Machine-Checked Semantic Session Typing

Jonas Kastberg Hinrichsen, IT University of Copenhagen

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

27 May 2020
TU Delft, The Netherlands

Problem

Consider the following program:

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

Problem

Consider the following program:

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

Is it typeable?

Problem

Consider the following program:

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

Is it typeable? No

Problem

Consider the following program:

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

Is it typeable? No It violates the ownership discipline

Problem

Consider the following program:

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

Is it typeable? No It violates the ownership discipline

Is it safe?

Problem

Consider the following program:

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

Is it typeable?	No	It violates the ownership discipline
Is it safe?	Yes	

Problem

Consider the following program:

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

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

Problem

Consider the following program:

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

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

Problem

Consider the following program:

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

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

Problem

Consider the following program:

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

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

Shortcomings of Syntactic Typing

Adding ad-hoc typing rules is infeasible

Shortcomings of Syntactic Typing

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

Shortcomings of Syntactic Typing

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

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

Shortcomings of Syntactic Typing

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

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

Shortcomings of Syntactic Typing

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

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

Shortcomings of Syntactic Typing

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

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

Shortcomings of Syntactic Typing

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

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

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

Shortcomings of Syntactic Typing

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

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

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

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

Shortcomings of Syntactic Typing

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

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

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

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

Shortcomings of Syntactic Typing

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

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

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

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

Goal: Session type system where ad-hoc rules can be added modularly

Solution - Semantic Typing!

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

Solution - Semantic Typing!

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

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

Solution - Semantic Typing!

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

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

Solution - Semantic Typing!

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

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

  e does not get *stuck*

Solution - Semantic Typing!

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

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

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

Solution - Semantic Typing!

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

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

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

Solution - Semantic Typing!

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

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

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

$$\Gamma \vDash i : \mathbf{Z}$$

Solution - Semantic Typing!

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

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

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

$$\Gamma \vDash i : \mathbf{Z} \qquad\qquad \Gamma, (x : A) \vDash x : A$$

Solution - Semantic Typing!

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

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

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

$$\Gamma \vDash i : \mathbf{Z} \qquad\qquad \Gamma, (x : A) \vDash x : A \qquad\qquad \frac{\Gamma_1 \vDash e_1 : A_1 \qquad \Gamma_2 \vDash e_2 : A_2}{\Gamma_1 \circ \Gamma_2 \vDash (e_1, e_2) : A_1 \times A_2}$$

Solution - Semantic Typing!

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

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

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

$$\Gamma \vDash i : \mathbf{Z} \qquad\qquad \Gamma, (x : A) \vDash x : A \qquad\qquad \frac{\Gamma_1 \vDash e_1 : A_1 \qquad \Gamma_2 \vDash e_2 : A_2}{\Gamma_1 \circ \Gamma_2 \vDash (e_1, e_2) : A_1 \times A_2}$$

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

Solution - Semantic Typing!

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

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

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

$$\Gamma \vDash i : \mathbf{Z} \qquad \Gamma, (x : A) \vDash x : A \qquad \frac{\Gamma_1 \vDash e_1 : A_1 \qquad \Gamma_2 \vDash e_2 : A_2}{\Gamma_1 \circ \Gamma_2 \vDash (e_1, e_2) : A_1 \times A_2}$$

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

- **Soundness** is a consequence of the judgement definition

Semantic Typing

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

- Supports adding ad-hoc rules modularly

Semantic Typing using **Iris**

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

- Supports adding ad-hoc rules modularly

Iris [Iris project]

- **Higher-Order:** Recursion, Polymorphism
- **Concurrent:** Ghost state mechanisms to reason about concurrency
- **Separation Logic:** Implicit separation of linear ownership
- Mechanised in Coq

Key Idea

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

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

- Supports adding ad-hoc rules modularly

Iris [Iris project]

- **Higher-Order:** Recursion, Polymorphism
- **Concurrent:** Ghost state mechanisms to reason about concurrency
- **Separation Logic:** Implicit separation of linear ownership
- Mechanised in Coq

Actris [Hinrichsen et al., POPL'20]

- Dependent separation protocols (logical session types)
- Mechanised in Coq

Semantic approach to Session Typing

- Supports adding ad-hoc rules modularly
- Rich extensible type system for session types
  - Term and session type equi-recursion
  - Term and session type polymorphism
  - Term and (asynchronous) session type subtyping
  - Unique and shared reference types
  - Copyable types
  - Lock types
- Full mechanisation in Coq (https://gitlab.mpi-sws.org/iris/actris)

Semantic Approach to Session Typing

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

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

Message-passing is:

- **Binary:** Each channel have one pair endpoints
- **Asynchronous:** `send` does not block, two buffers per endpoint pair
- **Affine:** No `close` expression, channels can be thrown away

Semantic Term Types

Types as Iris predicates:

$$\mathsf{Type}_{\bigstar} \triangleq \mathsf{Val} \to \mathsf{iProp}$$

Semantic Term Types

Types as Iris predicates:

$$\begin{aligned} \mathsf{Type}_{\bigstar} &\triangleq \mathsf{Val} \to \mathsf{iProp} \\ \mathbf{Z} &\triangleq \lambda\, w.\, w \in \mathbb{Z} \end{aligned}$$

Semantic Term Types

Types as Iris predicates:

$$\begin{aligned} \mathsf{Type}_\bigstar &\triangleq \mathsf{Val} \to \mathsf{iProp} \\ \mathbf{Z} &\triangleq \lambda\, w.\, w \in \mathbb{Z} \\ A_1 \times A_2 &\triangleq \lambda\, w.\, \exists w_1, w_2.\, w = (w_1, w_2) * \triangleright(A_1\, w_1) * \triangleright(A_2\, w_2) \end{aligned}$$

