One-shot protocol:

$$\mathtt{prot} \triangleq (\mathsf{Send}, \lambda w.\, w = 42)$$

One-Shot Channel Verification Examples – Basics

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\; c = \mathbf{new1}\,()\;\mathbf{in} \\
\{c \rightarrowtail \mathtt{prot} * c \rightarrowtail \overline{\mathtt{prot}}\} \\
\left(\begin{array}{l} \{c \rightarrowtail \mathtt{prot}\} \\ \mathbf{send1}\; c\; 42 \\ \; \end{array} \;\middle\|\; \begin{array}{l} \{c \rightarrowtail \overline{\mathtt{prot}}\} \\ \mathbf{let}\; x = \mathbf{recv1}\; c\;\mathbf{in} \\ \mathbf{assert}(x = 42) \end{array}\right)
\end{array}
$$

One-shot protocol:

$$\mathtt{prot} \triangleq (\mathsf{Send}, \lambda w.\, w = 42)$$

One-Shot Channel Verification Examples – Basics

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\; c = \mathtt{new1}\,()\; \mathtt{in} \\
\{c \rightarrowtail \mathtt{prot} * c \rightarrowtail \overline{\mathtt{prot}}\} \\
\left(\begin{array}{l}
\{c \rightarrowtail \mathtt{prot}\} \\
\mathtt{send1}\; c\; 42 \\
\{\mathsf{True}\}
\end{array}
\;\middle\|\;
\begin{array}{l}
\{c \rightarrowtail \overline{\mathtt{prot}}\} \\
\mathtt{let}\; x = \mathtt{recv1}\; c\; \mathtt{in} \\
\mathtt{assert}(x = 42)
\end{array}\right)
\end{array}
$$

One-shot protocol:

$$\mathtt{prot} \triangleq (\mathsf{Send}, \lambda w.\, w = 42)$$

One-Shot Channel Verification Examples – Basics

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\,()\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{prot} * c \rightarrowtail \overline{\mathtt{prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{prot}\} & \{c \rightarrowtail \overline{\mathtt{prot}}\} \\
\mathbf{send1}\ c\ 42 & \mathbf{let}\ x = \mathbf{recv1}\ c\ \mathbf{in} \\
\{\mathsf{True}\} & \{x = 42\} \\
 & \mathbf{assert}(x = 42)
\end{array}\right)
\end{array}
$$

One-shot protocol:

$$\mathtt{prot} \triangleq (\mathsf{Send}, \lambda w.\, w = 42)$$

One-Shot Channel Verification Examples – Basics

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\,()\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{prot} * c \rightarrowtail \overline{\mathtt{prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{prot}\} & \{c \rightarrowtail \overline{\mathtt{prot}}\} \\
\mathbf{send1}\ c\ 42 & \mathbf{let}\ x = \mathbf{recv1}\ c\ \mathbf{in} \\
\{\mathsf{True}\} & \{x = 42\} \\
 & \mathbf{assert}(x = 42) \\
 & \{\mathsf{True}\}
\end{array}\right)
\end{array}
$$

One-shot protocol:

$$\mathtt{prot} \triangleq (\mathsf{Send}, \lambda w.\, w = 42)$$

One-Shot Channel Verification Examples – Basics

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\, c = \mathtt{new1}\,()\,\mathtt{in} \\
\{c \rightarrowtail \mathtt{prot} * c \rightarrowtail \overline{\mathtt{prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{prot}\} & \{c \rightarrowtail \overline{\mathtt{prot}}\} \\
\mathtt{send1}\, c\, 42 & \mathtt{let}\, x = \mathtt{recv1}\, c\, \mathtt{in} \\
\{\mathsf{True}\} & \{x = 42\} \\
 & \mathtt{assert}(x = 42) \\
 & \{\mathsf{True}\}
\end{array}\right) \\
\{\mathsf{True}\}
\end{array}
$$

One-shot protocol:

$$\mathtt{prot} \triangleq (\mathsf{Send}, \lambda w.\, w = 42)$$

One-Shot Channel Verification Examples – References

$$
\begin{array}{l}
\mathbf{let}\ c = \mathbf{new1}\,()\ \mathbf{in} \\
\left(\begin{array}{l}
\mathbf{let}\ \ell = \mathbf{ref}\ 42\ \mathbf{in} \\
\mathbf{send1}\ c\ \ell \\

\end{array}
\;\middle\|\;
\begin{array}{l}
\mathbf{let}\ \ell = \mathbf{recv1}\ c\ \mathbf{in} \\
\mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\ \ell; \\
\mathbf{assert}(x = 42)
\end{array}\right)
\end{array}
$$

One-Shot Channel Verification Examples – References

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\; c = \mathbf{new1}\,()\;\mathbf{in} \\
\left(\begin{array}{l} \mathbf{let}\;\ell = \mathbf{ref}\;42\;\mathbf{in} \\ \mathbf{send1}\;c\;\ell \\ \end{array} \middle\| \begin{array}{l} \mathbf{let}\;\ell = \mathbf{recv1}\;c\;\mathbf{in} \\ \mathbf{let}\;x = !\,\ell\;\mathbf{in}\;\mathbf{free}\;\ell; \\ \mathbf{assert}(x = 42) \end{array}\right)
\end{array}
$$

One-Shot Channel Verification Examples – References

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\, c = \mathtt{new1}\,()\, \mathtt{in} \\
\left(\begin{array}{l} \mathtt{let}\, \ell = \mathtt{ref}\, 42\, \mathtt{in} \\ \mathtt{send1}\, c\, \ell \\ \end{array} \middle\| \begin{array}{l} \mathtt{let}\, \ell = \mathtt{recv1}\, c\, \mathtt{in} \\ \mathtt{let}\, x = !\, \ell\, \mathtt{in}\, \mathtt{free}\, \ell; \\ \mathtt{assert}(x = 42) \end{array}\right)
\end{array}
$$

One-shot reference protocol:

$$\mathtt{ref\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}).\, w = \ell * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – References

$$\{\mathsf{True}\}$$
$$\mathbf{let}\,c = \mathbf{new1}\,()\,\mathbf{in}$$
$$\{c \rightarrowtail \mathtt{ref\_prot} * c \rightarrowtail \overline{\mathtt{ref\_prot}}\}$$
$$\left(\begin{array}{l} \mathbf{let}\,\ell = \mathbf{ref}\,42\,\mathbf{in} \\ \mathbf{send1}\,c\,\ell \\ \end{array} \;\middle\|\; \begin{array}{l} \mathbf{let}\,\ell = \mathbf{recv1}\,c\,\mathbf{in} \\ \mathbf{let}\,x = !\,\ell\,\mathbf{in}\ \mathbf{free}\,\ell; \\ \mathbf{assert}(x = 42) \end{array}\right)$$

One-shot reference protocol:

$$\mathtt{ref\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists(\ell : \mathsf{Loc}).\, w = \ell * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – References

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\, c = \mathbf{new1}\,()\,\mathbf{in} \\
\{c \rightarrowtail \mathtt{ref\_prot} * c \rightarrowtail \overline{\mathtt{ref\_prot}}\} \\
\left(\begin{array}{l}
\{c \rightarrowtail \mathtt{ref\_prot}\} \\
\mathbf{let}\, \ell = \mathbf{ref}\, 42\, \mathbf{in} \\
\mathbf{send1}\, c\, \ell \\
\\
\end{array}
\middle\|
\begin{array}{l}
\{c \rightarrowtail \overline{\mathtt{ref\_prot}}\} \\
\mathbf{let}\, \ell = \mathbf{recv1}\, c\, \mathbf{in} \\
\mathbf{let}\, x = !\, \ell\, \mathbf{in}\, \mathbf{free}\, \ell; \\
\mathbf{assert}(x = 42)
\end{array}\right)
\end{array}
$$

One-shot reference protocol:

$$\mathtt{ref\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists(\ell : \mathsf{Loc}).\, w = \ell * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – References

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\,()\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{ref\_prot} * c \rightarrowtail \overline{\mathtt{ref\_prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{ref\_prot}\} & \{c \rightarrowtail \overline{\mathtt{ref\_prot}}\} \\
\mathbf{let}\ \ell = \mathbf{ref}\,42\ \mathbf{in} & \mathbf{let}\ \ell = \mathbf{recv1}\,c\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{ref\_prot} * \ell \mapsto 42\} & \mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\,\ell; \\
\mathbf{send1}\,c\,\ell & \mathbf{assert}(x = 42)
\end{array}\right)
\end{array}
$$

One-shot reference protocol:

$$\mathtt{ref\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists(\ell : \mathsf{Loc}).\, w = \ell * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – References

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\, c = \mathbf{new1}\,()\,\mathbf{in} \\
\{c \rightarrowtail \mathtt{ref\_prot} * c \rightarrowtail \overline{\mathtt{ref\_prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{ref\_prot}\} & \{c \rightarrowtail \overline{\mathtt{ref\_prot}}\} \\
\mathbf{let}\, \ell = \mathbf{ref}\, 42\, \mathbf{in} & \mathbf{let}\, \ell = \mathbf{recv1}\, c\, \mathbf{in} \\
\{c \rightarrowtail \mathtt{ref\_prot} * \ell \mapsto 42\} & \mathbf{let}\, x = !\,\ell\, \mathbf{in}\ \mathbf{free}\, \ell; \\
\mathbf{send1}\, c\, \ell & \mathbf{assert}(x = 42) \\
\{\mathsf{True}\} &
\end{array}\right)
\end{array}
$$

One-shot reference protocol:

$$\mathtt{ref\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists(\ell : \mathsf{Loc}).\, w = \ell * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – References

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\, c = \mathbf{new1}\,()\,\mathbf{in} \\
\{c \rightarrowtail \texttt{ref\_prot} * c \rightarrowtail \overline{\texttt{ref\_prot}}\} \\
\left(\begin{array}{l}
\{c \rightarrowtail \texttt{ref\_prot}\} \\
\mathbf{let}\, \ell = \mathbf{ref}\, 42\, \mathbf{in} \\
\{c \rightarrowtail \texttt{ref\_prot} * \ell \mapsto 42\} \\
\mathbf{send1}\, c\, \ell \\
\{\mathsf{True}\}
\end{array}
\middle\|
\begin{array}{l}
\{c \rightarrowtail \overline{\texttt{ref\_prot}}\} \\
\mathbf{let}\, \ell = \mathbf{recv1}\, c\, \mathbf{in} \\
\{\ell \mapsto 42\} \\
\mathbf{let}\, x = !\,\ell\, \mathbf{in}\, \mathbf{free}\, \ell; \\
\mathbf{assert}(x = 42)
\end{array}\right)
\end{array}
$$

One-shot reference protocol:

$$\texttt{ref\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}).\, w = \ell * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – References

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\,()\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{ref\_prot} * c \rightarrowtail \overline{\mathtt{ref\_prot}}\} \\
\left(\begin{array}{l}
\{c \rightarrowtail \mathtt{ref\_prot}\} \\
\mathbf{let}\ \ell = \mathbf{ref}\,42\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{ref\_prot} * \ell \mapsto 42\} \\
\mathbf{send1}\,c\,\ell \\
\{\mathsf{True}\} \\
\\
\end{array}
\ \middle\|\
\begin{array}{l}
\{c \rightarrowtail \overline{\mathtt{ref\_prot}}\} \\
\mathbf{let}\ \ell = \mathbf{recv1}\,c\ \mathbf{in} \\
\{\ell \mapsto 42\} \\
\mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\,\ell; \\
\{x = 42\} \\
\mathbf{assert}(x = 42)
\end{array}\right)
\end{array}
$$

One-shot reference protocol:

$$\mathtt{ref\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}).\, w = \ell * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – References

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\,()\ \mathbf{in} \\
\{c \rightarrowtail \texttt{ref\_prot} * c \rightarrowtail \overline{\texttt{ref\_prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \texttt{ref\_prot}\} & \{c \rightarrowtail \overline{\texttt{ref\_prot}}\} \\
\mathbf{let}\ \ell = \mathbf{ref}\,42\ \mathbf{in} & \mathbf{let}\ \ell = \mathbf{recv1}\,c\ \mathbf{in} \\
\{c \rightarrowtail \texttt{ref\_prot} * \ell \mapsto 42\} & \{\ell \mapsto 42\} \\
\mathbf{send1}\,c\,\ell & \mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\,\ell; \\
\{\mathsf{True}\} & \{x = 42\} \\
 & \mathbf{assert}(x = 42) \\
 & \{\mathsf{True}\}
\end{array}\right)
\end{array}
$$

One-shot reference protocol:

$$\texttt{ref\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists(\ell : \mathsf{Loc}).\, w = \ell * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – References

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\, c = \mathbf{new1}\,()\ \mathbf{in} \\
\{c \rightarrowtail \texttt{ref\_prot} * c \rightarrowtail \overline{\texttt{ref\_prot}}\} \\
\left(\begin{array}{l||l}
\{c \rightarrowtail \texttt{ref\_prot}\} & \{c \rightarrowtail \overline{\texttt{ref\_prot}}\} \\
\mathbf{let}\, \ell = \mathbf{ref}\, 42\ \mathbf{in} & \mathbf{let}\, \ell = \mathbf{recv1}\, c\ \mathbf{in} \\
\{c \rightarrowtail \texttt{ref\_prot} * \ell \mapsto 42\} & \{\ell \mapsto 42\} \\
\mathbf{send1}\, c\, \ell & \mathbf{let}\, x = {!}\,\ell\ \mathbf{in}\ \mathbf{free}\, \ell; \\
\{\mathsf{True}\} & \{x = 42\} \\
 & \mathbf{assert}(x = 42) \\
 & \{\mathsf{True}\}
\end{array}\right) \\
\{\mathsf{True}\}
\end{array}
$$

One-shot reference protocol:

$$\texttt{ref\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}).\, w = \ell * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – Higher-Order

$$
\begin{array}{l}
\mathbf{let}\, c = \mathbf{new1}\,()\,\mathbf{in} \\
\left(\begin{array}{l|l}
\mathbf{let}\, \ell = \mathbf{ref}\, 40\, \mathbf{in} & \mathbf{let}\, (\ell, c') = \mathbf{recv1}\, c\, \mathbf{in} \\
\mathbf{let}\, c' = \mathbf{new1}\,()\,\mathbf{in}\, \mathbf{send1}\, c\, (\ell, c'); & \ell \leftarrow (!\,\ell + 2); \\
\mathbf{recv1}\, c'; & \mathbf{send1}\, c'\, () \\
\mathbf{let}\, x = !\,\ell\, \mathbf{in}\, \mathbf{free}\, \ell; & \\
\mathbf{assert}(x = 42) &
\end{array}\right)
\end{array}
$$

One-Shot Channel Verification Examples – Higher-Order

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

$$
\begin{array}{l}
\mathbf{let}\; c = \mathbf{new1}\,()\;\mathbf{in} \\
\left(\begin{array}{l}
\mathbf{let}\;\ell = \mathbf{ref}\,40\;\mathbf{in} \\
\mathbf{let}\;c' = \mathbf{new1}\,()\;\mathbf{in}\;\mathbf{send1}\,c\,(\ell, c'); \\
\mathbf{recv1}\,c'; \\
\mathbf{let}\;x = !\,\ell\;\mathbf{in}\;\mathbf{free}\,\ell; \\
\mathbf{assert}(x = 42)
\end{array}
\;\middle|\middle|\;
\begin{array}{l}
\mathbf{let}\;(\ell, c') = \mathbf{recv1}\,c\;\mathbf{in} \\
\ell \leftarrow (!\,\ell + 2); \\
\mathbf{send1}\,c'\,() \\
\\
\\
\end{array}\right)
\end{array}
$$