Semantic Term Types

Types as Iris predicates:

$$
\begin{aligned}
\mathrm{Type}_{\bigstar} &\triangleq \mathsf{Val} \to \mathrm{iProp} \\
\mathbf{Z} &\triangleq \lambda\, w.\, w \in \mathbb{Z} \\
A_1 \times A_2 &\triangleq \lambda\, w.\, \exists w_1, w_2.\, w = (w_1, w_2) * \triangleright(A_1\, w_1) * \triangleright(A_2\, w_2) \\
\mathtt{ref}_{\mathtt{uniq}}\, A &\triangleq \lambda\, w.\, \exists v.\, w \in \mathsf{Loc} * (w \mapsto v) * \triangleright(A\, v)
\end{aligned}
$$

Semantic Term Types

Types as Iris predicates:

$$
\begin{aligned}
\mathsf{Type}_{\bigstar} &\triangleq \mathsf{Val} \to \mathsf{iProp} \\
\mathbf{Z} &\triangleq \lambda\, w.\, w \in \mathbb{Z} \\
A_1 \times A_2 &\triangleq \lambda\, w.\, \exists w_1, w_2.\, w = (w_1, w_2) * \triangleright(A_1\, w_1) * \triangleright(A_2\, w_2) \\
\mathtt{ref}_{\mathtt{uniq}}\, A &\triangleq \lambda\, w.\, \exists v.\, w \in \mathsf{Loc} * (w \mapsto v) * \triangleright(A\, v) \\
A \multimap B &\triangleq \lambda\, w.\, \forall v.\, \triangleright(A\, v) \mathrel{-\!\!*} \mathsf{wp}\,(w\ v)\,\{B\}
\end{aligned}
$$

$\mathsf{wp}\ e\ \{v.\Phi\}$ dictates e does not get *stuck* and if e reduces to a value v then $\Phi\ v$ holds

Semantic Term Types

Types as Iris predicates:

$$\begin{aligned}
\mathsf{Type}_{\bigstar} &\triangleq \mathsf{Val} \to \mathsf{iProp} \\
\mathbf{Z} &\triangleq \lambda\, w.\, w \in \mathbb{Z} \\
A_1 \times A_2 &\triangleq \lambda\, w.\, \exists w_1, w_2.\, w = (w_1, w_2) * \triangleright(A_1\, w_1) * \triangleright(A_2\, w_2) \\
\mathtt{ref}_{\mathtt{uniq}}\, A &\triangleq \lambda\, w.\, \exists v.\, w \in \mathsf{Loc} * (w \mapsto v) * \triangleright(A\, v) \\
A \multimap B &\triangleq \lambda\, w.\, \forall v.\, \triangleright(A\, v) \mathrel{-\!\!*} \mathsf{wp}\, (w\ \ v)\, \{B\}
\end{aligned}$$

Judgement as Iris weakest precondition:

$$\begin{aligned}
\Gamma \vDash \sigma &\triangleq \mathop{\ast}_{(x,A)\in\Gamma}.\ \exists v.\, (x, v) \in \sigma * A\, v \\
\Gamma \vDash e : A \dashv \Gamma' &\triangleq \forall \sigma.\, (\Gamma \vDash \sigma) \mathrel{-\!\!*} \mathsf{wp}\, e[\sigma]\, \{v. A\, v * (\Gamma' \vDash \sigma)\}
\end{aligned}$$

---

wp $e\,\{v.\Phi\}$ dictates e does not get *stuck* and if e reduces to a value v then $\Phi\ v$ holds

Semantic Term Types

Types as Iris predicates:

$$\begin{aligned}
\mathsf{Type}_\bigstar &\triangleq \mathsf{Val} \to \mathsf{iProp} \\
\mathbf{Z} &\triangleq \lambda\, w.\, w \in \mathbb{Z} \\
A_1 \times A_2 &\triangleq \lambda\, w.\, \exists w_1, w_2.\, w = (w_1, w_2) * \triangleright(A_1\, w_1) * \triangleright(A_2\, w_2) \\
\mathtt{ref}_{\mathtt{uniq}}\, A &\triangleq \lambda\, w.\, \exists v.\, w \in \mathsf{Loc} * (w \mapsto v) * \triangleright(A\, v) \\
A \multimap B &\triangleq \lambda\, w.\, \forall v.\, \triangleright(A\, v) \mathrel{-\!\!*} \mathsf{wp}\,(w\ \ v)\,\{B\}
\end{aligned}$$

Judgement as Iris weakest precondition:

$$\begin{aligned}
\Gamma \vDash \sigma &\triangleq \mathop{\Huge *}_{(x,A)\in\Gamma}.\ \exists v.\, (x, v) \in \sigma * A\, v \\
\Gamma \vDash e : A \dashv \Gamma' &\triangleq \forall \sigma.\, (\Gamma \vDash \sigma) \mathrel{-\!\!*} \mathsf{wp}\; e[\sigma]\,\{v. A\, v * (\Gamma' \vDash \sigma)\}
\end{aligned}$$

Type soundness: If $\varnothing \vDash e : A \dashv \Gamma$ then e does not get stuck

- Consequence of Iris's adequacy of weakest precondition

$\mathsf{wp}\ e\,\{v.\Phi\}$ dictates e does not get *stuck* and if e reduces to a value v then $\Phi\ v$ holds

But what about session types?

Semantic Session Types - Definitions

Session types as ?:

$$
\begin{aligned}
\mathrm{Type}_{\blacklozenge} &\triangleq ? \\
\mathbf{!}A.\,S &\triangleq ? \\
\mathbf{?}A.\,S &\triangleq ? \\
\mathtt{end} &\triangleq ?
\end{aligned}
\qquad\qquad
\begin{aligned}
\mathrm{Type}_{\bigstar} &\triangleq \mathrm{Val} \to \mathrm{iProp} \\
\mathtt{chan}\ S &\triangleq \lambda\, w.\, ?
\end{aligned}
$$

Requires capturing:

- **Linearity** of channel endpoint ownership
- **Delegation** of linear types / channels
- **Session fidelity** of communicated messages