One-Shot Channel Verification Examples – Higher-Order

$\{\mathsf{True}\}$

$$
\begin{array}{l}
\mathbf{let}\, c = \mathbf{new1}\,()\,\mathbf{in} \\
\left(\begin{array}{l}
\mathbf{let}\, \ell = \mathbf{ref}\, 40\, \mathbf{in} \\
\mathbf{let}\, c' = \mathbf{new1}\,()\,\mathbf{in}\, \mathbf{send1}\, c\, (\ell, c'); \\
\mathbf{recv1}\, c'; \\
\mathbf{let}\, x = !\,\ell\, \mathbf{in}\, \mathbf{free}\, \ell; \\
\mathbf{assert}(x = 42)
\end{array}
\middle\|
\begin{array}{l}
\mathbf{let}\, (\ell, c') = \mathbf{recv1}\, c\, \mathbf{in} \\
\ell \leftarrow (!\,\ell + 2); \\
\mathbf{send1}\, c'\, () \\
\\
\\
\end{array}\right)
\end{array}
$$

One-shot channel protocols:

$$
\begin{array}{l}
\mathtt{chan\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\ \ell}) \\
\mathtt{chan\_prot}'\ (\ell : \mathsf{Loc}) \triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)
\end{array}
$$

One-Shot Channel Verification Examples – Higher-Order

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\ ()\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\left(\begin{array}{l|l}
\mathbf{let}\ \ell = \mathbf{ref}\ 40\ \mathbf{in} & \mathbf{let}\ (\ell, c') = \mathbf{recv1}\ c\ \mathbf{in} \\
\mathbf{let}\ c' = \mathbf{new1}\ ()\ \mathbf{in}\ \mathbf{send1}\ c\ (\ell, c'); & \ell \leftarrow (!\,\ell + 2); \\
\mathbf{recv1}\ c'; & \mathbf{send1}\ c'\ () \\
\mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\ \ell; & \\
\mathbf{assert}(x = 42) &
\end{array}\right)
\end{array}
$$

One-shot channel protocols:

$$
\begin{array}{l}
\mathtt{chan\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\ \ell}) \\
\mathtt{chan\_prot}'\ (\ell : \mathsf{Loc}) \triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)
\end{array}
$$

One-Shot Channel Verification Examples – Higher-Order

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\; c = \mathbf{new1}\,()\;\mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{chan\_prot}\} & \{c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\mathbf{let}\; \ell = \mathbf{ref}\, 40\;\mathbf{in} & \mathbf{let}\,(\ell, c') = \mathbf{recv1}\, c\;\mathbf{in} \\
\mathbf{let}\; c' = \mathbf{new1}\,()\;\mathbf{in}\;\mathbf{send1}\, c\,(\ell, c'); & \ell \leftarrow (!\,\ell + 2); \\
\mathbf{recv1}\, c'; & \mathbf{send1}\, c'\,() \\
\mathbf{let}\; x = !\,\ell\;\mathbf{in}\;\mathbf{free}\,\ell; & \\
\mathbf{assert}(x = 42) &
\end{array}\right)
\end{array}
$$

One-shot channel protocols:

$$\mathtt{chan\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\;\ell})$$
$$\mathtt{chan\_prot}'\;(\ell : \mathsf{Loc}) \triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – Higher-Order

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\, c = \mathbf{new1}\,()\,\mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{chan\_prot}\} & \{c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\mathbf{let}\, \ell = \mathbf{ref}\, 40\, \mathbf{in} & \mathbf{let}\,(\ell, c') = \mathbf{recv1}\, c\, \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * \ell \mapsto 40\} & \ell \leftarrow (!\,\ell + 2); \\
\mathbf{let}\, c' = \mathbf{new1}\,()\,\mathbf{in}\, \mathbf{send1}\, c\,(\ell, c'); & \mathbf{send1}\, c'\,() \\
\mathbf{recv1}\, c'; & \\
\mathbf{let}\, x = !\,\ell\, \mathbf{in}\, \mathbf{free}\, \ell; & \\
\mathbf{assert}(x = 42) &
\end{array}\right)
\end{array}
$$

One-shot channel protocols:

$$
\begin{array}{l}
\mathtt{chan\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\ \ell}) \\
\mathtt{chan\_prot}'\ (\ell : \mathsf{Loc}) \triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)
\end{array}
$$

One-Shot Channel Verification Examples – Higher-Order

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\ ()\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{chan\_prot}\} & \{c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\mathbf{let}\ \ell = \mathbf{ref}\ 40\ \mathbf{in} & \mathbf{let}\ (\ell, c') = \mathbf{recv1}\ c\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * \ell \mapsto 40\} & \ell \leftarrow (!\,\ell + 2); \\
\mathbf{let}\ c' = \mathbf{new1}\ ()\ \mathbf{in}\ \mathbf{send1}\ c\ (\ell, c'); & \mathbf{send1}\ c'\ () \\
\{c' \rightarrowtail (\mathtt{chan\_prot}'\ \ell)\} & \\
\mathbf{recv1}\ c'; & \\
\mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\ \ell; & \\
\mathbf{assert}(x = 42) &
\end{array}\right)
\end{array}
$$

One-shot channel protocols:

$$
\begin{aligned}
\mathtt{chan\_prot} &\triangleq (\mathsf{Send}, \lambda w.\ \exists (\ell : \mathsf{Loc}), c'.\ w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\ \ell}) \\
\mathtt{chan\_prot}'\ (\ell : \mathsf{Loc}) &\triangleq (\mathsf{Recv}, \lambda w.\ w = () * \ell \mapsto 42)
\end{aligned}
$$

One-Shot Channel Verification Examples – Higher-Order

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\,()\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{chan\_prot}\} & \{c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\mathbf{let}\ \ell = \mathbf{ref}\, 40\ \mathbf{in} & \mathbf{let}\ (\ell, c') = \mathbf{recv1}\, c\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * \ell \mapsto 40\} & \ell \leftarrow (!\,\ell + 2); \\
\mathbf{let}\ c' = \mathbf{new1}\,()\ \mathbf{in}\ \mathbf{send1}\, c\,(\ell, c'); & \mathbf{send1}\, c'\,() \\
\{c' \rightarrowtail (\mathtt{chan\_prot}'\ \ell)\} & \\
\mathbf{recv1}\, c'; & \\
\{\ell \mapsto 42\} & \\
\mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\,\ell; & \\
\mathbf{assert}(x = 42) &
\end{array}\right)
\end{array}
$$

One-shot channel protocols:

$$
\begin{aligned}
\mathtt{chan\_prot} &\triangleq (\mathsf{Send}, \lambda w.\ \exists(\ell : \mathsf{Loc}), c'.\ w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\ \ell}) \\
\mathtt{chan\_prot}'\ (\ell : \mathsf{Loc}) &\triangleq (\mathsf{Recv}, \lambda w.\ w = () * \ell \mapsto 42)
\end{aligned}
$$

One-Shot Channel Verification Examples – Higher-Order

$$\{\mathsf{True}\}$$
$$\mathbf{let}\ c = \mathbf{new1}\ ()\ \mathbf{in}$$
$$\{c \rightarrowtail \mathtt{chan\_prot} * c \rightarrowtail \overline{\mathtt{chan\_prot}}\}$$

$$\left(\begin{array}{l||l}
\{c \rightarrowtail \mathtt{chan_prot}\} & \{c \rightarrowtail \overline{\mathtt{chan_prot}}\} \\
\mathbf{let}\ \ell = \mathbf{ref}\ 40\ \mathbf{in} & \mathbf{let}\ (\ell, c') = \mathbf{recv1}\ c\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan_prot} * \ell \mapsto 40\} & \ell \leftarrow (!\,\ell + 2); \\
\mathbf{let}\ c' = \mathbf{new1}\ ()\ \mathbf{in}\ \mathbf{send1}\ c\ (\ell, c'); & \mathbf{send1}\ c'\ () \\
\{c' \rightarrowtail (\mathtt{chan_prot}'\ \ell)\} & \\
\mathbf{recv1}\ c'; & \\
\{\ell \mapsto 42\} & \\
\mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\ \ell; & \\
\{x = 42\} & \\
\mathbf{assert}(x = 42) &
\end{array}\right)$$

One-shot channel protocols:

$$\mathtt{chan\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\ \ell})$$
$$\mathtt{chan\_prot}'\ (\ell : \mathsf{Loc}) \triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – Higher-Order

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\,()\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{chan\_prot}\} & \{c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\mathbf{let}\ \ell = \mathbf{ref}\,40\ \mathbf{in} & \mathbf{let}\ (\ell, c') = \mathbf{recv1}\,c\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * \ell \mapsto 40\} & \ell \leftarrow (!\,\ell + 2); \\
\mathbf{let}\ c' = \mathbf{new1}\,()\ \mathbf{in}\ \mathbf{send1}\,c\,(\ell, c'); & \mathbf{send1}\,c'\,() \\
\{c' \rightarrowtail (\mathtt{chan\_prot}'\ \ell)\} & \\
\mathbf{recv1}\,c'; & \\
\{\ell \mapsto 42\} & \\
\mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\,\ell; & \\
\{x = 42\} & \\
\mathbf{assert}(x = 42) & \\
\{\mathsf{True}\} &
\end{array}\right)
\end{array}
$$

One-shot channel protocols:

$$
\begin{aligned}
\mathtt{chan\_prot} &\triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\ \ell}) \\
\mathtt{chan\_prot}'\ (\ell : \mathsf{Loc}) &\triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)
\end{aligned}
$$

One-Shot Channel Verification Examples – Higher-Order

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\; c = \mathbf{new1}\,()\;\mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{chan\_prot}\} & \{c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\mathbf{let}\;\ell = \mathbf{ref}\,40\;\mathbf{in} & \mathbf{let}\,(\ell, c') = \mathbf{recv1}\,c\;\mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * \ell \mapsto 40\} & \{c' \rightarrowtail \overline{(\mathtt{chan\_prot}'\;\ell)} * \ell \mapsto 40\} \\
\mathbf{let}\;c' = \mathbf{new1}\,()\;\mathbf{in}\;\mathbf{send1}\,c\,(\ell, c'); & \ell \leftarrow (!\,\ell + 2); \\
\{c' \rightarrowtail (\mathtt{chan\_prot}'\;\ell)\} & \mathbf{send1}\,c'\,() \\
\mathbf{recv1}\,c'; & \\
\{\ell \mapsto 42\} & \\
\mathbf{let}\;x = !\,\ell\;\mathbf{in}\;\mathbf{free}\,\ell; & \\
\{x = 42\} & \\
\mathbf{assert}(x = 42) & \\
\{\mathsf{True}\} &
\end{array}\right)
\end{array}
$$

One-shot channel protocols:

$$
\begin{array}{l}
\mathtt{chan\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\;\ell}) \\
\mathtt{chan\_prot}'\;(\ell : \mathsf{Loc}) \triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)
\end{array}
$$

One-Shot Channel Verification Examples – Higher-Order

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\, c = \mathbf{new1}\,()\,\mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\left(\begin{array}{l}
\{c \rightarrowtail \mathtt{chan\_prot}\} \\
\mathbf{let}\, \ell = \mathbf{ref}\, 40\, \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * \ell \mapsto 40\} \\
\mathbf{let}\, c' = \mathbf{new1}\,()\,\mathbf{in}\, \mathbf{send1}\, c\, (\ell, c'); \\
\{c' \rightarrowtail (\mathtt{chan\_prot}'\ \ell)\} \\
\mathbf{recv1}\, c'; \\
\{\ell \mapsto 42\} \\
\mathbf{let}\, x = !\,\ell\, \mathbf{in}\, \mathbf{free}\, \ell; \\
\{x = 42\} \\
\mathbf{assert}(x = 42) \\
\{\mathsf{True}\}
\end{array}
\ \middle\|\
\begin{array}{l}
\{c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\mathbf{let}\, (\ell, c') = \mathbf{recv1}\, c\, \mathbf{in} \\
\{c' \rightarrowtail \overline{(\mathtt{chan\_prot}'\ \ell)} * \ell \mapsto 40\} \\
\ell \leftarrow (!\,\ell + 2); \\
\{c' \rightarrowtail \overline{(\mathtt{chan\_prot}'\ \ell)} * \ell \mapsto 42\} \\
\mathbf{send1}\, c'\, () \\
\\ \\ \\ \\ \\
\end{array}\right)
\end{array}
$$

One-shot channel protocols:

$$\mathtt{chan\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\ \ell})$$
$$\mathtt{chan\_prot}'\ (\ell : \mathsf{Loc}) \triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)$$

One-Shot Channel Verification Examples – Higher-Order

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\ ()\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{chan\_prot}\} & \{c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\mathbf{let}\ \ell = \mathbf{ref}\ 40\ \mathbf{in} & \mathbf{let}\ (\ell, c') = \mathbf{recv1}\ c\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * \ell \mapsto 40\} & \{c' \rightarrowtail \overline{(\mathtt{chan\_prot}'\ \ell)} * \ell \mapsto 40\} \\
\mathbf{let}\ c' = \mathbf{new1}\ ()\ \mathbf{in}\ \mathbf{send1}\ c\ (\ell, c'); & \ell \leftarrow (!\,\ell + 2); \\
\{c' \rightarrowtail (\mathtt{chan\_prot}'\ \ell)\} & \{c' \rightarrowtail \overline{(\mathtt{chan\_prot}'\ \ell)} * \ell \mapsto 42\} \\
\mathbf{recv1}\ c'; & \mathbf{send1}\ c'\ () \\
\{\ell \mapsto 42\} & \{\mathsf{True}\} \\
\mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\ \ell; & \\
\{x = 42\} & \\
\mathbf{assert}(x = 42) & \\
\{\mathsf{True}\} &
\end{array}\right)
\end{array}
$$

One-shot channel protocols:

$$
\begin{array}{l}
\mathtt{chan\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists(\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\ \ell}) \\
\mathtt{chan\_prot}'\ (\ell : \mathsf{Loc}) \triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)
\end{array}
$$

One-Shot Channel Verification Examples – Higher-Order

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\ c = \mathbf{new1}\ ()\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\left(\begin{array}{l||l}
\{c \rightarrowtail \mathtt{chan\_prot}\} & \{c \rightarrowtail \overline{\mathtt{chan\_prot}}\} \\
\mathbf{let}\ \ell = \mathbf{ref}\ 40\ \mathbf{in} & \mathbf{let}\ (\ell, c') = \mathbf{recv1}\ c\ \mathbf{in} \\
\{c \rightarrowtail \mathtt{chan\_prot} * \ell \mapsto 40\} & \{c' \rightarrowtail \overline{(\mathtt{chan\_prot}'\ \ell)} * \ell \mapsto 40\} \\
\mathbf{let}\ c' = \mathbf{new1}\ ()\ \mathbf{in}\ \mathbf{send1}\ c\ (\ell, c'); & \ell \leftarrow (!\,\ell + 2); \\
\{c' \rightarrowtail (\mathtt{chan\_prot}'\ \ell)\} & \{c' \rightarrowtail \overline{(\mathtt{chan\_prot}'\ \ell)} * \ell \mapsto 42\} \\
\mathbf{recv1}\ c'; & \mathbf{send1}\ c'\ () \\
\{\ell \mapsto 42\} & \{\mathsf{True}\} \\
\mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\ \ell; & \\
\{x = 42\} & \\
\mathbf{assert}(x = 42) & \\
\{\mathsf{True}\} &
\end{array}\right) \\
\{\mathsf{True}\}
\end{array}
$$

One-shot channel protocols:

$$
\begin{aligned}
&\mathtt{chan\_prot} \triangleq (\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\mathtt{chan\_prot}'\ \ell}) \\
&\mathtt{chan\_prot}'\ (\ell : \mathsf{Loc}) \triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)
\end{aligned}
$$

Questions?

Break (10 min!)