Actris Dependent Separation Protocols - Definitions

Session type inspired protocols for functional correctness

	Dependent separation protocols	Syntactic session types
Symbols	$prot \triangleq\ \mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot \mid \mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot \mid \mathbf{end}$	$S \triangleq \mathbf{!}A.\,S \mid \mathbf{?}A.\,S \mid \mathtt{end} \mid \ldots$
Example	$\mathbf{?}\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,\mathbf{?}\,(y:\mathbb{Z})\,\langle y\rangle\{x+y=10\}.\,\mathbf{end}$	$\mathbf{?Z}.\,\mathbf{?Z}.\,\mathtt{end}$
Duality	$\overline{\mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot} = \mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,\overline{prot}$ $\overline{\mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot} = \mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,\overline{prot}$ $\overline{\mathbf{end}} = \mathbf{end}$	$\overline{\mathbf{!}A.\,S} = \mathbf{?}A.\,\overline{S}$ $\overline{\mathbf{?}A.\,S} = \mathbf{!}A.\,\overline{S}$ $\overline{\mathtt{end}} = \mathtt{end}$
Usage	$c \rightarrowtail prot$	$c : S$

Actris Dependent Separation Protocols - Rules

	Dependent separation protocols	**Syntactic session types**
New	$\text{wp}\ \texttt{new\_chan}\ ()\ \{(c, c').\ c \rightarrowtail \mathit{prot} * c' \rightarrowtail \overline{\mathit{prot}}\}$	$\Gamma \vdash \texttt{new\_chan}\ () : S \times \overline{S} \dashv \Gamma$
Send	$c \rightarrowtail \mathbf{!}\, \vec{x}{:}\vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot} \mathrel{-\!\!*} \triangleright P[\vec{t}/\vec{x}] \mathrel{-\!\!*}$ $\text{wp}\ \texttt{send}\ c\ (v[\vec{t}/\vec{x}])\ \{c \rightarrowtail \mathit{prot}[\vec{t}/\vec{x}]\}$	$\Gamma, (c : \mathbf{!}A.\, S), (x : A) \vdash \texttt{send}\ c\ x : \mathbf{1} \dashv \Gamma, (c : S)$
Recv	$c \rightarrowtail \mathbf{?}\, \vec{x}{:}\vec{\tau}\, \langle v \rangle \{\triangleright P\}.\, \mathit{prot} \mathrel{-\!\!*}$ $\text{wp}\ \texttt{recv}\ c \left\{ w.\ \exists \vec{y}.\ \begin{array}{l} (w = v[\vec{y}/\vec{x}]) * \\ c \rightarrowtail \mathit{prot}[\vec{y}/\vec{x}] * P[\vec{y}/\vec{x}] \end{array} \right\}$	$\Gamma, (c : \mathbf{?}A.\, S) \vdash \texttt{recv}\ c : A \dashv \Gamma, (c : S)$

Dependent separation protocols:

Symbols:	$\mathbf{!}\, \vec{x}{:}\vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot} \quad \mid \quad \mathbf{?}\, \vec{x}{:}\vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot} \quad \mid \quad \mathbf{end}$
Example:	$\mathbf{?}\, (x : \mathbb{Z})\, \langle x \rangle \{\text{True}\}.\ \mathbf{?}\, (y : \mathbb{Z})\, \langle y \rangle \{x + y = 10\}.\ \mathbf{end}$
Ownership:	$c \rightarrowtail \mathit{prot}$

Semantic Session Types - Definitions

Session types as dependent separation protocols:

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

Dependent separation protocols:

Symbols:	$\mathbf{!}\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\, \mathit{prot}$ | $\mathbf{?}\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\, \mathit{prot}$ | **end**
Example:	$\mathbf{?}\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\, \mathbf{?}\,(y:\mathbb{Z})\,\langle y\rangle\{x+y=10\}.\, \mathbf{end}$
Ownership:	$c \rightarrowtail \mathit{prot}$

Semantic Session Types - Rules

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

	Semantic session types	Dependent separation protocols
New	$\Gamma \vDash \texttt{new\_chan}\ () : \texttt{chan}\ S \times \texttt{chan}\ \overline{S} \dashv \Gamma$	$\text{wp}\ \texttt{new\_chan}\ ()\ \{(c, c').\ c \rightarrowtail \mathit{prot} * c' \rightarrowtail \overline{\mathit{prot}}\}$
Send	$\Gamma, (c : \texttt{chan}\ !A.\ S), (x : A) \vDash \texttt{send}\ c\ x : \mathbf{1} \dashv$ $\Gamma, (c : \texttt{chan}\ S)$	$c \rightarrowtail \mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\ \mathit{prot} \twoheadrightarrow\, \triangleright P[\vec{t}/\vec{x}] \twoheadrightarrow$ $\text{wp}\ \texttt{send}\ c\ (v[\vec{t}/\vec{x}])\ \{c \rightarrowtail \mathit{prot}[\vec{t}/\vec{x}]\}$
Recv	$\Gamma, (c : \texttt{chan}\ (?A.\ S)) \vDash \texttt{recv}\ c : A \dashv$ $\Gamma, (c : \texttt{chan}\ S)$	$c \rightarrowtail \mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{\triangleright P\}.\ \mathit{prot} \twoheadrightarrow$ $\text{wp}\ \texttt{recv}\ c \left\{ w.\ \exists \vec{y}.\ \begin{array}{l}(w = v[\vec{y}/\vec{x}]) * \\ c \rightarrowtail \mathit{prot}[\vec{y}/\vec{x}] * P[\vec{y}/\vec{x}]\end{array} \right\}$

Dependent separation protocols:

Symbols:	$\mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\ \mathit{prot} \quad \mid \quad \mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\ \mathit{prot} \quad \mid \quad \mathbf{end}$
Example:	$\mathbf{?}\,(x{:}\mathbb{Z})\ \langle x\rangle\{\text{True}\}.\ \mathbf{?}\,(y{:}\mathbb{Z})\ \langle y\rangle\{x + y = 10\}.\ \mathbf{end}$
Ownership:	$c \rightarrowtail \mathit{prot}$

Typing the Untypeable

Typing the Untypeable Program

The rule:

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

Is just another lemma

Typing the Untypeable Program

The rule:

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

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

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

Typing the Untypeable Program

The rule:

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

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

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

And then using Iris's ghost state machinery!

Typing the Untypeable Program

The rule:

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

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

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

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

Typing the Untypeable Program

The rule:

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

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

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

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

Adding ad-hoc rules for safe untypeable programs

Typing the Untypeable Program

The rule:

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

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

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

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

Adding ad-hoc rules for safe untypeable programs ✓

Typing the Untypeable Program

The rule:

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

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

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

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

Adding ad-hoc rules for safe untypeable programs ✓
Extensibility of type system

Typing the Untypeable Program

The rule:

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

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

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

And then using Iris's ghost state machinery!$_{\text{Beyond the scope of this talk}}$

Adding ad-hoc rules for safe untypeable programs	✓
Extensibility of type system	?

Overview of features

Iris and **Actris** gives immediate rise to many type features

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)
Function types	Wand ($-\!\!*$) and weakest precondition ($\mathrm{wp}\ e\ \{\Phi\}$)

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)
Function types	Wand ($-\!\!*$) and weakest precondition ($\mathrm{wp}\ e\ \{\Phi\}$)
Session types	Actris dependent separation protocols (iProto)

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)
Function types	Wand ($-\!*$) and weakest precondition ($\mathrm{wp}\ e\ \{\Phi\}$)
Session types	Actris dependent separation protocols (iProto)
Copyable types	Persistent modality ($\Box$)

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)
Function types	Wand ($-\!*$) and weakest precondition ($\mathrm{wp}\ e\ \{\Phi\}$)
Session types	Actris dependent separation protocols (iProto)
Copyable types	Persistent modality ($\square$)
Shared references	Invariants ($\boxed{P}$)

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)
Function types	Wand ($-\!\!*$) and weakest precondition ($\text{wp } e \, \{\Phi\}$)
Session types	Actris dependent separation protocols (iProto)
Copyable types	Persistent modality ($\square$)
Shared references	Invariants ($\boxed{P}$)
Lock types	Invariants ($\boxed{P}$) and Iris's lock library

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)
Function types	Wand ($-\!\!*$) and weakest precondition ($\mathrm{wp}\ e\ \{\Phi\}$)
Session types	Actris dependent separation protocols (iProto)
Copyable types	Persistent modality ($\Box$)
Shared references	Invariants ($\boxed{P}$)
Lock types	Invariants ($\boxed{P}$) and Iris's lock library
Session choice types	Actris dependent separation protocols (iProto)

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)
Function types	Wand ($-\!\!*$) and weakest precondition ($\text{wp}\ e\ \{\Phi\}$)
Session types	Actris dependent separation protocols (iProto)
Copyable types	Persistent modality ($\square$)
Shared references	Invariants ($\boxed{P}$)
Lock types	Invariants ($\boxed{P}$) and Iris's lock library
Session choice types	Actris dependent separation protocols (iProto)
Recursion	Guarded step-indexed recursion ($\triangleright$)

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)
Function types	Wand ($-\!*$) and weakest precondition ($\mathrm{wp}\ e \ \{\Phi\}$)
Session types	Actris dependent separation protocols (iProto)
Copyable types	Persistent modality ($\Box$)
Shared references	Invariants ($\boxed{P}$)
Lock types	Invariants ($\boxed{P}$) and Iris's lock library
Session choice types	Actris dependent separation protocols (iProto)
Recursion	Guarded step-indexed recursion ($\triangleright$)
Term polymorphism	Higher-order impredicative quantifiers

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)
Function types	Wand ($-\!*$) and weakest precondition ($\mathrm{wp}\ e\ \{\Phi\}$)
Session types	Actris dependent separation protocols (iProto)
Copyable types	Persistent modality ($\Box$)
Shared references	Invariants ($\boxed{P}$)
Lock types	Invariants ($\boxed{P}$) and Iris's lock library
Session choice types	Actris dependent separation protocols (iProto)
Recursion	Guarded step-indexed recursion ($\triangleright$)
Term polymorphism	Higher-order impredicative quantifiers
Session polymorphism	Higher-order impredicative protocols binders

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)
Function types	Wand ($-\!\!*$) and weakest precondition ($\mathsf{wp}\ e\ \{\Phi\}$)
Session types	Actris dependent separation protocols (iProto)
Copyable types	Persistent modality ($\square$)
Shared references	Invariants ($\boxed{P}$)
Lock types	Invariants ($\boxed{P}$) and Iris's lock library
Session choice types	Actris dependent separation protocols (iProto)
Recursion	Guarded step-indexed recursion ($\triangleright$)
Term polymorphism	Higher-order impredicative quantifiers
Session polymorphism	Higher-order impredicative protocols binders
Term subtyping	Predicates closed under wand ($\forall v.\ A_1\ v -\!\!*\ A_2\ v$)

Overview of features

Iris and **Actris** gives immediate rise to many type features

Linear products	Separation Conjunction ($*$)
Unique references	Points-to connective ($\ell \mapsto v$)
Function types	Wand ($-\!\!*$) and weakest precondition ($\text{wp } e \, \{\Phi\}$)
Session types	Actris dependent separation protocols (iProto)
Copyable types	Persistent modality ($\Box$)
Shared references	Invariants ($\boxed{P}$)
Lock types	Invariants ($\boxed{P}$) and Iris's lock library
Session choice types	Actris dependent separation protocols (iProto)
Recursion	Guarded step-indexed recursion ($\triangleright$)
Term polymorphism	Higher-order impredicative quantifiers
Session polymorphism	Higher-order impredicative protocols binders
Term subtyping	Predicates closed under wand ($\forall v.\, A_1 \; v -\!\!* A_2 \; v$)
Session subtyping	Actris 2.0 subprotocols ($\sqsubseteq$)

Overview of features - Definitions

Copyable types: $\mathtt{copy}\, A \triangleq \lambda\, w.\ \Box(A\, w)$

Overview of features - Definitions

Copyable types: $\texttt{copy}\, A \triangleq \lambda\, w.\ \Box(A\, w)$

Shared references: $\texttt{ref}_{\texttt{shr}}\ A \triangleq \lambda\, w.\, (w \in \mathsf{Loc}) * \boxed{\exists v.\, (w \mapsto v) * \Box(A\, v)}$