Dependent separation protocols

Tutorial Timeline

Part 1: 14:00 – 15:30

- ~~Introduction (10 min)~~
- ~~Layered implementation of session channels (10 min)~~
- ~~Basic concurrent separation logic and one-shot protocols (30 min)~~
- ~~**Break** (10 min)~~
- Dependent separation protocols (30 min)

Break (30 min)

Part 2: 16:00 – 17:30

- Iris invariants and ghost state (30 min)
- **Break** (10 min)
- Supervised Coq hacking (50 min)

Overview of Abstraction Layers

Layer	**Reasoning principles / specifications**
#1 Iris's HeapLang	Basic concurrent separation logic Iris invariants and ghost state
#2 One-shot channels	One-shot protocols
#3 Functional session channels	Dependent separation protocols
#4 Session channels	Dependent separation protocols

Implementation (inspired by Kobayashi et al., Dardha et al.):

$$\mathbf{new}_{\mathbf{fun}}\,() \triangleq \mathbf{new1}\,()$$

$$\mathbf{send}\,c\,v \triangleq \mathbf{let}\,c' = \mathbf{new1}\,()\,\mathbf{in}\,\mathbf{send1}\,c\,(v, c');\; c' \qquad \mathbf{close}\,c \triangleq \mathbf{send1}\,c\,()$$

$$\mathbf{recv}\,c \triangleq \mathbf{recv1}\,c \qquad \mathbf{wait}\,c \triangleq \mathbf{recv1}\,c$$

Example program:

$$\begin{array}{l}
\mathtt{ses_fun_ref_prog} \triangleq \\
\quad \mathbf{let}\,c = \mathbf{new}_{\mathbf{fun}}\,()\,\mathbf{in} \\
\quad \left(\begin{array}{l|l}
\mathbf{let}\,\ell = \mathbf{ref}\,40\,\mathbf{in} & \mathbf{let}\,(\ell, c') = \mathbf{recv}\,c\,\mathbf{in} \\
\mathbf{let}\,c' = \mathbf{send}\,c\,\ell\,\mathbf{in} & \ell \leftarrow (!\,\ell + 2); \mathbf{close}\,c' \\
\mathbf{wait}\,c'; \mathbf{let}\,x = !\,\ell\,\mathbf{in}\,\mathbf{free}\,\ell; & \\
\mathbf{assert}(x = 42) &
\end{array}\right)
\end{array}$$

Functional Session Channel Specifications?

Implementation (inspired by Kobayashi et al., Dardha et al.):

$$\mathbf{new}_{\mathbf{fun}}\,() \triangleq \mathtt{new1}\,()$$

$$\mathtt{send}\,c\,v \triangleq \mathtt{let}\,c' = \mathtt{new1}\,()\,\mathtt{in}\,\mathtt{send1}\,c\,(v, c');\; c' \qquad \mathtt{close}\,c \triangleq \mathtt{send1}\,c\,()$$

$$\mathtt{recv}\,c \triangleq \mathtt{recv1}\,c \qquad \mathtt{wait}\,c \triangleq \mathtt{recv1}\,c$$

Specifications:

$$\{\mathsf{True}\}\ \mathtt{new}_{\mathbf{fun}}\,()\ \{w.\,\exists c.\, w = c * c \rightarrowtail p * c \rightarrowtail \overline{p}\}$$

$$\{c \rightarrowtail ??? * ???\}\ \mathtt{send}\,c\,v\ \{w.\,???\}$$

$$\{c \rightarrowtail ???\}\ \mathtt{recv}\,c\ \{w.\,???\}$$

$$\{c \rightarrowtail ??? * ???\}\ \mathtt{close}\,c\ \{w.\,???\}$$

$$\{c \rightarrowtail ???\}\ \mathtt{wait}\,c\ \{w.\,???\}$$

Dependent Separation Protocols

$$\texttt{chan\_prot} \triangleq$$
$$(\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\texttt{chan\_prot}'\ \ell})$$

Dependent Separation Protocols

$$\texttt{send\_prot} \triangleq$$
$$(\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\texttt{chan\_prot}'\ \ell})$$

Dependent Separation Protocols

$$\texttt{send\_prot} \triangleq$$
$$(\mathsf{Send}, \lambda w.\, \exists(\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{\texttt{chan\_prot}'\ \ell})$$

Dependent Separation Protocols

$$\texttt{send\_prot}\ (p : \mathsf{Loc} \to \mathsf{iProto}) \triangleq$$
$$(\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (\ell, c') * \ell \mapsto 40 * c' \rightarrowtail \overline{p\ \ell})$$

Dependent Separation Protocols

$$\texttt{send\_prot}\ (P : \mathsf{Loc} \to \mathsf{iProp})\ (p : \mathsf{Loc} \to \mathsf{iProto}) \triangleq$$
$$(\mathsf{Send}, \lambda w.\ \exists (\ell : \mathsf{Loc}), c'.\ w = (\ell, c') * P\ \ell * c' \rightarrowtail \overline{p\ \ell})$$

Dependent Separation Protocols

$$\texttt{send\_prot}\ (v : \mathsf{Loc} \to \mathsf{Val})\ (P : \mathsf{Loc} \to \mathsf{iProp})\ (p : \mathsf{Loc} \to \mathsf{iProto}) \triangleq$$
$$(\mathsf{Send}, \lambda w.\, \exists (\ell : \mathsf{Loc}), c'.\, w = (v\ \ell, c') * P\ x * c' \rightarrowtail \overline{p\ x})$$

Dependent Separation Protocols

$$\texttt{send\_prot}\ (\tau : \mathsf{Type})\ (v : \tau \to \mathsf{Val})\ (P : \tau \to \mathsf{iProp})\ (p : \tau \to \mathsf{iProto}) \triangleq$$
$$(\mathsf{Send}, \lambda w.\, \exists (x : \tau), c'.\, w = (v\ x, c') * P\ x * c' \rightarrowtail \overline{p\ x})$$

Dependent Separation Protocols

$$
\begin{aligned}
&\mathbf{!}\,(x : \tau)\,\langle v \rangle\{P\}.\,p \triangleq \\
&\quad (\mathsf{Send}, \lambda w.\, \exists (x : \tau), c'.\, w = (v\ x, c') * P\ x * c' \rightarrowtail \overline{p\ x})
\end{aligned}
$$

Dependent Separation Protocols

$$!\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\,\exists(x : \tau), c'.\,w = (v\,x, c') * P\,x * c' \rightarrowtail \overline{p\,x})$$

Dependent Separation Protocols

$$\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\,\exists(x : \tau), c'.\,w = (v\;x, c') * P\;x * c' \rightarrowtail \overline{p\;x})$$
$$\mathbf{?}(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq \overline{\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,\overline{p}}$$

Dependent Separation Protocols

$$\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\,\exists(x : \tau), c'.\,w = (v\,x, c') * P\,x * c' \rightarrowtail \overline{p\,x})$$
$$\mathbf{?}(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Recv}, \lambda w.\,\exists(x : \tau), c'.\,w = (v\,x, c') * P\,x * c' \rightarrowtail \overline{\overline{p\,x}})$$

Dependent Separation Protocols

$$
\begin{aligned}
&\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\, \exists (x : \tau), c'.\, w = (v\,x, c') * P\,x * c' \rightarrowtail \overline{p\,x})\\
&\mathbf{?}(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Recv}, \lambda w.\, \exists (x : \tau), c'.\, w = (v\,x, c') * P\,x * c' \rightarrowtail p\,x)
\end{aligned}
$$

Dependent Separation Protocols

$$\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\,\exists (x : \tau), c'.\, w = (v\;x, c') * P\;x * c' \rightarrowtail \overline{p\;x})$$

$$\mathbf{?}(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Recv}, \lambda w.\,\exists (x : \tau), c'.\, w = (v\;x, c') * P\;x * c' \rightarrowtail p\;x)$$

$$\texttt{chan\_prot}'\,(\ell : \mathsf{Loc}) \triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)$$

Dependent Separation Protocols

$$
\begin{aligned}
&\textbf{!}\,(x:\tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\,\exists(x:\tau), c'.\, w = (v\;x, c') * P\;x * c' \rightarrowtail \overline{p\;x})\\
&\textbf{?}(x:\tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Recv}, \lambda w.\,\exists(x:\tau), c'.\, w = (v\;x, c') * P\;x * c' \rightarrowtail p\;x)
\end{aligned}
$$

$$
\texttt{close\_prot}\,(\ell : \mathsf{Loc}) \triangleq (\mathsf{Recv}, \lambda w.\, w = () * \ell \mapsto 42)
$$

Dependent Separation Protocols

$$\mathbf{!}\,(x:\tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\, \exists (x:\tau), c'.\, w = (v\ x, c') * P\ x * c' \rightarrowtail \overline{p\ x})$$

$$\mathbf{?}(x:\tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Recv}, \lambda w.\, \exists (x:\tau), c'.\, w = (v\ x, c') * P\ x * c' \rightarrowtail p\ x)$$

$$\texttt{close\_prot}\ (\ell : \mathsf{Loc}) \triangleq (\mathsf{Send}, \lambda w.\, w = () * \ell \mapsto 42)$$

Dependent Separation Protocols

$$\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathrm{Send}, \lambda w.\, \exists (x:\tau), c'.\, w = (v\; x, c') * P\; x * c' \rightarrowtail \overline{p\; x})$$

$$\mathbf{?}(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathrm{Recv}, \lambda w.\, \exists (x:\tau), c'.\, w = (v\; x, c') * P\; x * c' \rightarrowtail p\; x)$$

$$\texttt{close\_prot}\;(P : \mathrm{iProp}) \triangleq (\mathrm{Send}, \lambda w.\, w = () * P)$$

Dependent Separation Protocols

$$\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathrm{Send}, \lambda w.\,\exists (x:\tau), c'.\, w = (v\,x, c') * P\,x * c' \rightarrowtail \overline{p\,x})$$

$$\mathbf{?}(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathrm{Recv}, \lambda w.\,\exists (x:\tau), c'.\, w = (v\,x, c') * P\,x * c' \rightarrowtail p\,x)$$

$$\mathbf{!\,end}\{P\} \triangleq (\mathrm{Send}, \lambda w.\, w = () * P)$$

Dependent Separation Protocols

$$\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\, \exists (x : \tau), c'.\, w = (v\,x, c') * P\,x * c' \rightarrowtail \overline{p\,x})$$
$$\mathbf{?}(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Recv}, \lambda w.\, \exists (x : \tau), c'.\, w = (v\,x, c') * P\,x * c' \rightarrowtail p\,x)$$

$$\mathbf{!\,end}\{P\} \triangleq (\mathsf{Send}, \lambda w.\, w = () * P)$$
$$\mathbf{?end}\{P\} \triangleq \overline{\mathbf{!\,end}\{P\}}$$

Dependent Separation Protocols

$$\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\, \exists (x : \tau), c'.\, w = (v\; x, c') * P\; x * c' \rightarrowtail \overline{p\; x})$$
$$\mathbf{?}(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Recv}, \lambda w.\, \exists (x : \tau), c'.\, w = (v\; x, c') * P\; x * c' \rightarrowtail p\; x)$$

$$\mathbf{!\,end}\{P\} \triangleq (\mathsf{Send}, \lambda w.\, w = () * P)$$
$$\mathbf{?end}\{P\} \triangleq (\mathsf{Recv}, \lambda w.\, w = () * P)$$

Dependent Separation Protocols

$$\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\,\exists(x : \tau), c'.\,w = (v\;x, c') * P\;x * c' \rightarrowtail \overline{p\;x})$$

$$\mathbf{?}(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Recv}, \lambda w.\,\exists(x : \tau), c'.\,w = (v\;x, c') * P\;x * c' \rightarrowtail p\;x)$$

$$\mathbf{!\,end}\{P\} \triangleq (\mathsf{Send}, \lambda w.\,w = () * P)$$

$$\mathbf{?end}\{P\} \triangleq (\mathsf{Recv}, \lambda w.\,w = () * P)$$

$$\texttt{chan\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell\rangle\{\ell \mapsto 40\}.\,\mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channels Specifications!

Dependent session protocols:

$$
\begin{aligned}
\mathbf{!}\,(x:\tau)\,\langle v\rangle\{P\}.\,p &\triangleq (\mathsf{Send}, \lambda w.\, \exists (x:\tau), c'.\, w = (v\,x, c') * (P\,x) * c' \rightarrowtail \overline{p\,x}) \\
\mathbf{?}(x:\tau)\,\langle v\rangle\{P\}.\,p &\triangleq (\mathsf{Recv}, \lambda w.\, \exists (x:\tau), c'.\, w = (v\,x, c') * (P\,x) * c' \rightarrowtail p\,x) \\
\mathbf{!\,end}\{P\} &\triangleq (\mathsf{Send}, \lambda w.\, w = () * P) \\
\mathbf{?end}\{P\} &\triangleq (\mathsf{Recv}, \lambda w.\, w = () * P)
\end{aligned}
$$

Functional session channel specifications:

$$
\begin{aligned}
\{\mathsf{True}\}\ &\mathtt{new_{fun}}\,()\ \{w.\, \exists c.\, w = c * c \rightarrowtail p * c \rightarrowtail \overline{p}\} \\
\{c \rightarrowtail (\mathbf{!}\,(x:\tau)\,\langle v\rangle\{P\}.\,p) * P\,t\}\ &\mathtt{send}\,c\,(v\,t)\ \{w.\, \exists c'.\, w = c' * c' \rightarrowtail p\,t\} \\
\{c \rightarrowtail (\mathbf{?}\,(x:\tau)\,\langle v\rangle\{P\}.\,p)\}\ &\mathtt{recv}\,c\ \{w.\, \exists (x:\tau), c'.\, w = (v\,x, c') * P\,x * c' \rightarrowtail p\,x\} \\
\{c \rightarrowtail \mathbf{!\,end}\{P\} * P\}\ &\mathtt{close}\,c\ \{\mathsf{True}\} \\
\{c \rightarrowtail \mathbf{?end}\{P\}\}\ &\mathtt{wait}\,c\ \{P\}
\end{aligned}
$$

Proof of Send Specification

$$
\begin{array}{l}
\texttt{let}\ c' = \texttt{new1}\ ()\ \texttt{in} \\
\texttt{send1}\ c\ (v\ t, c'); \\
c'
\end{array}
$$

Proof of Send Specification

$$\{c \rightarrowtail \mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \quad * \quad P\;t\}$$

$$\begin{array}{l}\texttt{let}\;c' = \texttt{new1}\,()\;\texttt{in}\\ \texttt{send1}\;c\;(v\;t, c');\\ c'\end{array}$$

Proof of Send Specification

$$\{c \rightarrowtail \mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \quad * \quad P\;t\}$$
$$\texttt{let}\; c' = \texttt{new1}\,()\;\texttt{in}$$
$$\texttt{send1}\; c\,(v\;t, c');$$
$$c'$$

Send protocol:

$$\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\,\exists (x : \tau), c'.\, w = (v\;x, c') * (P\;x) * c' \rightarrowtail \overline{p\;x})$$

Proof of Send Specification

$\{c \rightarrowtail \mathbf{!}\,(x : \tau)\,\langle v \rangle \{P\}.\, p \quad * \quad P\ t\}$
$\mathtt{let}\ c' = \mathtt{new1}\,()\ \mathtt{in}$
$\{c \rightarrowtail \mathbf{!}\,(x : \tau)\,\langle v \rangle \{P\}.\, p \quad * \quad P\ t \quad * \quad c' \rightarrowtail p\ t \quad * \quad c' \rightarrowtail \overline{p\ t}\}$
$\mathtt{send1}\ c\ (v\ t, c');$
c'

Send protocol:

$$\mathbf{!}\,(x : \tau)\,\langle v \rangle \{P\}.\, p \triangleq (\mathsf{Send}, \lambda w.\, \exists (x : \tau), c'.\, w = (v\ x, c') * (P\ x) * c' \rightarrowtail \overline{p\ x})$$