Overview of features - Definitions

Copyable types: $\texttt{copy}\, A \triangleq \lambda\, w.\ \Box(A\, w)$

Shared references: $\texttt{ref}_{\texttt{shr}}\ A \triangleq \lambda\, w.\, (w \in \mathsf{Loc}) * \boxed{\exists v.\, (w \mapsto v) * \Box(A\, v)}$

Lock types:

$$\texttt{mutex}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v))$$

$$\overline{\texttt{mutex}}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v)) * (\ell \mapsto -)$$

Overview of features - Definitions

Copyable types: $\texttt{copy}\, A \triangleq \lambda\, w.\ \Box(A\, w)$

Shared references: $\texttt{ref}_{\texttt{shr}}\ A \triangleq \lambda\, w.\, (w \in \mathsf{Loc}) * \boxed{\exists v.\, (w \mapsto v) * \Box(A\, v)}$

Lock types:

$$\texttt{mutex}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v))$$
$$\overline{\texttt{mutex}}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v)) * (\ell \mapsto -)$$

Session choice:

$$\oplus\{\vec{S}\} \triangleq \mathbf{!}\,(I : \mathbb{Z})\,\langle I\rangle\big\{\triangleright(I \in \mathrm{dom}(\vec{S}))\big\}.\, \vec{S}(I)$$
$$\&\{\vec{S}\} \triangleq \mathbf{?}\,(I : \mathbb{Z})\,\langle I\rangle\big\{\triangleright(I \in \mathrm{dom}(\vec{S}))\big\}.\, \vec{S}(I)$$

Overview of features - Definitions

Copyable types: $\texttt{copy}\, A \triangleq \lambda\, w.\, \Box(A\, w)$

Shared references: $\texttt{ref}_{\texttt{shr}}\ A \triangleq \lambda\, w.\, (w \in \mathsf{Loc}) * \boxed{\exists v.\, (w \mapsto v) * \Box(A\, v)}$

Lock types:

$$\texttt{mutex}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v))$$

$$\overline{\texttt{mutex}}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v)) * (\ell \mapsto -)$$

Session choice:

$$\oplus\{\vec{S}\} \triangleq \mathbf{!}\,(I : \mathbb{Z})\,\langle I\rangle\big\{\triangleright(I \in \mathrm{dom}(\vec{S}))\big\}.\, \vec{S}(I)$$

$$\&\{\vec{S}\} \triangleq \mathbf{?}\,(I : \mathbb{Z})\,\langle I\rangle\big\{\triangleright(I \in \mathrm{dom}(\vec{S}))\big\}.\, \vec{S}(I)$$

Recursion: $\mu\,(X : k).\, K \triangleq \mu\,(X : \mathsf{Type}_k).\, K \qquad (K \text{ must be contractive in } X)$

Overview of features - Definitions

Copyable types: $\texttt{copy}\, A \triangleq \lambda\, w.\, \Box(A\, w)$

Shared references: $\texttt{ref}_{\texttt{shr}}\, A \triangleq \lambda\, w.\, (w \in \mathsf{Loc}) * \boxed{\exists v.\, (w \mapsto v) * \Box(A\, v)}$

Lock types:

$$\texttt{mutex}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\; lk\; (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v))$$
$$\overline{\texttt{mutex}}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\; lk\; (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v)) * (\ell \mapsto -)$$

Session choice:

$$\oplus\{\vec{S}\} \triangleq \mathbf{!}\, (I : \mathbb{Z})\, \langle I \rangle \big\{\triangleright(I \in \mathrm{dom}(\vec{S}))\big\}.\, \vec{S}(I)$$
$$\&\{\vec{S}\} \triangleq \mathbf{?}\, (I : \mathbb{Z})\, \langle I \rangle \big\{\triangleright(I \in \mathrm{dom}(\vec{S}))\big\}.\, \vec{S}(I)$$

Recursion: $\mu\, (X : k).\, K \triangleq \mu\, (X : \mathsf{Type}_k).\, K \qquad (K \text{ must be contractive in } X)$

Polymorphism:

$$\forall (X : k).\, A \triangleq \lambda\, w.\, \forall (X : \mathsf{Type}_k).\, \mathsf{wp}\; w\, ()\, \{A\}$$
$$\exists (X : k).\, A \triangleq \lambda\, w.\, \exists (X : \mathsf{Type}_k).\, \triangleright(A\, w)$$

Overview of features - Definitions

Copyable types: $\texttt{copy}\, A \triangleq \lambda\, w.\ \Box(A\, w)$

Shared references: $\texttt{ref}_{\texttt{shr}}\ A \triangleq \lambda\, w.\, (w \in \mathsf{Loc}) * \boxed{\exists v.\, (w \mapsto v) * \Box(A\, v)}$

Lock types:

$$\texttt{mutex}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v))$$

$$\overline{\texttt{mutex}}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v)) * (\ell \mapsto -)$$

Session choice:

$$\oplus\{\vec{S}\} \triangleq \mathbf{!}\,(I : \mathbb{Z})\,\langle I \rangle \big\{\triangleright(I \in \mathrm{dom}(\vec{S}))\big\}.\, \vec{S}(I)$$

$$\&\{\vec{S}\} \triangleq \mathbf{?}\,(I : \mathbb{Z})\,\langle I \rangle \big\{\triangleright(I \in \mathrm{dom}(\vec{S}))\big\}.\, \vec{S}(I)$$

Recursion: $\mu\,(X : k).\, K \triangleq \mu\,(X : \mathsf{Type}_k).\, K \qquad (K \text{ must be contractive in } X)$

Polymorphism:

$$\forall (X : k).\, A \triangleq \lambda\, w.\, \forall (X : \mathsf{Type}_k).\, \mathsf{wp}\ w\ ()\ \{A\}$$

$$\exists (X : k).\, A \triangleq \lambda\, w.\, \exists (X : \mathsf{Type}_k).\ \triangleright(A\, w)$$

$$\mathbf{!}_{\vec{X}:\vec{k}}\, A.\, S \triangleq \mathbf{!}\,(\vec{X} : \vec{\mathsf{Type}_k})(v : \mathsf{Val})\,\langle v \rangle \{\triangleright(A\, v)\}.\, S$$

$$\mathbf{?}_{\vec{X}:\vec{k}}\, A.\, S \triangleq \mathbf{?}\,(\vec{X} : \vec{\mathsf{Type}_k})(v : \mathsf{Val})\,\langle v \rangle \{\triangleright(A\, v)\}.\, S$$

Overview of features - Definitions

Copyable types: $\texttt{copy}\, A \triangleq \lambda\, w.\ \Box(A\, w)$

Shared references: $\texttt{ref}_{\texttt{shr}}\ A \triangleq \lambda\, w.\, (w \in \mathrm{Loc}) * \boxed{\exists v.\, (w \mapsto v) * \Box(A\, v)}$

Lock types:

$$\texttt{mutex}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v))$$

$$\overline{\texttt{mutex}}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v)) * (\ell \mapsto -)$$

Session choice:

$$\oplus\{\vec{S}\} \triangleq \mathbf{!}\,(l : \mathbb{Z})\,\langle l\rangle\big\{\triangleright(l \in \mathrm{dom}(\vec{S}))\big\}.\ \vec{S}(l)$$

$$\&\{\vec{S}\} \triangleq \mathbf{?}\,(l : \mathbb{Z})\,\langle l\rangle\big\{\triangleright(l \in \mathrm{dom}(\vec{S}))\big\}.\ \vec{S}(l)$$

Recursion: $\mu\,(X : k).\ K \triangleq \mu\,(X : \mathrm{Type}_k).\ K \qquad (K \text{ must be contractive in } X)$

Polymorphism:

$$\forall (X : k).\ A \triangleq \lambda\, w.\, \forall (X : \mathrm{Type}_k).\ \mathrm{wp}\ w\,()\ \{A\}$$

$$\exists (X : k).\ A \triangleq \lambda\, w.\, \exists (X : \mathrm{Type}_k).\ \triangleright(A\, w)$$

$$\mathbf{!}_{\vec{X}:\vec{k}}\, A.\ S \triangleq \mathbf{!}\,(\vec{X} : \vec{\mathrm{Type}}_k)(v : \mathrm{Val})\,\langle v\rangle\{\triangleright(A\, v)\}.\ S$$

$$\mathbf{?}_{\vec{X}:\vec{k}}\, A.\ S \triangleq \mathbf{?}\,(\vec{X} : \vec{\mathrm{Type}}_k)(v : \mathrm{Val})\,\langle v\rangle\{\triangleright(A\, v)\}.\ S$$

Term subtyping: $A <: B \triangleq \forall v.\ A\ v \mathrel{-\!\!*} B\ v$

Overview of features - Definitions

Copyable types: $\texttt{copy}\, A \triangleq \lambda\, w.\ \Box(A\, w)$

Shared references: $\texttt{ref}_{\texttt{shr}}\ A \triangleq \lambda\, w.\, (w \in \mathsf{Loc}) * \boxed{\exists v.\, (w \mapsto v) * \Box(A\, v)}$

Lock types:

$$\texttt{mutex}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v))$$
$$\overline{\texttt{mutex}}\, A \triangleq \lambda\, w.\, \exists lk, \ell.\, (w = (lk, \ell)) * \texttt{isLock}\ lk\ (\exists v.\, (\ell \mapsto u) * \triangleright(A\, v)) * (\ell \mapsto -)$$

Session choice:

$$\oplus\{\vec{S}\} \triangleq \mathbf{!}\, (I : \mathbb{Z})\, \langle I \rangle \big\{\triangleright(I \in \mathrm{dom}(\vec{S}))\big\}.\, \vec{S}(I)$$
$$\&\{\vec{S}\} \triangleq \mathbf{?}\, (I : \mathbb{Z})\, \langle I \rangle \big\{\triangleright(I \in \mathrm{dom}(\vec{S}))\big\}.\, \vec{S}(I)$$

Recursion: $\mu\,(X : k).\, K \triangleq \mu\,(X : \mathsf{Type}_k).\, K \qquad (K \text{ must be contractive in } X)$

Polymorphism:

$$\forall (X : k).\, A \triangleq \lambda\, w.\, \forall (X : \mathsf{Type}_k).\, \mathsf{wp}\ w\, ()\, \{A\}$$
$$\exists (X : k).\, A \triangleq \lambda\, w.\, \exists (X : \mathsf{Type}_k).\ \triangleright(A\, w)$$
$$\mathbf{!}_{\vec{X}:\vec{k}}\, A.\, S \triangleq \mathbf{!}\, (\vec{X} : \vec{\mathsf{Type}_k})(v : \mathsf{Val})\, \langle v \rangle \{\triangleright(A\, v)\}.\, S$$
$$\mathbf{?}_{\vec{X}:\vec{k}}\, A.\, S \triangleq \mathbf{?}\, (\vec{X} : \vec{\mathsf{Type}_k})(v : \mathsf{Val})\, \langle v \rangle \{\triangleright(A\, v)\}.\, S$$

Term subtyping: $A <: B \triangleq \forall v.\, A\ v \mathrel{-\!\!*} B\ v$

Session subtyping: $S_1 <: S_2 \triangleq S_1 \sqsubseteq S_2$

Concluding Remarks

Concluding Remarks

Semantic typing and separation logic is a good fit for session types

- **Linearity** is implicit from separation logic
- **Binders** are inherited from meta-logic

Using a strong logic gives immediate rise to advanced features

- **Iris:** Polymorphism, recursion, locks and more
- **Actris:** Session types, session polymorphism, session subtyping

Sources:

- Paper (https://iris-project.org/pdfs/2020-actris2-submission.pdf)
- Mechanisation in Coq (https://gitlab.mpi-sws.org/iris/actris)