Proof of Send Specification

$$
\begin{array}{l}
\{c \rightarrowtail \mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \quad * \quad P\,t\} \\
\texttt{let}\; c' = \texttt{new1}\,()\;\texttt{in} \\
\{c \rightarrowtail \mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \quad * \quad P\,t \quad * \quad c' \rightarrowtail p\,t \quad * \quad c' \rightarrowtail \overline{p\,t}\} \\
\texttt{send1}\; c\,(v\,t, c'); \\
\{c' \rightarrowtail p\,t\} \\
c'
\end{array}
$$

Send protocol:

$$
\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\, \exists (x : \tau), c'.\, w = (v\,x, c') * (P\,x) * c' \rightarrowtail \overline{p\,x})
$$

Proof of Send Specification

$$
\begin{array}{l}
\{c \rightarrowtail \mathbf{!}\,(x:\tau)\,\langle v\rangle\{P\}.\,p \quad * \quad P\,t\} \\
\mathtt{let}\; c' = \mathtt{new1}\,()\;\mathtt{in} \\
\{c \rightarrowtail \mathbf{!}\,(x:\tau)\,\langle v\rangle\{P\}.\,p \quad * \quad P\,t \quad * \quad c' \rightarrowtail p\,t \quad * \quad c' \rightarrowtail \overline{p\,t}\} \\
\mathtt{send1}\,c\,(v\,t, c'); \\
\{c' \rightarrowtail p\,t\} \\
c' \\
\{w.\,\exists c'.\,w = c' * c' \rightarrowtail p\,t\}
\end{array}
$$

Send protocol:

$$\mathbf{!}\,(x:\tau)\,\langle v\rangle\{P\}.\,p \triangleq (\mathsf{Send}, \lambda w.\,\exists(x:\tau), c'.\,w = (v\,x, c') * (P\,x) * c' \rightarrowtail \overline{p\,x})$$

Functional Session Channel Example

$$
\begin{array}{l}
\mathbf{let}\, c = \mathbf{new_{fun}}\,()\,\mathbf{in} \\
\left(\begin{array}{l}
\mathbf{let}\,\ell = \mathbf{ref}\,40\,\mathbf{in} \\
\mathbf{let}\,c' = \mathbf{send}\,c\,\ell\,\mathbf{in} \\
\mathbf{wait}\,c'; \\
\mathbf{let}\,x = !\,\ell\,\mathbf{in}\ \mathbf{free}\,\ell; \\
\mathbf{assert}(x = 42)
\end{array}
\;\middle\|\;
\begin{array}{l}
\mathbf{let}\,(\ell, c') = \mathbf{recv}\,c\,\mathbf{in} \\
\ell \leftarrow (!\,\ell + 2); \\
\mathbf{close}\,c' \\
\\
\\
\end{array}\right)
\end{array}
$$

Functional Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\; c = \mathtt{new}_{\mathtt{fun}}\,()\;\mathtt{in} \\
\left(\begin{array}{l}
\mathtt{let}\;\ell = \mathtt{ref}\,40\;\mathtt{in} \\
\mathtt{let}\;c' = \mathtt{send}\,c\,\ell\;\mathtt{in} \\
\mathtt{wait}\;c'; \\
\mathtt{let}\;x = !\,\ell\;\mathtt{in}\;\mathtt{free}\,\ell; \\
\mathtt{assert}(x = 42)
\end{array}
\;\middle\|\;
\begin{array}{l}
\mathtt{let}\;(\ell, c') = \mathtt{recv}\,c\;\mathtt{in} \\
\ell \leftarrow (!\,\ell + 2); \\
\mathtt{close}\;c' \\
\\
\\
\end{array}\right)
\end{array}
$$

Functional Session Channel Example

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

$$\begin{array}{l}
\mathtt{let}\, c = \mathtt{new_{fun}}\,()\,\mathtt{in} \\
\left(\begin{array}{l}
\mathtt{let}\,\ell = \mathtt{ref}\,40\,\mathtt{in} \\
\mathtt{let}\,c' = \mathtt{send}\,c\,\ell\,\mathtt{in} \\
\mathtt{wait}\,c'; \\
\mathtt{let}\,x = \,!\,\ell\,\mathtt{in}\ \mathtt{free}\,\ell; \\
\mathtt{assert}(x = 42)
\end{array}
\;\middle\|\;
\begin{array}{l}
\mathtt{let}\,(\ell, c') = \mathtt{recv}\,c\,\mathtt{in} \\
\ell \leftarrow (!\,\ell + 2); \\
\mathtt{close}\,c' \\
\\
\\
\end{array}\right)
\end{array}$$

Protocol:

$$\mathtt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\,\mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$$\{\mathsf{True}\}$$
$$\mathbf{let}\ c = \mathbf{new}_{\mathbf{fun}}\ ()\ \mathbf{in}$$
$$\{c \rightarrowtail \mathtt{ses\_prot} * c \rightarrowtail \overline{\mathtt{ses\_prot}}\}$$
$$\left(\begin{array}{l} \mathbf{let}\ \ell = \mathbf{ref}\ 40\ \mathbf{in} \\ \mathbf{let}\ c' = \mathbf{send}\ c\ \ell\ \mathbf{in} \\ \mathbf{wait}\ c'; \\ \mathbf{let}\ x = !\,\ell\ \mathbf{in}\ \mathbf{free}\ \ell; \\ \mathbf{assert}(x = 42) \end{array} \;\middle\|\; \begin{array}{l} \mathbf{let}\ (\ell, c') = \mathbf{recv}\ c\ \mathbf{in} \\ \ell \leftarrow (!\,\ell + 2); \\ \mathbf{close}\ c' \\ \\ \\ \end{array}\right)$$

Protocol:

$$\mathtt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\,\mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$\{\mathsf{True}\}$
$\texttt{let}\, c = \texttt{new}_{\texttt{fun}}\,()\,\texttt{in}$
$\{c \rightarrowtail \texttt{ses\_prot} * c \rightarrowtail \overline{\texttt{ses\_prot}}\}$

$$\left(\begin{array}{l} \{c \rightarrowtail \texttt{ses\_prot}\} \\ \texttt{let}\,\ell = \texttt{ref}\,40\,\texttt{in} \\ \texttt{let}\,c' = \texttt{send}\,c\,\ell\,\texttt{in} \\ \texttt{wait}\,c'; \\ \texttt{let}\,x = !\,\ell\,\texttt{in}\ \texttt{free}\,\ell; \\ \texttt{assert}(x = 42) \end{array} \middle\| \begin{array}{l} \{c \rightarrowtail \overline{\texttt{ses\_prot}}\} \\ \texttt{let}\,(\ell, c') = \texttt{recv}\,c\,\texttt{in} \\ \ell \leftarrow (!\,\ell + 2); \\ \texttt{close}\,c' \\ \\ \\ \end{array}\right)$$

Protocol:

$$\texttt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\,\mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\ c = \mathtt{new_{fun}}\ ()\ \mathtt{in} \\
\{c \rightarrowtail \mathtt{ses\_prot} * c \rightarrowtail \overline{\mathtt{ses\_prot}}\} \\
\left(\begin{array}{l}
\{c \rightarrowtail \mathtt{ses\_prot}\} \\
\mathtt{let}\ \ell = \mathtt{ref}\ 40\ \mathtt{in} \\
\{c \rightarrowtail \mathtt{ses\_prot} * \ell \mapsto 40\} \\
\mathtt{let}\ c' = \mathtt{send}\ c\ \ell\ \mathtt{in} \\
\mathtt{wait}\ c'; \\
\mathtt{let}\ x = !\,\ell\ \mathtt{in}\ \mathtt{free}\ \ell; \\
\mathtt{assert}(x = 42)
\end{array}
\middle\|
\begin{array}{l}
\{c \rightarrowtail \overline{\mathtt{ses\_prot}}\} \\
\mathtt{let}\ (\ell, c') = \mathtt{recv}\ c\ \mathtt{in} \\
\ell \leftarrow (!\,\ell + 2); \\
\mathtt{close}\ c' \\
\\
\\
\\
\end{array}\right)
\end{array}
$$

Protocol:

$$\mathtt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\ \mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\, c = \mathtt{new_{fun}}\,()\ \mathtt{in} \\
\{c \rightarrowtail \mathtt{ses\_prot} * c \rightarrowtail \overline{\mathtt{ses\_prot}}\} \\
\left(\begin{array}{l}
\{c \rightarrowtail \mathtt{ses\_prot}\} \\
\mathtt{let}\, \ell = \mathtt{ref}\, 40\ \mathtt{in} \\
\{c \rightarrowtail \mathtt{ses\_prot} * \ell \mapsto 40\} \\
\mathtt{let}\, c' = \mathtt{send}\, c\, \ell\ \mathtt{in} \\
\{c' \rightarrowtail \mathbf{?end}\{\ell \mapsto 42\}\} \\
\mathtt{wait}\, c'; \\
\mathtt{let}\, x = !\,\ell\ \mathtt{in}\ \mathtt{free}\, \ell; \\
\mathtt{assert}(x = 42)
\end{array}
\middle\|
\begin{array}{l}
\{c \rightarrowtail \overline{\mathtt{ses\_prot}}\} \\
\mathtt{let}\, (\ell, c') = \mathtt{recv}\, c\ \mathtt{in} \\
\ell \leftarrow (!\,\ell + 2); \\
\mathtt{close}\, c' \\
\\
\\
\\
\\
\end{array}\right)
\end{array}
$$

Protocol:

$$\mathtt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\, \mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$$\{\mathsf{True}\}$$
$$\mathtt{let}\, c = \mathtt{new}_{\mathtt{fun}}\,()\,\mathtt{in}$$
$$\{c \rightarrowtail \mathtt{ses\_prot} * c \rightarrowtail \overline{\mathtt{ses\_prot}}\}$$

$$\left(\begin{array}{l}
\{c \rightarrowtail \mathtt{ses_prot}\} \\
\mathtt{let}\, \ell = \mathtt{ref}\, 40\, \mathtt{in} \\
\{c \rightarrowtail \mathtt{ses_prot} * \ell \mapsto 40\} \\
\mathtt{let}\, c' = \mathtt{send}\, c\, \ell\, \mathtt{in} \\
\{c' \rightarrowtail \mathbf{?end}\{\ell \mapsto 42\}\} \\
\mathtt{wait}\, c'; \\
\{\ell \mapsto 42\} \\
\mathtt{let}\, x = !\,\ell\, \mathtt{in}\ \mathtt{free}\, \ell; \\
\mathtt{assert}(x = 42)
\end{array}
\middle\|
\begin{array}{l}
\{c \rightarrowtail \overline{\mathtt{ses_prot}}\} \\
\mathtt{let}\, (\ell, c') = \mathtt{recv}\, c\, \mathtt{in} \\
\ell \leftarrow (!\,\ell + 2); \\
\mathtt{close}\, c' \\
\\ \\ \\ \\ \\
\end{array}\right)$$

Protocol:

$$\mathtt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\,\mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$$\{\mathsf{True}\}$$
$$\mathbf{let}\, c = \mathbf{new}_{\mathbf{fun}}\,()\,\mathbf{in}$$
$$\{c \rightarrowtail \mathtt{ses\_prot} * c \rightarrowtail \overline{\mathtt{ses\_prot}}\}$$

$$\left(\begin{array}{l} \{c \rightarrowtail \mathtt{ses\_prot}\} \\ \mathbf{let}\,\ell = \mathbf{ref}\,40\,\mathbf{in} \\ \{c \rightarrowtail \mathtt{ses\_prot} * \ell \mapsto 40\} \\ \mathbf{let}\,c' = \mathbf{send}\,c\,\ell\,\mathbf{in} \\ \{c' \rightarrowtail \mathbf{?end}\{\ell \mapsto 42\}\} \\ \mathbf{wait}\,c'; \\ \{\ell \mapsto 42\} \\ \mathbf{let}\,x = !\,\ell\,\mathbf{in}\ \mathbf{free}\,\ell; \\ \{x = 42\} \\ \mathbf{assert}(x = 42) \end{array} \middle\| \begin{array}{l} \{c \rightarrowtail \overline{\mathtt{ses\_prot}}\} \\ \mathbf{let}\,(\ell, c') = \mathbf{recv}\,c\,\mathbf{in} \\ \ell \leftarrow (!\,\ell + 2); \\ \mathbf{close}\,c' \\ \\ \\ \\ \\ \\ \\ \end{array}\right)$$

Protocol:

$$\mathtt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\,\mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\, c = \mathbf{new_{fun}}\,()\,\mathbf{in} \\
\{c \rightarrowtail \mathtt{ses\_prot} * c \rightarrowtail \overline{\mathtt{ses\_prot}}\} \\
\left(\begin{array}{l|l}
\{c \rightarrowtail \mathtt{ses\_prot}\} & \{c \rightarrowtail \overline{\mathtt{ses\_prot}}\} \\
\mathbf{let}\, \ell = \mathbf{ref}\, 40\, \mathbf{in} & \mathbf{let}\, (\ell, c') = \mathbf{recv}\, c\, \mathbf{in} \\
\{c \rightarrowtail \mathtt{ses\_prot} * \ell \mapsto 40\} & \ell \leftarrow (!\,\ell + 2); \\
\mathbf{let}\, c' = \mathbf{send}\, c\, \ell\, \mathbf{in} & \mathbf{close}\, c' \\
\{c' \rightarrowtail \mathbf{?end}\{\ell \mapsto 42\}\} & \\
\mathbf{wait}\, c'; & \\
\{\ell \mapsto 42\} & \\
\mathbf{let}\, x = !\,\ell\, \mathbf{in}\, \mathbf{free}\, \ell; & \\
\{x = 42\} & \\
\mathbf{assert}(x = 42) & \\
\{\mathsf{True}\} &
\end{array}\right)
\end{array}
$$

Protocol:

$$\mathtt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\, \mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\ c = \mathtt{new}_{\mathrm{fun}}\ ()\ \mathtt{in} \\
\{c \rightarrowtail \mathtt{ses\_prot} * c \rightarrowtail \overline{\mathtt{ses\_prot}}\}
\end{array}
$$

$$
\left(
\begin{array}{l}
\{c \rightarrowtail \mathtt{ses\_prot}\} \\
\mathtt{let}\ \ell = \mathtt{ref}\ 40\ \mathtt{in} \\
\{c \rightarrowtail \mathtt{ses\_prot} * \ell \mapsto 40\} \\
\mathtt{let}\ c' = \mathtt{send}\ c\ \ell\ \mathtt{in} \\
\{c' \rightarrowtail \mathbf{?end}\{\ell \mapsto 42\}\} \\
\mathtt{wait}\ c'; \\
\{\ell \mapsto 42\} \\
\mathtt{let}\ x = !\,\ell\ \mathtt{in}\ \mathtt{free}\ \ell; \\
\{x = 42\} \\
\mathtt{assert}(x = 42) \\
\{\mathsf{True}\}
\end{array}
\middle\|
\begin{array}{l}
\{c \rightarrowtail \overline{\mathtt{ses\_prot}}\} \\
\mathtt{let}\ (\ell, c') = \mathtt{recv}\ c\ \mathtt{in} \\
\{c' \rightarrowtail \mathbf{!end}\{\ell \mapsto 42\} * \ell \mapsto 40\} \\
\ell \leftarrow (!\,\ell + 2); \\
\mathtt{close}\ c' \\
\\ \\ \\ \\ \\ \\
\end{array}
\right)
$$