Subtyping

Semantic Asynchronous Session Subtyping

Conventional subtyping:

$$\frac{S_1 <: S_2}{\texttt{chan}\ S_1 <: \texttt{chan}\ S_2} \qquad \frac{A_2 <: A_1 \qquad S_1 <: S_2}{\mathbf{!}A_1.\,S_1 <: \mathbf{!}A_2.\,S_2} \qquad \frac{A_1 <: A_2 \qquad S_1 <: S_2}{\mathbf{?}A_1.\,S_1 <: \mathbf{?}A_2.\,S_2}$$

Semantic Asynchronous Session Subtyping

Conventional subtyping:

$$\frac{S_1 <: S_2}{\texttt{chan}\ S_1 <: \texttt{chan}\ S_2} \qquad \frac{A_2 <: A_1 \qquad S_1 <: S_2}{\mathbf{!}A_1.\, S_1 <: \mathbf{!}A_2.\, S_2} \qquad \frac{A_1 <: A_2 \qquad S_1 <: S_2}{\mathbf{?}A_1.\, S_1 <: \mathbf{?}A_2.\, S_2}$$

Asynchronous Subtyping:

$$\mathbf{?}A_1.\, \mathbf{!}A_2.\, S <: \mathbf{!}A_2.\, \mathbf{?}A_1.\, S$$

Conventional subtyping:

$$\frac{S_1 <: S_2}{\texttt{chan}\ S_1 <: \texttt{chan}\ S_2} \qquad \frac{A_2 <: A_1 \qquad S_1 <: S_2}{\mathbf{!}A_1.\, S_1 <: \mathbf{!}A_2.\, S_2} \qquad \frac{A_1 <: A_2 \qquad S_1 <: S_2}{\mathbf{?}A_1.\, S_1 <: \mathbf{?}A_2.\, S_2}$$

Asynchronous Subtyping:

$$\mathbf{?}A_1.\, \mathbf{!}A_2.\, S <: \mathbf{!}A_2.\, \mathbf{?}A_1.\, S$$

Polymorphism subtyping:

$$\begin{array}{rcl} \mathbf{!}_{(\vec{X}:\vec{k})}\, A.\, S & <: & \mathbf{!}A[\vec{K}/\vec{X}].\, S[\vec{K}/\vec{X}] \\ \mathbf{?}A[\vec{K}/\vec{X}].\, S[\vec{K}/\vec{X}] & <: & \mathbf{?}_{(\vec{X}:\vec{k})}\, A.\, S \end{array} \qquad \frac{S_1 <: \mathbf{!}A.\, S_2}{S_1 <: \mathbf{!}_{(\vec{X}:\vec{k})} A.\, S_2} \qquad \frac{\mathbf{?}A.\, S_1 <: S_2}{\mathbf{?}_{(\vec{X}:\vec{k})} A.\, S_1 <: S_2}$$

Semantic Asynchronous Session Subtyping - Example

Goal:

$$\mu\,(\mathit{rec} : \blacklozenge).\, \mathbf{!}_{(X,Y:\bigstar)}\,(X \multimap Y).\,\mathbf{!}X.\,\mathbf{?}Y.\,\mathit{rec} <: \mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,\mathbf{!}X_1.\,\mathbf{!}(X_2 \multimap \mathbf{Z}).\,\mathbf{!}X_2.\,\mathbf{?B}.\,\mathbf{?Z}.\,\mathit{rec}$$

Semantic Asynchronous Session Subtyping - Example

Goal:

$$\mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X,Y:\bigstar)}\,(X \multimap Y).\,\mathbf{!}X.\,\mathbf{?}Y.\,\mathit{rec} <: \mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,\mathbf{!}X_1.\,\mathbf{!}(X_2 \multimap \mathbf{Z}).\,\mathbf{!}X_2.\,\mathbf{?B}.\,\mathbf{?Z}.\,\mathit{rec}$$

Derivation:

$$\mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X,Y:\bigstar)}\,(X \multimap Y).\,\mathbf{!}X.\,\mathbf{?}Y.\,\mathit{rec}$$

Semantic Asynchronous Session Subtyping - Example

Goal:

$$\mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X,Y:\bigstar)}\,(X \multimap Y).\,\mathbf{!}X.\,\mathbf{?}Y.\,\mathit{rec} <: \mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,\mathbf{!}X_1.\,\mathbf{!}(X_2 \multimap \mathbf{Z}).\,\mathbf{!}X_2.\,\mathbf{?B}.\,\mathbf{?Z}.\,\mathit{rec}$$

Derivation:

$$\begin{aligned} &\mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X,Y:\bigstar)}\,(X \multimap Y).\,\mathbf{!}X.\,\mathbf{?}Y.\,\mathit{rec} \\ <: \;&\mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X_1,Y_1:\bigstar)}\,(X_1 \multimap Y_1).\,\mathbf{!}X_1.\,\mathbf{?}Y_1.\,\mathbf{!}_{(X_2,Y_2:\bigstar)}\,(X_2 \multimap Y_2).\,\mathbf{!}X_2.\,\mathbf{?}Y_2.\,\mathit{rec} && (\text{LÖB}) \end{aligned}$$

Semantic Asynchronous Session Subtyping - Example

Goal:

$$\mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X,Y:\bigstar)}\,(X \multimap Y).\,\mathbf{!}X.\,\mathbf{?}Y.\,\mathit{rec} <: \mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,\mathbf{!}X_1.\,\mathbf{!}(X_2 \multimap \mathbf{Z}).\,\mathbf{!}X_2.\,\mathbf{?B}.\,\mathbf{?Z}.\,\mathit{rec}$$

Derivation:

$$\begin{aligned}
&\mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X,Y:\bigstar)}\,(X \multimap Y).\,\mathbf{!}X.\,\mathbf{?}Y.\,\mathit{rec} \\
<: {}&\mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X_1,Y_1:\bigstar)}\,(X_1 \multimap Y_1).\,\mathbf{!}X_1.\,\mathbf{?}Y_1.\,\mathbf{!}_{(X_2,Y_2:\bigstar)}\,(X_2 \multimap Y_2).\,\mathbf{!}X_2.\,\mathbf{?}Y_2.\,\mathit{rec} && (\text{LÖB}) \\
<: {}&\mu\,(\mathit{rec} : \blacklozenge).\,\mathbf{!}_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,\mathbf{!}X_1.\,\mathbf{?B}.\,\mathbf{!}(X_2 \multimap \mathbf{Z}).\,\mathbf{!}X_2.\,\mathbf{?Z}.\,\mathit{rec} && (\text{S-ELIM}, \text{S-INTRO})
\end{aligned}$$

Rules:

S-ELIM

$$\frac{S_1 <: \mathbf{!}A.\,S_2}{S_1 <: \mathbf{!}_{(\vec{X}:\vec{k})}A.\,S_2}$$

S-INTRO

$$\mathbf{!}_{(\vec{X}:\vec{k})}\,A.\,S <: \mathbf{!}A[\vec{K}/\vec{X}].\,S[\vec{K}/\vec{X}]$$

Semantic Asynchronous Session Subtyping - Example

Goal:

$$\mu\,(\mathit{rec} : \blacklozenge).\,!_{(X,Y:\bigstar)}\,(X \multimap Y).\,!X.\,?Y.\,\mathit{rec} <: \mu\,(\mathit{rec} : \blacklozenge).\,!_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,!X_1.\,!(X_2 \multimap \mathbf{Z}).\,!X_2.\,?\mathbf{B}.\,?\mathbf{Z}.\,\mathit{rec}$$

Derivation:

$$\begin{aligned}
&\mu\,(\mathit{rec} : \blacklozenge).\,!_{(X,Y:\bigstar)}\,(X \multimap Y).\,!X.\,?Y.\,\mathit{rec} \\
<:\ &\mu\,(\mathit{rec} : \blacklozenge).\,!_{(X_1,Y_1:\bigstar)}\,(X_1 \multimap Y_1).\,!X_1.\,?Y_1.\,!_{(X_2,Y_2:\bigstar)}\,(X_2 \multimap Y_2).\,!X_2.\,?Y_2.\,\mathit{rec} && (\text{LÖB}) \\
<:\ &\mu\,(\mathit{rec} : \blacklozenge).\,!_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,!X_1.\,?\mathbf{B}.\,!(X_2 \multimap \mathbf{Z}).\,!X_2.\,?\mathbf{Z}.\,\mathit{rec} && (\text{S-ELIM}, \text{S-INTRO}) \\
<:\ &\mu\,(\mathit{rec} : \blacklozenge).\,!_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,!X_1.\,!(X_2 \multimap \mathbf{Z}).\,?\mathbf{B}.\,!X_2.\,?\mathbf{Z}.\,\mathit{rec} && (\text{SWAP})
\end{aligned}$$

Rules:

S-ELIM
$$\frac{S_1 <: !A.\,S_2}{S_1 <: !_{(\vec{X}:\vec{k})}A.\,S_2}$$

S-INTRO
$$!_{(\vec{X}:\vec{k})}\,A.\,S <: !A[\vec{K}/\vec{X}].\,S[\vec{K}/\vec{X}]$$

SWAP
$$?A_1.\,!A_2.\,S <: !A_2.\,?A_1.\,S$$

Semantic Asynchronous Session Subtyping - Example

Goal:

$$\mu\,(rec : \blacklozenge).\,\mathbf{!}_{(X,Y:\bigstar)}\,(X \multimap Y).\,\mathbf{!}X.\,\mathbf{?}Y.\,rec <: \mu\,(rec : \blacklozenge).\,\mathbf{!}_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,\mathbf{!}X_1.\,\mathbf{!}(X_2 \multimap \mathbf{Z}).\,\mathbf{!}X_2.\,\mathbf{?B}.\,\mathbf{?Z}.\,rec$$

Derivation:

$$\begin{aligned}
& \mu\,(rec : \blacklozenge).\,\mathbf{!}_{(X,Y:\bigstar)}\,(X \multimap Y).\,\mathbf{!}X.\,\mathbf{?}Y.\,rec \\
<:\ & \mu\,(rec : \blacklozenge).\,\mathbf{!}_{(X_1,Y_1:\bigstar)}\,(X_1 \multimap Y_1).\,\mathbf{!}X_1.\,\mathbf{?}Y_1.\,\mathbf{!}_{(X_2,Y_2:\bigstar)}\,(X_2 \multimap Y_2).\,\mathbf{!}X_2.\,\mathbf{?}Y_2.\,rec && (\text{LÖB}) \\
<:\ & \mu\,(rec : \blacklozenge).\,\mathbf{!}_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,\mathbf{!}X_1.\,\mathbf{?B}.\,\mathbf{!}(X_2 \multimap \mathbf{Z}).\,\mathbf{!}X_2.\,\mathbf{?Z}.\,rec && (\text{S-ELIM}, \text{S-INTRO}) \\
<:\ & \mu\,(rec : \blacklozenge).\,\mathbf{!}_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,\mathbf{!}X_1.\,\mathbf{!}(X_2 \multimap \mathbf{Z}).\,\mathbf{?B}.\,\mathbf{!}X_2.\,\mathbf{?Z}.\,rec && (\text{SWAP}) \\
<:\ & \mu\,(rec : \blacklozenge).\,\mathbf{!}_{(X_1,X_2:\bigstar)}\,(X_1 \multimap \mathbf{B}).\,\mathbf{!}X_1.\,\mathbf{!}(X_2 \multimap \mathbf{Z}).\,\mathbf{!}X_2.\,\mathbf{?B}.\,\mathbf{?Z}.\,rec && (\text{SWAP})
\end{aligned}$$

Rules:

S-ELIM
$$\frac{S_1 <: \mathbf{!}A.\,S_2}{S_1 <: \mathbf{!}_{(\vec{X}:\vec{k})}A.\,S_2}$$

S-INTRO
$$\mathbf{!}_{(\vec{X}:\vec{k})}\,A.\,S <: \mathbf{!}A[\vec{K}/\vec{X}].\,S[\vec{K}/\vec{X}]$$

SWAP
$$\mathbf{?}A_1.\,\mathbf{!}A_2.\,S <: \mathbf{!}A_2.\,\mathbf{?}A_1.\,S$$