Protocol:

$$\mathtt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\ \mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$$\{\mathsf{True}\}$$
$$\mathtt{let}\ c = \mathbf{new}_{\mathrm{fun}}\ ()\ \mathtt{in}$$
$$\{c \rightarrowtail \mathtt{ses\_prot} * c \rightarrowtail \overline{\mathtt{ses\_prot}}\}$$

$$\left(\begin{array}{l} \{c \rightarrowtail \mathtt{ses\_prot}\} \\ \mathtt{let}\ \ell = \mathtt{ref}\ 40\ \mathtt{in} \\ \{c \rightarrowtail \mathtt{ses\_prot} * \ell \mapsto 40\} \\ \mathtt{let}\ c' = \mathbf{send}\ c\ \ell\ \mathtt{in} \\ \{c' \rightarrowtail \mathbf{?end}\{\ell \mapsto 42\}\} \\ \mathtt{wait}\ c'; \\ \{\ell \mapsto 42\} \\ \mathtt{let}\ x = !\,\ell\ \mathtt{in}\ \mathtt{free}\ \ell; \\ \{x = 42\} \\ \mathtt{assert}(x = 42) \\ \{\mathsf{True}\} \end{array} \middle\| \begin{array}{l} \{c \rightarrowtail \overline{\mathtt{ses\_prot}}\} \\ \mathtt{let}\ (\ell, c') = \mathbf{recv}\ c\ \mathtt{in} \\ \{c' \rightarrowtail \mathbf{!end}\{\ell \mapsto 42\} * \ell \mapsto 40\} \\ \ell \leftarrow (!\,\ell + 2); \\ \{c' \rightarrowtail \mathbf{!end}\{\ell \mapsto 42\} * \ell \mapsto 42\} \\ \mathtt{close}\ c' \\ \\ \\ \\ \\ \\ \end{array}\right)$$

Protocol:

$$\mathtt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\,\mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$$\{\mathsf{True}\}$$
$$\mathtt{let}\ c = \mathtt{new}_{\mathrm{fun}}\ ()\ \mathtt{in}$$
$$\{c \rightarrowtail \mathtt{ses\_prot} * c \rightarrowtail \overline{\mathtt{ses\_prot}}\}$$

$$\left(\begin{array}{l||l}
\{c \rightarrowtail \mathtt{ses_prot}\} & \{c \rightarrowtail \overline{\mathtt{ses_prot}}\} \\
\mathtt{let}\ \ell = \mathtt{ref}\ 40\ \mathtt{in} & \mathtt{let}\ (\ell, c') = \mathtt{recv}\ c\ \mathtt{in} \\
\{c \rightarrowtail \mathtt{ses_prot} * \ell \mapsto 40\} & \{c' \rightarrowtail \mathbf{!\,end}\{\ell \mapsto 42\} * \ell \mapsto 40\} \\
\mathtt{let}\ c' = \mathtt{send}\ c\ \ell\ \mathtt{in} & \ell \leftarrow (!\,\ell + 2); \\
\{c' \rightarrowtail \mathbf{?end}\{\ell \mapsto 42\}\} & \{c' \rightarrowtail \mathbf{!\,end}\{\ell \mapsto 42\} * \ell \mapsto 42\} \\
\mathtt{wait}\ c'; & \mathtt{close}\ c' \\
\{\ell \mapsto 42\} & \{\mathsf{True}\} \\
\mathtt{let}\ x = !\,\ell\ \mathtt{in}\ \mathtt{free}\ \ell; & \\
\{x = 42\} & \\
\mathtt{assert}(x = 42) & \\
\{\mathsf{True}\} &
\end{array}\right)$$

Protocol:

$$\mathtt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\ \mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$$\{\mathsf{True}\}$$
$$\texttt{let}\ c = \texttt{new}_{\text{fun}}\ ()\ \texttt{in}$$
$$\{c \rightarrowtail \texttt{ses\_prot} * c \rightarrowtail \overline{\texttt{ses\_prot}}\}$$

$$\left(\begin{array}{l|l}
\{c \rightarrowtail \texttt{ses_prot}\} & \{c \rightarrowtail \overline{\texttt{ses_prot}}\} \\
\texttt{let}\ \ell = \texttt{ref}\ 40\ \texttt{in} & \texttt{let}\ (\ell, c') = \texttt{recv}\ c\ \texttt{in} \\
\{c \rightarrowtail \texttt{ses_prot} * \ell \mapsto 40\} & \{c' \rightarrowtail \mathbf{!\,end}\{\ell \mapsto 42\} * \ell \mapsto 40\} \\
\texttt{let}\ c' = \texttt{send}\ c\ \ell\ \texttt{in} & \ell \leftarrow (!\,\ell + 2); \\
\{c' \rightarrowtail \mathbf{?end}\{\ell \mapsto 42\}\} & \{c' \rightarrowtail \mathbf{!\,end}\{\ell \mapsto 42\} * \ell \mapsto 42\} \\
\texttt{wait}\ c'; & \texttt{close}\ c' \\
\{\ell \mapsto 42\} & \{\mathsf{True}\} \\
\texttt{let}\ x = !\,\ell\ \texttt{in}\ \texttt{free}\ \ell; & \\
\{x = 42\} & \\
\texttt{assert}(x = 42) & \\
\{\mathsf{True}\} &
\end{array}\right)$$

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

Protocol:

$$\texttt{ses\_prot} \triangleq \mathbf{!}\,(\ell : \mathsf{Loc})\,\langle \ell \rangle \{\ell \mapsto 40\}.\ \mathbf{?end}\{\ell \mapsto 42\}$$

Functional Session Channel Example

$$\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\, c = \mathtt{new}_{\mathbf{fun}}\,()\,\mathtt{in} \\
\{c \rightarrowtail \mathtt{ses_prot} * c \rightarrowtail \overline{\mathtt{ses_prot}}\} \\
\left(\begin{array}{l}
\{c \rightarrowtail \mathtt{ses_prot}\} \\
\mathtt{let}\, \ell = \mathtt{ref}\, 40\, \mathtt{in} \\
\{c \rightarrowtail \mathtt{ses_prot} * \ell \mapsto 40\} \\
\mathtt{let}\, c' = \mathtt{send}\, c\, \ell\, \mathtt{in} \\
\{c' \rightarrowtail \mathbf{?end}\{\ell \mapsto 42\}\} \\
\mathtt{wait}\, c'; \\
\{\ell \mapsto 42\} \\
\mathtt{let}\, x = !\,\ell\, \mathtt{in}\ \mathtt{free}\, \ell; \\
\{x = 42\} \\
\mathtt{assert}(x = 42) \\
\{\mathsf{True}\}
\end{array}
\;\middle\|\;
\begin{array}{l}
\{c \rightarrowtail \overline{\mathtt{ses_prot}}\} \\
\mathtt{let}\, (\ell, c') = \mathtt{recv}\, c\, \mathtt{in} \\
\{c' \rightarrowtail \mathbf{!end}\{\ell \mapsto (x+2)\} * \ell \mapsto x\} \\
\ell \leftarrow (!\,\ell + 2); \\
\{c' \rightarrowtail \mathbf{!end}\{\ell \mapsto (x+2)\} * \ell \mapsto (x+2)\} \\
\mathtt{close}\, c' \\
\{\mathsf{True}\} \\
\\ \\ \\ \\
\end{array}\right) \\
\{\mathsf{True}\}
\end{array}$$

Protocol:

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

Session Channels Specifications

$$\begin{aligned}\mathbf{new}\,() &\triangleq \mathbf{let}\,c = \mathbf{new_{fun}}\,()\,\mathbf{in}\,(\mathbf{ref}\,c, \mathbf{ref}\,c)\\ c.\mathbf{send}(v) &\triangleq c \leftarrow \mathbf{send}\,(!\,c)\,v \\ c.\mathbf{recv}() &\triangleq \mathbf{let}\,(v, c') = \mathbf{recv}\;!\,c\;\mathbf{in}\,c \leftarrow c'; v\end{aligned}$$

$$\begin{aligned}c.\mathbf{close}() &\triangleq \mathbf{close}\,(!\,c); \mathbf{free}\,c \\ c.\mathbf{wait}() &\triangleq \mathbf{wait}\,(!\,c); \mathbf{free}\,c\end{aligned}$$

$$c \stackrel{\text{imp}}{\rightarrowtail} p \triangleq \exists (c' : \mathsf{Val}).\; c \mapsto c' * c' \rightarrowtail p$$

Session Channels Specifications

$$\begin{aligned}
\mathtt{new}\,() &\triangleq \mathtt{let}\, c = \mathtt{new}_{\mathtt{fun}}\,()\,\mathtt{in}\,(\mathtt{ref}\, c, \mathtt{ref}\, c) \\
c.\mathtt{send}(v) &\triangleq c \leftarrow \mathtt{send}\,(!\,c)\, v \qquad & c.\mathtt{close}() &\triangleq \mathtt{close}\,(!\,c); \mathtt{free}\, c \\
c.\mathtt{recv}() &\triangleq \mathtt{let}\,(v, c') = \mathtt{recv}\;!\,c\;\mathtt{in}\, c \leftarrow c'; v \qquad & c.\mathtt{wait}() &\triangleq \mathtt{wait}\,(!\,c); \mathtt{free}\, c
\end{aligned}$$

$$c \xrightarrowtail{\mathsf{imp}} p \triangleq \exists (c' : \mathrm{Val}).\, c \mapsto c' * c' \rightarrowtail p$$

Actris specifications:

$$\begin{aligned}
\{\mathrm{True}\}\ & \mathtt{new}\,()\ \{w.\, \exists c_1, c_2.\, w = (c_1, c_2) * c_1 \xrightarrowtail{\mathsf{imp}} p * c_2 \xrightarrow{\mathsf{imp}} \overline{p}\} \\
\{c \xrightarrowtail{\mathsf{imp}} (\mathbf{!}\,(x : \tau)\,\langle v\rangle\{P\}.\, p) * P\, t\}\ & c.\mathtt{send}(v\, t)\ \{c \xrightarrowtail{\mathsf{imp}} p\, t\} \\
\{c \xrightarrowtail{\mathsf{imp}} (\mathbf{?}(x : \tau)\,\langle v\rangle\{P\}.\, p)\}\ & c.\mathtt{recv}()\ \{w.\, \exists (x : \tau).\, w = (v\, x) * P\, x * c \xrightarrowtail{\mathsf{imp}} p\, x\} \\
\{c \rightarrowtail \mathbf{!end}\{P\} * P\}\ & c.\mathtt{close}()\ \{\mathrm{True}\} \\
\{c \rightarrowtail \mathbf{?end}\{P\}\}\ & c.\mathtt{wait}()\ \{P\}
\end{aligned}$$

Session Channel Example

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

Session Channel Example

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

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

Session Channel Example

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

Protocol:

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

Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\,(c_1, c_2) = \mathbf{new}\,()\ \mathbf{in} \\
\{c_1 \stackrel{\mathsf{imp}}{\rightarrowtail} \mathtt{ses\_prot} * c_2 \stackrel{\mathsf{imp}}{\rightarrowtail} \overline{\mathtt{ses\_prot}}\} \\
\left(\begin{array}{l}
\mathbf{let}\,\ell = \mathbf{ref}\,40\ \mathbf{in} \\
c_1.\mathbf{send}(\ell); \\
c_1.\mathbf{wait}(); \\
\mathbf{let}\,x = !\,\ell\ \mathbf{in}\ \mathbf{free}\,\ell; \\
\mathbf{assert}(x = 42)
\end{array}
\;\middle\|\;
\begin{array}{l}
\mathbf{let}\,\ell = c_2.\mathbf{recv}()\ \mathbf{in} \\
\ell \leftarrow (!\,\ell + 2); \\
c_2.\mathbf{close}() \\
\\
\\
\end{array}\right)
\end{array}
$$

Protocol:

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

Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\,(c_1, c_2) = \mathbf{new}\,()\ \mathbf{in} \\
\{c_1 \xrightarrow{\mathsf{imp}} \mathtt{ses\_prot} * c_2 \xrightarrow{\mathsf{imp}} \overline{\mathtt{ses\_prot}}\} \\
\left(\begin{array}{l}
\{c_1 \xrightarrow{\mathsf{imp}} \mathtt{ses\_prot}\} \\
\mathbf{let}\,\ell = \mathbf{ref}\,40\ \mathbf{in} \\
c_1.\mathbf{send}(\ell); \\
c_1.\mathbf{wait}(); \\
\mathbf{let}\,x = !\,\ell\ \mathbf{in}\ \mathbf{free}\,\ell; \\
\mathbf{assert}(x = 42)
\end{array}
\;\middle\|\;
\begin{array}{l}
\{c_2 \xrightarrow{\mathsf{imp}} \overline{\mathtt{ses\_prot}}\} \\
\mathbf{let}\,\ell = c_2.\mathbf{recv}()\ \mathbf{in} \\
\ell \leftarrow (!\,\ell + 2); \\
c_2.\mathbf{close}() \\
\\
\\
\end{array}\right)
\end{array}
$$

Protocol:

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

Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\,(c_1, c_2) = \mathbf{new}\,()\ \mathbf{in} \\
\{c_1 \xrightarrow{\text{imp}} \mathtt{ses\_prot} * c_2 \xrightarrow{\text{imp}} \overline{\mathtt{ses\_prot}}\} \\
\left(\begin{array}{l}
\{c_1 \xrightarrow{\text{imp}} \mathtt{ses\_prot}\} \\
\mathbf{let}\,\ell = \mathbf{ref}\,40\ \mathbf{in} \\
\{c_1 \xrightarrow{\text{imp}} \mathtt{ses\_prot} * \ell \mapsto 40\} \\
c_1.\mathbf{send}(\ell); \\
c_1.\mathbf{wait}(); \\
\mathbf{let}\,x = {!}\,\ell\ \mathbf{in}\ \mathbf{free}\,\ell; \\
\mathbf{assert}(x = 42)
\end{array}
\middle\|
\begin{array}{l}
\{c_2 \xrightarrow{\text{imp}} \overline{\mathtt{ses\_prot}}\} \\
\mathbf{let}\,\ell = c_2.\mathbf{recv}()\ \mathbf{in} \\
\ell \leftarrow ({!}\,\ell + 2); \\
c_2.\mathbf{close}() \\
\\
\\
\\
\end{array}\right)
\end{array}
$$

Protocol:

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

Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\,(c_1, c_2) = \mathbf{new}\,()\ \mathbf{in} \\
\{c_1 \xrightarrowtail{\mathsf{imp}} \mathtt{ses\_prot} * c_2 \xrightarrowtail{\mathsf{imp}} \overline{\mathtt{ses\_prot}}\} \\
\left(\begin{array}{l}
\{c_1 \xrightarrowtail{\mathsf{imp}} \mathtt{ses\_prot}\} \\
\mathbf{let}\,\ell = \mathbf{ref}\,40\ \mathbf{in} \\
\{c_1 \xrightarrowtail{\mathsf{imp}} \mathtt{ses\_prot} * \ell \mapsto 40\} \\
c_1.\mathbf{send}(\ell); \\
\{c_1 \xrightarrowtail{\mathsf{imp}} \mathbf{?end}\{\ell \mapsto 42\}\} \\
c_1.\mathbf{wait}(); \\
\mathbf{let}\,x = !\,\ell\ \mathbf{in}\ \mathbf{free}\,\ell; \\
\mathbf{assert}(x = 42)
\end{array}
\middle\|
\begin{array}{l}
\{c_2 \xrightarrowtail{\mathsf{imp}} \overline{\mathtt{ses\_prot}}\} \\
\mathbf{let}\,\ell = c_2.\mathbf{recv}()\ \mathbf{in} \\
\ell \leftarrow (!\,\ell + 2); \\
c_2.\mathbf{close}() \\
\\ \\ \\ \\
\end{array}\right)
\end{array}
$$

Protocol:

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

Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathbf{let}\,(c_1, c_2) = \mathbf{new}\,()\,\mathbf{in} \\
\{c_1 \overset{\mathsf{imp}}{\rightarrowtail} \mathtt{ses\_prot} * c_2 \overset{\mathsf{imp}}{\rightarrowtail} \overline{\mathtt{ses\_prot}}\} \\
\left(\begin{array}{l}
\{c_1 \overset{\mathsf{imp}}{\rightarrowtail} \mathtt{ses\_prot}\} \\
\mathbf{let}\,\ell = \mathbf{ref}\,40\,\mathbf{in} \\
\{c_1 \overset{\mathsf{imp}}{\rightarrowtail} \mathtt{ses\_prot} * \ell \mapsto 40\} \\
c_1.\mathbf{send}(\ell); \\
\{c_1 \overset{\mathsf{imp}}{\rightarrowtail} \mathbf{?end}\{\ell \mapsto 42\}\} \\
c_1.\mathbf{wait}(); \\
\{\ell \mapsto 42\} \\
\mathbf{let}\,x = !\,\ell\,\mathbf{in}\ \mathbf{free}\,\ell; \\
\mathbf{assert}(x = 42)
\end{array}
\;\middle\|\;
\begin{array}{l}
\{c_2 \overset{\mathsf{imp}}{\rightarrowtail} \overline{\mathtt{ses\_prot}}\} \\
\mathbf{let}\,\ell = c_2.\mathbf{recv}()\,\mathbf{in} \\
\ell \leftarrow (!\,\ell + 2); \\
c_2.\mathbf{close}() \\
\\ \\ \\ \\ \\
\end{array}\right)
\end{array}
$$

Protocol:

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

Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\,(c_1, c_2) = \mathtt{new}\,()\ \mathtt{in} \\
\{c_1 \stackrel{\mathsf{imp}}{\rightarrowtail} \mathtt{ses\_prot} * c_2 \stackrel{\mathsf{imp}}{\rightarrowtail} \overline{\mathtt{ses\_prot}}\} \\
\left(\begin{array}{l}
\{c_1 \stackrel{\mathsf{imp}}{\rightarrowtail} \mathtt{ses\_prot}\} \\
\mathtt{let}\ \ell = \mathtt{ref}\,40\ \mathtt{in} \\
\{c_1 \stackrel{\mathsf{imp}}{\rightarrowtail} \mathtt{ses\_prot} * \ell \mapsto 40\} \\
c_1.\mathtt{send}(\ell); \\
\{c_1 \stackrel{\mathsf{imp}}{\rightarrowtail} \mathbf{?end}\{\ell \mapsto 42\}\} \\
c_1.\mathtt{wait}(); \\
\{\ell \mapsto 42\} \\
\mathtt{let}\ x = !\,\ell\ \mathtt{in}\ \mathtt{free}\ \ell; \\
\{x = 42\} \\
\mathtt{assert}(x = 42)
\end{array}
\middle\|
\begin{array}{l}
\{c_2 \stackrel{\mathsf{imp}}{\rightarrowtail} \overline{\mathtt{ses\_prot}}\} \\
\mathtt{let}\ \ell = c_2.\mathtt{recv}()\ \mathtt{in} \\
\ell \leftarrow (!\,\ell + 2); \\
c_2.\mathtt{close}() \\
\\ \\ \\ \\ \\ \\
\end{array}\right)
\end{array}
$$

Protocol:

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

Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\}\\
\mathtt{let}\,(c_1, c_2) = \mathtt{new}\,()\,\mathtt{in}\\
\{c_1 \overset{\text{imp}}{\rightarrowtail} \mathtt{ses\_prot} * c_2 \overset{\text{imp}}{\rightarrowtail} \overline{\mathtt{ses\_prot}}\}\\
\left(\begin{array}{l}
\{c_1 \overset{\text{imp}}{\rightarrowtail} \mathtt{ses\_prot}\}\\
\mathtt{let}\,\ell = \mathtt{ref}\,40\,\mathtt{in}\\
\{c_1 \overset{\text{imp}}{\rightarrowtail} \mathtt{ses\_prot} * \ell \mapsto 40\}\\
c_1.\mathtt{send}(\ell);\\
\{c_1 \overset{\text{imp}}{\rightarrowtail} \mathbf{?end}\{\ell \mapsto 42\}\}\\
c_1.\mathtt{wait}();\\
\{\ell \mapsto 42\}\\
\mathtt{let}\,x = !\,\ell\,\mathtt{in}\ \mathtt{free}\,\ell;\\
\{x = 42\}\\
\mathtt{assert}(x = 42)\\
\{\mathsf{True}\}
\end{array}
\;\middle\|\;
\begin{array}{l}
\{c_2 \overset{\text{imp}}{\rightarrowtail} \overline{\mathtt{ses\_prot}}\}\\
\mathtt{let}\,\ell = c_2.\mathtt{recv}()\,\mathtt{in}\\
\ell \leftarrow (!\,\ell + 2);\\
c_2.\mathtt{close}()\\
\\
\\
\\
\\
\\
\\
\\
\end{array}\right)
\end{array}
$$

Protocol:

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

Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\texttt{let}\,(c_1, c_2) = \textbf{new}\,()\,\texttt{in} \\
\{c_1 \xrightarrow{\text{imp}} \texttt{ses\_prot} * c_2 \xrightarrow{\text{imp}} \overline{\texttt{ses\_prot}}\}
\end{array}
$$

$$
\left(
\begin{array}{l|l}
\{c_1 \xrightarrow{\text{imp}} \texttt{ses\_prot}\} & \{c_2 \xrightarrow{\text{imp}} \overline{\texttt{ses\_prot}}\} \\
\texttt{let}\,\ell = \texttt{ref}\,40\,\texttt{in} & \texttt{let}\,\ell = c_2.\texttt{recv}()\,\texttt{in} \\
\{c_1 \xrightarrow{\text{imp}} \texttt{ses\_prot} * \ell \mapsto 40\} & \{c_2 \xrightarrow{\text{imp}} \mathbf{!\,end}\{\ell \mapsto (x+2)\} * \ell \mapsto x\} \\
c_1.\texttt{send}(\ell); & \ell \leftarrow (!\,\ell + 2); \\
\{c_1 \xrightarrow{\text{imp}} \mathbf{?end}\{\ell \mapsto 42\}\} & c_2.\texttt{close}() \\
c_1.\texttt{wait}(); & \\
\{\ell \mapsto 42\} & \\
\texttt{let}\,x = !\,\ell\,\texttt{in}\,\texttt{free}\,\ell; & \\
\{x = 42\} & \\
\texttt{assert}(x = 42) & \\
\{\mathsf{True}\} &
\end{array}
\right)
$$

Protocol:

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

Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\textbf{let}\,(c_1, c_2) = \textbf{new}\,()\,\textbf{in} \\
\{c_1 \overset{\text{imp}}{\rightarrowtail} \texttt{ses\_prot} * c_2 \overset{\text{imp}}{\rightarrowtail} \overline{\texttt{ses\_prot}}\} \\
\left(\begin{array}{l|l}
\{c_1 \overset{\text{imp}}{\rightarrowtail} \texttt{ses\_prot}\} & \{c_2 \overset{\text{imp}}{\rightarrowtail} \overline{\texttt{ses\_prot}}\} \\
\textbf{let}\,\ell = \textbf{ref}\,40\,\textbf{in} & \textbf{let}\,\ell = c_2.\textbf{recv}()\,\textbf{in} \\
\{c_1 \overset{\text{imp}}{\rightarrowtail} \texttt{ses\_prot} * \ell \mapsto 40\} & \{c_2 \overset{\text{imp}}{\rightarrowtail} \,!\,\textbf{end}\{\ell \mapsto (x+2)\} * \ell \mapsto x\} \\
c_1.\textbf{send}(\ell); & \ell \leftarrow (!\,\ell + 2); \\
\{c_1 \overset{\text{imp}}{\rightarrowtail} \textbf{?end}\{\ell \mapsto 42\}\} & \{c_2 \overset{\text{imp}}{\rightarrow tail} \,!\,\textbf{end}\{\ell \mapsto (x+2)\} * \ell \mapsto (x+2)\} \\
c_1.\textbf{wait}(); & c_2.\textbf{close}() \\
\{\ell \mapsto 42\} & \\
\textbf{let}\,x = !\,\ell\,\textbf{in}\,\textbf{free}\,\ell; & \\
\{x = 42\} & \\
\textbf{assert}(x = 42) & \\
\{\mathsf{True}\} &
\end{array}\right)
\end{array}
$$

Protocol:

$$\texttt{ses\_prot} \triangleq \,!\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\textbf{?end}\{\ell \mapsto (x+2)\}$$

Session Channel Example

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\mathtt{let}\,(c_1, c_2) = \mathtt{new}\,()\,\mathtt{in} \\
\{c_1 \overset{\text{imp}}{\rightarrowtail} \mathtt{ses\_prot} * c_2 \overset{\text{imp}}{\rightarrowtail} \overline{\mathtt{ses\_prot}}\} \\
\left(\begin{array}{l||l}
\{c_1 \overset{\text{imp}}{\rightarrowtail} \mathtt{ses\_prot}\} & \{c_2 \overset{\text{imp}}{\rightarrowtail} \overline{\mathtt{ses\_prot}}\} \\
\mathtt{let}\,\ell = \mathtt{ref}\,40\,\mathtt{in} & \mathtt{let}\,\ell = c_2.\mathtt{recv}()\,\mathtt{in} \\
\{c_1 \overset{\text{imp}}{\rightarrowtail} \mathtt{ses\_prot} * \ell \mapsto 40\} & \{c_2 \overset{\text{imp}}{\rightarrowtail} \mathbf{!\,end}\{\ell \mapsto (x+2)\} * \ell \mapsto x\} \\
c_1.\mathtt{send}(\ell); & \ell \leftarrow (!\,\ell + 2); \\
\{c_1 \overset{\text{imp}}{\rightarrowtail} \mathbf{?end}\{\ell \mapsto 42\}\} & \{c_2 \overset{\text{imp}}{\rightarrowtail} \mathbf{!\,end}\{\ell \mapsto (x+2)\} * \ell \mapsto (x+2)\} \\
c_1.\mathtt{wait}(); & c_2.\mathtt{close}() \\
\{\ell \mapsto 42\} & \{\mathsf{True}\} \\
\mathtt{let}\,x = !\,\ell\,\mathtt{in}\,\mathtt{free}\,\ell; & \\
\{x = 42\} & \\
\mathtt{assert}(x = 42) & \\
\{\mathsf{True}\} &
\end{array}\right)
\end{array}
$$

Protocol:

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

Session Channel Example

$$\{\mathsf{True}\}$$
$$\texttt{let}\,(c_1, c_2) = \texttt{new}\,()\;\texttt{in}$$
$$\{c_1 \overset{\mathsf{imp}}{\rightarrowtail} \mathtt{ses\_prot} * c_2 \overset{\mathsf{imp}}{\rightarrowtail} \overline{\mathtt{ses\_prot}}\}$$

$$\left(\begin{array}{l|l}
\{c_1 \overset{\mathsf{imp}}{\rightarrowtail} \mathtt{ses_prot}\} & \{c_2 \overset{\mathsf{imp}}{\rightarrowtail} \overline{\mathtt{ses_prot}}\} \\
\texttt{let}\,\ell = \texttt{ref}\,40\;\texttt{in} & \texttt{let}\,\ell = c_2.\texttt{recv}()\;\texttt{in} \\
\{c_1 \overset{\mathsf{imp}}{\rightarrowtail} \mathtt{ses_prot} * \ell \mapsto 40\} & \{c_2 \overset{\mathsf{imp}}{\rightarrowtail} \mathbf{!\,end}\{\ell \mapsto (x+2)\} * \ell \mapsto x\} \\
c_1.\texttt{send}(\ell); & \ell \leftarrow (!\,\ell + 2); \\
\{c_1 \overset{\mathsf{imp}}{\rightarrowtail} \mathbf{?end}\{\ell \mapsto 42\}\} & \{c_2 \overset{\mathsf{imp}}{\rightarrowtail} \mathbf{!\,end}\{\ell \mapsto (x+2)\} * \ell \mapsto (x+2)\} \\
c_1.\texttt{wait}(); & c_2.\texttt{close}() \\
\{\ell \mapsto 42\} & \{\mathsf{True}\} \\
\texttt{let}\,x = !\,\ell\;\texttt{in}\;\texttt{free}\,\ell; & \\
\{x = 42\} & \\
\texttt{assert}(x = 42) & \\
\{\mathsf{True}\} &
\end{array}\right)$$

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

Protocol:

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

Break (30 min!)
We start again at 16:00!

If you attend the Coq hacking session please pull
`https://gitlab.mpi-sws.org/iris/tutorial-popl24`
and follow the installation instructions!

Iris invariants and ghost state

Tutorial Timeline

Part 1: 14:00 – 15:30

- ~~Introduction (10 min)~~
- ~~Layered implementation of session channels (10 min)~~
- ~~Basic concurrent separation logic and one-shot protocols (30 min)~~
- ~~**Break** (10 min)~~
- ~~Dependent separation protocols (30 min)~~

~~**Break** (30 min)~~

Part 2: 16:00 – 17:30

- Iris invariants and ghost state (30 min)
- **Break** (10 min)
- Supervised Coq hacking (50 min)

Overview of Abstraction Layers

Layer	Reasoning principles / specifications
#1 Iris's HeapLang	Basic concurrent separation logic Iris invariants and ghost state
#2 One-shot channels	One-shot protocols
#3 Functional session channels	Dependent separation protocols
#4 Session channels	Dependent separation protocols

One-Shot Channels Recap

One-shot channel implementations:

$$\begin{aligned}
\texttt{new1}\,() &\triangleq \texttt{ref None} \\
\texttt{send1}\,c\,v &\triangleq c \leftarrow \texttt{Some}\,v \\
\texttt{recv1}\,c &\triangleq \texttt{let}\,x = {!}\,c\,\texttt{in} \\
&\ \texttt{match}\,x\,\texttt{with} \\
&\ \quad \texttt{None} \Rightarrow \texttt{recv1}\,c \\
&\ \ \mid \texttt{Some}\,v \Rightarrow \texttt{free}\,c;\ v \\
&\ \texttt{end}
\end{aligned}$$

One-shot channel specifications:

$$\begin{aligned}
\{\mathsf{True}\}\ &\texttt{new1}\,()\ \{w.\,\exists c.\, w = c * c \rightarrowtail p * c \rightarrowtail \overline{p}\} \\
\{c \rightarrowtail (\mathsf{Send}, \Phi) * \Phi\, v\}\ &\texttt{send1}\,c\,v\ \{\mathsf{True}\} \\
\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}\ &\texttt{recv1}\,c\ \{w.\,\Phi\, w\}
\end{aligned}$$

One-Shot Channels Recap

One-shot channel implementations:

$$
\begin{aligned}
\texttt{new1}\,() &\triangleq \texttt{ref None} \\
\texttt{send1}\,c\,v &\triangleq c \leftarrow \texttt{Some}\,v \\
\texttt{recv1}\,c &\triangleq \texttt{let}\,x = {!}\,c\,\texttt{in} \\
&\quad\ \texttt{match}\,x\,\texttt{with} \\
&\quad\quad \texttt{None} \quad\ \Rightarrow \texttt{recv1}\,c \\
&\quad\ \mid \texttt{Some}\,v \Rightarrow \texttt{free}\,c;\ v \\
&\quad\ \texttt{end}
\end{aligned}
$$

One-shot channel specifications:

$$
\begin{aligned}
\{\mathsf{True}\}\ &\texttt{new1}\,()\ \{w.\, \exists c.\, w = c * c \rightarrowtail p * c \rightarrowtail \overline{p}\} \\
\{c \rightarrowtail (\mathsf{Send}, \Phi) * \Phi\, v\}\ &\texttt{send1}\,c\,v\ \{\mathsf{True}\} \\
\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}\ &\texttt{recv1}\,c\ \{w.\, \Phi\, w\}
\end{aligned}
$$

Crux: Definition of $c \rightarrowtail p$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology

$$c \rightarrowtail (\mathit{tag}, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

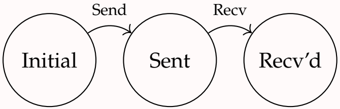
1. Model abstraction as a state transition system (STS)

$$c \rightarrowtail (\mathit{tag}, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)

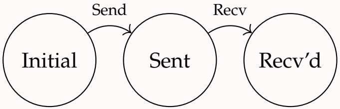


$$c \rightarrowtail (tag, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)
2. Define an invariant as a disjunction of the states

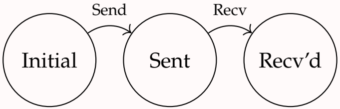


$$c \rightarrowtail (tag, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)
2. Define an invariant as a disjunction of the states



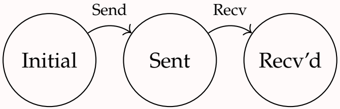
$$\mathsf{chan\_inv} \triangleq (\underbrace{\qquad\qquad}_{\text{(1) initial state}}) \vee (\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\qquad\qquad}_{\text{(3) final state}})$$

$$c \rightarrowtail (\mathit{tag}, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)
2. Define an invariant as a disjunction of the states
3. Determine resource ownership of each state



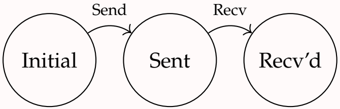
$$\mathsf{chan\_inv} \triangleq (\underbrace{\qquad\qquad}_{\text{(1) initial state}}) \lor (\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{(2) message sent, but not yet received}}) \lor (\underbrace{\qquad\qquad\qquad}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)
2. Define an invariant as a disjunction of the states
3. Determine resource ownership of each state



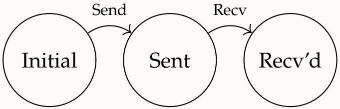
$$\mathsf{chan\_inv}\quad c \quad \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\, v \qquad\qquad}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\qquad\qquad}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)
2. Define an invariant as a disjunction of the states
3. Determine resource ownership of each state



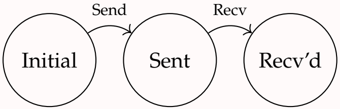
$$\mathsf{chan\_inv} \qquad c\ \Phi \triangleq (\underbrace{c \mapsto \texttt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \texttt{Some}\, v * \Phi\ v \qquad\qquad}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\qquad\qquad\qquad}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)
2. Define an invariant as a disjunction of the states
3. Determine resource ownership of each state
4. Encode STS transition permissions with ghost state



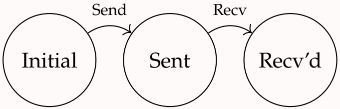
$$\mathsf{chan\_inv} \quad c\ \Phi \triangleq (\underbrace{c \mapsto \texttt{None}}_{\text{(1) initial state}}) \lor (\underbrace{\exists v.\, c \mapsto \texttt{Some}\, v * \Phi\, v}_{\text{(2) message sent, but not yet received}}) \lor (\underbrace{\qquad\qquad}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)
2. Define an invariant as a disjunction of the states
3. Determine resource ownership of each state
4. Encode STS transition permissions with ghost state



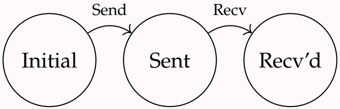
$$\mathsf{chan\_inv}\ \gamma_s \quad c\ \Phi \triangleq (\underbrace{c \mapsto \texttt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \texttt{Some}\, v * \Phi\, v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\qquad\qquad}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)
2. Define an invariant as a disjunction of the states
3. Determine resource ownership of each state
4. Encode STS transition permissions with ghost state



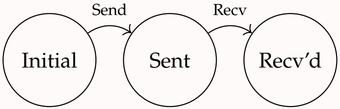
$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)
2. Define an invariant as a disjunction of the states
3. Determine resource ownership of each state
4. Encode STS transition permissions with ghost state
5. Give concurrent actors access to the invariant and their respective ghost state



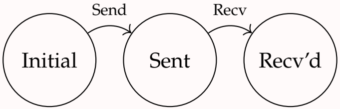
$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)
2. Define an invariant as a disjunction of the states
3. Determine resource ownership of each state
4. Encode STS transition permissions with ghost state
5. Give concurrent actors access to the invariant and their respective ghost state



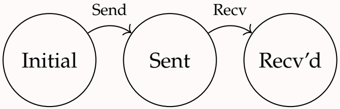
$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\ \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}\ \ldots$$

Definition of One-Shot Channel Resource

One-shot channel ownership defined using standard Iris methodology:

1. Model abstraction as a state transition system (STS)
2. Define an invariant as a disjunction of the states
3. Determine resource ownership of each state
4. Encode STS transition permissions with ghost state
5. Give concurrent actors access to the invariant and their respective ghost state



$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\, \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

Invariants

The invariant assertion $\boxed{R}$ expresses that R is maintained as an invariant on the state

Invariants

The invariant assertion $\boxed{R}$ expresses that R is maintained as an invariant on the state

Invariant opening:

$$\frac{\{R * P\}\; e\; \{R * Q\} \qquad e \text{ atomic}}{\left\{\boxed{R} * P\right\}\; e\; \left\{\boxed{R} * Q\right\}}$$

Invariants

The invariant assertion $\boxed{R}$ expresses that R is maintained as an invariant on the state

Invariant opening:

$$\frac{\{R * P\}\; e\; \{R * Q\} \qquad e \text{ atomic}}{\left\{\boxed{R} * P\right\}\; e\; \left\{\boxed{R} * Q\right\}}$$

Invariant allocation:

$$\frac{P \Rrightarrow P' \qquad \{P'\}\; e\; \{w.\, Q'\} \qquad \forall w.\, Q' \Rrightarrow Q}{\{P\}\; e\; \{w.\, Q\}} \qquad P \Rrightarrow \boxed{P}$$

Invariants

The invariant assertion $\boxed{R}$ expresses that R is maintained as an invariant on the state

Invariant opening:

$$\frac{\{R * P\}\, e\, \{R * Q\} \qquad e \text{ atomic}}{\left\{\boxed{R} * P\right\}\, e\, \left\{\boxed{R} * Q\right\}}$$

Invariant allocation:

$$\frac{P \Rrightarrow P' \qquad \{P'\}\, e\, \{w.\, Q'\} \qquad \forall w.\, Q' \Rrightarrow Q}{\{P\}\, e\, \{w.\, Q\}} \qquad P \Rrightarrow \boxed{P}$$

Invariant duplication: $\boxed{R} \vdash \boxed{R} * \boxed{R}$

Invariants

> **The invariant assertion** $\boxed{R}^{\mathcal{N}}$ expresses that R is maintained as an invariant on the state

Invariant opening:

$$\frac{\{R * P\}\; e\; \{R * Q\}_{\mathcal{E}} \qquad e \text{ atomic}}{\left\{\boxed{R}^{\mathcal{N}} * P\right\}\; e\; \left\{\boxed{R}^{\mathcal{N}} * Q\right\}_{\mathcal{E} \uplus \mathcal{N}}}$$

Invariant allocation:

$$\frac{P \Rrightarrow P' \qquad \{P'\}\; e\; \{w.\, Q'\}_{\mathcal{N}} \qquad \forall w.\, Q' \Rrightarrow Q}{\{P\}\; e\; \{w.\, Q\}_{\mathcal{N}}} \qquad P \Rrightarrow \boxed{P}^{\mathcal{N}}$$

Invariant duplication: $\boxed{R}^{\mathcal{N}} \vdash \boxed{R}^{\mathcal{N}} * \boxed{R}^{\mathcal{N}}$

Technicalities: names prevent opening the same invariant twice

Invariants

The invariant assertion $\boxed{R}^{\mathcal{N}}$ expresses that R is maintained as an invariant on the state

Invariant opening:

$$\frac{\{\triangleright R * P\}\; e\; \{\triangleright R * Q\}_{\mathcal{E}} \qquad e \text{ atomic}}{\left\{\boxed{R}^{\mathcal{N}} * P\right\}\; e\; \left\{\boxed{R}^{\mathcal{N}} * Q\right\}_{\mathcal{E} \uplus \mathcal{N}}}$$

Invariant allocation:

$$\frac{P \Rrightarrow P' \qquad \{P'\}\; e\; \{w.\, Q'\}_{\mathcal{N}} \qquad \forall w.\, Q' \Rrightarrow Q}{\{P\}\; e\; \{w.\, Q\}_{\mathcal{N}}} \qquad \triangleright P \Rrightarrow \boxed{P}^{\mathcal{N}}$$

Invariant duplication: $\boxed{R}^{\mathcal{N}} \vdash \boxed{R}^{\mathcal{N}} * \boxed{R}^{\mathcal{N}}$

Technicalities: names prevent opening the same invariant twice and the later $\triangleright$ is needed for impredicativity, i.e., $\boxed{\ldots \boxed{R}^{\mathcal{N}_2} \ldots}^{\mathcal{N}_1}$

Consider the invariant:

$$\boxed{(\underbrace{c \mapsto \texttt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \texttt{Some}\, v * \Phi\, v \ldots}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\ldots}_{\text{(3) final state}})}$$

How to determine which state the one-shot channel is in?

Ghost Tokens

Consider the invariant:

$$\boxed{(\underbrace{c \mapsto \texttt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \texttt{Some}\, v * \Phi\, v * \mathsf{tok}\, \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\, \gamma_s * \mathsf{tok}\, \gamma_r}_{\text{(3) final state}})}$$

How to determine which state the one-shot channel is in?

Ghost tokens allow deriving contradictions:

$$\mathsf{True} \Rrightarrow \exists \gamma.\, \mathsf{tok}\, \gamma \qquad \mathsf{tok}\, \gamma * \mathsf{tok}\, \gamma \vdash \mathsf{False}$$

Ghost Tokens

Consider the invariant:

$$\boxed{(\underbrace{c \mapsto \texttt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \texttt{Some}\, v * \Phi\, v * \mathsf{tok}\, \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\, \gamma_s * \mathsf{tok}\, \gamma_r}_{\text{(3) final state}})}$$

How to determine which state the one-shot channel is in?

Ghost tokens allow deriving contradictions:

$$\mathsf{True} \Rrightarrow \exists \gamma.\, \mathsf{tok}\, \gamma \qquad \mathsf{tok}\, \gamma * \mathsf{tok}\, \gamma \vdash \mathsf{False}$$

You can exclude cases which would end up in duplicate tokens:

$$\frac{((c \mapsto \texttt{None}) \vee (\exists v.\, c \mapsto \texttt{Some}\, v * \Phi\, v * \mathsf{tok}\, \gamma_s) \vee (\mathsf{tok}\, \gamma_s * \mathsf{tok}\, \gamma_r)) * \mathsf{tok}\, \gamma_s}{c \mapsto \texttt{None} * \mathsf{tok}\, \gamma_s}$$

Proof of New

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\, \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

ref None

Proof of New

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \lor (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\, v * \Phi\, v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \lor (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\, \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{\mathsf{True}\}$

ref None

$\{w.\, \exists c.\, w = c * c \rightarrowtail p * c \rightarrowtail \overline{p}\}$

Proof of New

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\, \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{\mathsf{True}\}$

ref None

$\{w.\, \exists c.\, w = c * c \mapsto \mathtt{None}\}$

$\{w.\, \exists c.\, w = c * c \rightarrowtail p * c \rightarrowtail \overline{p}\}$

Proof of New

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\, \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{\mathsf{True}\}$

$\mathtt{ref}\, \mathtt{None}$

$\{w.\, \exists c.\, w = c * c \mapsto \mathtt{None}\}$

$\{w.\, \exists c.\, w = c * c \mapsto \mathtt{None} * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$

$\{w.\, \exists c.\, w = c * c \rightarrowtail p * c \rightarrowtail \overline{p}\}$

Proof of New

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \texttt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \texttt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\, \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{\mathsf{True}\}$

$\texttt{ref None}$

$\{w.\, \exists c.\, w = c * c \mapsto \texttt{None}\}$

$\{w.\, \exists c.\, w = c * c \mapsto \texttt{None} * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$

$\{w.\, \exists c.\, w = c * \boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$ $\quad // p = (tag, \Phi)$

$\{w.\, \exists c.\, w = c * c \rightarrowtail p * c \rightarrowtail \bar{p}\}$

Proof of New

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\ c \mapsto \mathtt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\ \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{\mathsf{True}\}$

$\mathtt{ref}\, \mathtt{None}$

$\{w.\ \exists c.\ w = c * c \mapsto \mathtt{None}\}$

$\{w.\ \exists c.\ w = c * c \mapsto \mathtt{None} * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$

$\{w.\ \exists c.\ w = c * \boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$ $\quad // p = (tag, \Phi)$

$\{w.\ \exists c.\ w = c * \boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_s * \boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_r\}$

$\{w.\ \exists c.\ w = c * c \rightarrowtail p * c \rightarrowtail \overline{p}\}$

Proof of New

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \lor (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \lor (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\ \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{\mathsf{True}\}$

$\mathtt{ref\,None}$

$\{w.\ \exists c.\ w = c * c \mapsto \mathtt{None}\}$

$\{w.\ \exists c.\ w = c * c \mapsto \mathtt{None} * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$

$\{w.\ \exists c.\ w = c * \boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$ $\quad // p = (tag, \Phi)$

$\{w.\ \exists c.\ w = c * \boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_s * \boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_r\}$

$\{w.\ \exists c.\ w = c * c \rightarrowtail (\mathsf{Send}, \Phi) * c \rightarrowtail (\mathsf{Recv}, \Phi)\}$

$\{w.\ \exists c.\ w = c * c \rightarrowtail p * c \rightarrowtail \overline{p}\}$

Proof of Send

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \texttt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \texttt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\, \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$$c \leftarrow \texttt{Some}\ v$$

Proof of Send

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \texttt{None}}_{\text{(1) initial state}}) \lor (\underbrace{\exists v.\, c \mapsto \texttt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \lor (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\, \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{c \rightarrowtail (\mathsf{Send}, \Phi) * \Phi\ v\}$
$c \leftarrow \texttt{Some}\ v$
$\{\mathsf{True}\}$

Proof of Send

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\ c \mapsto \mathtt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\ \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{c \rightarrowtail (\mathsf{Send}, \Phi) * \Phi\ v\}$
$\{\boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_s * \Phi\ v\}$
$c \leftarrow \mathtt{Some}\ v$
$\{\mathsf{True}\}$

Proof of Send

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \texttt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \texttt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\ \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{c \rightarrowtail (\mathsf{Send}, \Phi) * \Phi\ v\}$

$\{\boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

$\quad\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

$c \leftarrow \texttt{Some}\ v$

$\{\mathsf{True}\}$

Proof of Send

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \texttt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \texttt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\ \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{c \rightarrowtail (\mathsf{Send}, \Phi) * \Phi\ v\}$

$\{\boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

$\quad\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

$\quad\{c \mapsto \texttt{None} * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

$c \leftarrow \texttt{Some}\ v$

$\{\mathsf{True}\}$

Proof of Send

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\, \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{c \rightarrowtail (\mathsf{Send}, \Phi) * \Phi\ v\}$

$\{\boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

  $\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

  $\{c \mapsto \mathtt{None} * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

$c \leftarrow \mathtt{Some}\ v$

  $\{c \mapsto \mathtt{Some}\ v * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

$\{\mathsf{True}\}$

Proof of Send

$$\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi \triangleq (\underbrace{c \mapsto \mathtt{None}}_{\text{(1) initial state}}) \vee (\underbrace{\exists v.\, c \mapsto \mathtt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s}_{\text{(2) message sent, but not yet received}}) \vee (\underbrace{\mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r}_{\text{(3) final state}})$$

$$c \rightarrowtail (tag, \Phi) \triangleq \exists \gamma_s, \gamma_r.\, \boxed{\mathsf{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \begin{cases} \mathsf{tok}\ \gamma_s & \text{if } tag = \mathsf{Send} \\ \mathsf{tok}\ \gamma_r & \text{if } tag = \mathsf{Recv} \end{cases}$$

$\{c \rightarrowtail (\mathsf{Send}, \Phi) * \Phi\ v\}$

$\{\boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi} * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

$\quad\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

$\quad\{c \mapsto \mathtt{None} * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

$c \leftarrow \mathtt{Some}\ v$

$\quad\{c \mapsto \mathtt{Some}\ v * \mathsf{tok}\ \gamma_s * \Phi\ v\}$

$\quad\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi\}$

$\{\mathsf{True}\}$

Proof of Receive

```
let w = !c in
match w with
  None   ⇒ recv1 c
| Some v ⇒ free c; v
end
```

Proof of Receive

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$

$\texttt{let}\ w = !\,c\ \texttt{in}$

$\texttt{match}\ w\ \texttt{with}$

$\quad \texttt{None} \quad \Rightarrow \texttt{recv1}\ c$

$\mid \texttt{Some}\ v \Rightarrow \texttt{free}\ c;\ v$

$\texttt{end}$

$\{w.\ \Phi\ w\}$

Proof of Receive

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$

$\{\mathsf{tok}\ \gamma_r * \boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}\}$

```
let w = ! c in
match w with
  None   ⇒ recv1 c
| Some v ⇒ free c; v
end
```

$\{w.\ \Phi\ w\}$

Proof of Receive

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$

```
let w = !c in
match w with
  None   ⇒ recv1 c
| Some v ⇒ free c; v
end
```

$\{w.\ \Phi\ w\}$

Duplicable propositions:

$\boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

Proof of Receive

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\quad\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$

```
let w = !c in
match w with
  None   ⇒ recv1 c
| Some v ⇒ free c; v
end
```

$\{w.\ \Phi\ w\}$

Duplicable propositions:

$\boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

Proof of Receive

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\quad\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\quad\{(c \mapsto \textbf{None}) \vee (\exists v.\, c \mapsto \textbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$

```
let w = !c in
match w with
  None   => recv1 c
| Some v => free c; v
end
```

$\{w.\, \Phi\ w\}$

Duplicable propositions:

$\boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

Proof of Receive

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\{(c \mapsto \textbf{None}) \vee (\exists v.\, c \mapsto \textbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\{c \mapsto \textbf{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \textbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$

$\textbf{let}\ w = !\,c\ \textbf{in}$
$\textbf{match}\ w\ \textbf{with}$
$\quad \textbf{None} \Rightarrow \textbf{recv1}\ c$
$\mid \textbf{Some}\ v \Rightarrow \textbf{free}\ c;\ v$
$\textbf{end}$
$\{w.\, \Phi\ w\}$

Duplicable propositions:

$\boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

Proof of Receive

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\{(c \mapsto \textbf{None}) \vee (\exists v.\, c \mapsto \textbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\{c \mapsto \textbf{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \textbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$

let $w = !\,c$ **in**

$\{w = \textbf{None} * c \mapsto \textbf{None} * \mathsf{tok}\ \gamma_r\}$

match w **with**
  None $\Rightarrow$ **recv1** c
| **Some** v $\Rightarrow$ **free** c; v
end

$\{w.\, \Phi\ w\}$

Duplicable propositions:

$\boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

Proof of Receive

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\quad \{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\quad \{(c \mapsto \texttt{None}) \vee (\exists v.\ c \mapsto \texttt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\quad \{c \mapsto \texttt{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \texttt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\texttt{let}\ w = !\,c\ \texttt{in}$
$\quad \{w = \texttt{None} * c \mapsto \texttt{None} * \mathsf{tok}\ \gamma_r\}$
$\quad \{w = \texttt{None} * \texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\texttt{match}\ w\ \texttt{with}$
$\quad \texttt{None} \Rightarrow \texttt{recv1}\ c$
$\mid \texttt{Some}\ v \Rightarrow \texttt{free}\ c;\ v$
$\texttt{end}$
$\{w.\ \Phi\ w\}$

Duplicable propositions:

$\boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

Proof of Receive

Duplicable propositions:

$\boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\quad\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\quad\{(c \mapsto \texttt{None}) \vee (\exists v.\, c \mapsto \texttt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\quad\{c \mapsto \texttt{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \texttt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\texttt{let}\ w = {!}c\ \texttt{in}$
$\quad\{w = \texttt{None} * c \mapsto \texttt{None} * \mathsf{tok}\ \gamma_r\}\{w = \texttt{Some}\, v * c \mapsto \texttt{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\quad\{w = \texttt{None} * \texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\texttt{match}\ w\ \texttt{with}$
$\quad \texttt{None} \Rightarrow \texttt{recv1}\ c$
$\mid \texttt{Some}\ v \Rightarrow \texttt{free}\ c;\ v$
$\texttt{end}$
$\{w.\, \Phi\ w\}$

Proof of Receive

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\quad \{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\quad \{(c \mapsto \mathtt{None}) \vee (\exists v.\ c \mapsto \mathtt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\quad \{c \mapsto \mathtt{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \mathtt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\mathtt{let}\ w = !\,c\ \mathtt{in}$
$\quad \{w = \mathtt{None} * c \mapsto \mathtt{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathtt{Some}\ v * c \mapsto \mathtt{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\quad \{w = \mathtt{None} * \mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}\{w = \mathtt{Some}\ v * \mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * c \mapsto \mathtt{Some}\ v * \Phi\ v\}$
$\mathtt{match}\ w\ \mathtt{with}$
$\quad \mathtt{None} \Rightarrow \mathtt{recv1}\ c$
$\mid \mathtt{Some}\ v \Rightarrow \mathtt{free}\ c;\ v$
$\mathtt{end}$
$\{w.\ \Phi\ w\}$

Duplicable propositions:

$\boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

Proof of Receive

Duplicable propositions:

$\boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\quad\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\quad\{(c \mapsto \textbf{None}) \vee (\exists v.\, c \mapsto \textbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\quad\{c \mapsto \textbf{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \textbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\textbf{let}\ w = !c\ \textbf{in}$
$\quad\{w = \textbf{None} * c \mapsto \textbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \textbf{Some}\, v * c \mapsto \textbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\quad\{w = \textbf{None} * \texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}\{w = \textbf{Some}\, v * \texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * c \mapsto \textbf{Some}\, v * \Phi\ v\}$
$\quad\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * (w = \textbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\, w = \textbf{Some}\, v * c \mapsto \textbf{Some}\, v * \Phi\ v)\}$
$\textbf{match}\ w\ \textbf{with}$
$\quad \textbf{None} \Rightarrow \textbf{recv1}\ c$
$\mid \textbf{Some}\, v \Rightarrow \textbf{free}\ c;\ v$
$\textbf{end}$
$\{w.\, \Phi\ w\}$

Proof of Receive

Duplicable propositions:

$\boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\quad\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\quad\{(c \mapsto \textbf{None}) \vee (\exists v.\ c \mapsto \textbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\quad\{c \mapsto \textbf{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \textbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\textbf{let}\ w = !c\ \textbf{in}$
$\quad\{w = \textbf{None} * c \mapsto \textbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \textbf{Some}\ v * c \mapsto \textbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\quad\{w = \textbf{None} * \texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}\{w = \textbf{Some}\ v * \texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * c \mapsto \textbf{Some}\ v * \Phi\ v\}$
$\quad\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * (w = \textbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\ w = \textbf{Some}\ v * c \mapsto \textbf{Some}\ v * \Phi\ v)\}$
$\{(w = \textbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\ w = \textbf{Some}\ v * c \mapsto \textbf{Some}\ v * \Phi\ v)\}$
$\textbf{match}\ w\ \textbf{with}$
$\quad \textbf{None} \Rightarrow \textbf{recv1}\ c$
$\mid \textbf{Some}\ v \Rightarrow \textbf{free}\ c;\ v$
$\textbf{end}$
$\{w.\ \Phi\ w\}$

Proof of Receive

Duplicable propositions:

$\boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\quad\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\quad\{(c \mapsto \mathbf{None}) \vee (\exists v.\, c \mapsto \mathbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\quad\{c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \mathbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\mathtt{let}\ w = !c\ \mathtt{in}$
$\quad\{w = \mathbf{None} * c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\quad\{w = \mathbf{None} * \mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\, v * \mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * c \mapsto \mathbf{Some}\, v * \Phi\ v\}$
$\quad\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * (w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\, w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v)\}$
$\{(w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\, w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v)\}$
$\{w = \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v\}$

```
match w with
  None    ⇒ recv1 c
| Some v  ⇒ free c; v
end
```

$\{w.\, \Phi\ w\}$

Proof of Receive

Duplicable propositions:

$\boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\quad\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\quad\{(c \mapsto \mathbf{None}) \vee (\exists v.\ c \mapsto \mathbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\quad\{c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \mathbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\mathtt{let}\ w = {!c}\ \mathtt{in}$
$\quad\{w = \mathbf{None} * c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\quad\{w = \mathbf{None} * \mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\ v * \mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * c \mapsto \mathbf{Some}\ v * \Phi\ v\}$
$\quad\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * (w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\ w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v)\}$
$\{(w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\ w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v)\}$
$\{w = \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v\}$
$\mathtt{match}\ w\ \mathtt{with}$
$\quad\mathtt{None} \Rightarrow \{\mathsf{tok}\ \gamma_r\}\mathtt{recv1}\ c$
$\mid \mathtt{Some}\ v \Rightarrow \mathtt{free}\ c;\ v$
$\mathtt{end}$
$\{w.\ \Phi\ w\}$

Proof of Receive

Duplicable propositions:

$\boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\quad\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\quad\{(c \mapsto \mathbf{None}) \vee (\exists v.\ c \mapsto \mathbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\quad\{c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \mathbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\mathtt{let}\ w = !c\ \mathtt{in}$
$\quad\{w = \mathbf{None} * c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\quad\{w = \mathbf{None} * \texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\ v * \texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * c \mapsto \mathbf{Some}\ v * \Phi\ v\}$
$\quad\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * (w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\ w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v)\}$
$\{(w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\ w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v)\}$
$\{w = \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v\}$
$\mathtt{match}\ w\ \mathtt{with}$
$\quad\mathtt{None} \Rightarrow \{\mathsf{tok}\ \gamma_r\}\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}\mathtt{recv1}\ c$
$\mid \mathtt{Some}\ v \Rightarrow \mathtt{free}\ c;\ v$
$\mathtt{end}$
$\{w.\ \Phi\ w\}$

Proof of Receive

Duplicable propositions:

$\boxed{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\quad\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\quad\{(c \mapsto \mathbf{None}) \lor (\exists v.\ c \mapsto \mathbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\quad\{c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \mathbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\mathbf{let}\ w = \,!c\ \mathbf{in}$
$\quad\{w = \mathbf{None} * c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\quad\{w = \mathbf{None} * \texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\ v * \texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * c \mapsto \mathbf{Some}\ v * \Phi\ v\}$
$\quad\{\texttt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * (w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \lor (\exists v.\ w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v)\}$
$\{(w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \lor (\exists v.\ w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v)\}$
$\{w = \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\ v * c \mapsto \mathbf{Some}\ v * \Phi\ v\}$
$\mathbf{match}\ w\ \mathbf{with}$
$\quad \mathbf{None} \Rightarrow \{\mathsf{tok}\ \gamma_r\}\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}\mathbf{recv1}\ c\{w.\ \Phi\ w\}$
$\mid \mathbf{Some}\ v \Rightarrow \mathbf{free}\ c;\ v$
$\mathbf{end}$
$\{w.\ \Phi\ w\}$

Proof of Receive

Duplicable propositions:

$\boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

$$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$$
$$\{\mathsf{tok}\ \gamma_r\}$$
$$\quad\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$$
$$\quad\{(c \mapsto \mathbf{None}) \vee (\exists v.\, c \mapsto \mathbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$$
$$\quad\{c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \mathbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$$
$$\mathbf{let}\ w = !c\ \mathbf{in}$$
$$\quad\{w = \mathbf{None} * c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$$
$$\quad\{w = \mathbf{None} * \mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\, v * \mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * c \mapsto \mathbf{Some}\, v * \Phi\ v\}$$
$$\quad\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * (w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\, w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v)\}$$
$$\{(w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\, w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v)\}$$
$$\{w = \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v\}$$
$$\mathbf{match}\ w\ \mathbf{with}$$
$$\quad \mathbf{None} \quad \Rightarrow \{\mathsf{tok}\ \gamma_r\}\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}\mathbf{recv1}\ c\{w.\, \Phi\ w\}$$
$$\mid \mathbf{Some}\, v \Rightarrow \{c \mapsto \mathbf{Some}\, v * \Phi\ v\}\ \mathbf{free}\ c;\ v$$
$$\mathbf{end}$$
$$\{w.\, \Phi\ w\}$$

Proof of Receive

Duplicable propositions:

$\boxed{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi}$

$\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}$
$\{\mathsf{tok}\ \gamma_r\}$
$\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}$
$\{(c \mapsto \mathbf{None}) \vee (\exists v.\, c \mapsto \mathbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s) * \mathsf{tok}\ \gamma_r\}$
$\{c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{c \mapsto \mathbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\mathtt{let}\ w = {!c}\ \mathtt{in}$
$\{w = \mathbf{None} * c \mapsto \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v * \mathsf{tok}\ \gamma_s * \mathsf{tok}\ \gamma_r\}$
$\{w = \mathbf{None} * \mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\, v * \mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * c \mapsto \mathbf{Some}\, v * \Phi\ v\}$
$\{\mathtt{chan\_inv}\ \gamma_s\ \gamma_r\ c\ \Phi * (w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\, w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v)\}$
$\{(w = \mathbf{None} * \mathsf{tok}\ \gamma_r) \vee (\exists v.\, w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v)\}$
$\{w = \mathbf{None} * \mathsf{tok}\ \gamma_r\}\{w = \mathbf{Some}\, v * c \mapsto \mathbf{Some}\, v * \Phi\ v\}$
$\mathtt{match}\ w\ \mathtt{with}$
$\quad \mathtt{None} \Rightarrow \{\mathsf{tok}\ \gamma_r\}\{c \rightarrowtail (\mathsf{Recv}, \Phi)\}\mathbf{recv1}\ c\{w.\, \Phi\ w\}$
$\mid \mathtt{Some}\ v \Rightarrow \{c \mapsto \mathbf{Some}\, v * \Phi\ v\}\ \mathbf{free}\ c;\ v\{w.\, \Phi\ w\}$
$\mathtt{end}$
$\{w.\, \Phi\ w\}$

Questions?

Iris and Actris Beyond This Tutorial

Iris

- **Features:** Custom ghost state, persistent modality, Löb induction, ...
- **Technicalities:** Later modality, invariant masks, ghost updates, ...
- Website: `https://iris-project.org`

Iris and Actris Beyond This Tutorial

Iris

- **Features:** Custom ghost state, persistent modality, Löb induction, ...
- **Technicalities:** Later modality, invariant masks, ghost updates, ...
- Website: `https://iris-project.org`

Actris

- Recursive protocols (POPL'20)
- Semantic Session Type System (CPP'21)
- Subprotocols (cf. subtyping) (LMCS'22)
- Dependent separation protocol ghost state and rules (LMCS'22)
- Application to distributed systems (ICFP'23)
- Deadlock-freedom (POPL'24 on Thursday: 14:40)
- Website: `https://iris-project.org/actris`

Break (10 min!)

Time for Coq hacking session!