Typing Asymmetric Client-Server Interaction

Sara Capecchi

Dipartimento di Informatica, Università di Torino

joint work with **Franco Barbanera** & **Ugo de'Liguoro**

FSEN 15 april 2009

Client-Server: an asymmetric relation

Client-Server: an asymmetric relation

- It is an intrinsically asymmetric relationship

Client-Server: an asymmetric relation

- It is an intrinsically asymmetric relationship
- differences in the rights and duties of the parties:

Client-Server: an asymmetric relation

- It is an intrinsically asymmetric relationship
- differences in the rights and duties of the parties:
  it is unreasonable to prohibit to the client to abort the connection at any time, while it would be unfair to admit such a behavior on the server side

Contracts

Contracts

Web services require:

Contracts

Web services require:

- unambiguous, machine-analyzable standardized descriptions of their capabilities

Contracts

Web services require:

- unambiguous, machine-analyzable standardized descriptions of their capabilities
- automatic tools to perform runtime service searches and validations

Contracts

Web services require:

- unambiguous, machine-analyzable standardized descriptions of their capabilities
- automatic tools to perform runtime service searches and validations

Contract

specification of mutual behavioural constraints among communication components

Contracts

Web services require:

- unambiguous, machine-analyzable standardized descriptions of their capabilities
- automatic tools to perform runtime service searches and validations

Contract

specification of mutual behavioural constraints among communication components

Properties

Contracts

Web services require:

- unambiguous, machine-analyzable standardized descriptions of their capabilities
- automatic tools to perform runtime service searches and validations

Contract

specification of mutual behavioural constraints among communication components

Properties

- conformance: a service conforms a contract if it meets his requirements;

Contracts

Web services require:

- unambiguous, machine-analyzable standardized descriptions of their capabilities
- automatic tools to perform runtime service searches and validations

Contract

specification of mutual behavioural constraints among communication components

Properties

- conformance: a service conforms a contract if it meets his requirements;
- compliance: all 'rightful' clients will be able to complete their intended protocols while interacting with a conformant service;

Contracts

Web services require:

- unambiguous, machine-analyzable standardized descriptions of their capabilities
- automatic tools to perform runtime service searches and validations

Contract

specification of mutual behavioural constraints among communication components

Properties

- conformance: a service conforms a contract if it meets his requirements;
- compliance: all 'rightful' clients will be able to complete their intended protocols while interacting with a conformant service;
- compatibility: two contracts are compatible when the services of one conform also with the other.

Formalising contracts with session types

Formalising contracts with session types

Session types

A type system for a dialect of the π-calculus adding primitives to handle sessions

Formalising contracts with session types

Session types

A type system for a dialect of the π-calculus adding primitives to handle sessions

Session

A session is an abstraction of a sequence of communications through a private channel between two parties over a session channel in such a way that both privacy and duality are guaranteed.

Formalising contracts with session types

Session types

A type system for a dialect of the π-calculus adding primitives to handle sessions

Session

A session is an abstraction of a sequence of communications through a private channel between two parties over a session channel in such a way that both privacy and duality are guaranteed.

Goals of the type system:

Formalising contracts with session types

Session types

A type system for a dialect of the π-calculus adding primitives to handle sessions

Session

A session is an abstraction of a sequence of communications through a private channel between two parties over a session channel in such a way that both privacy and duality are guaranteed.

Goals of the type system:

- to abstract a discipline of the interaction into a session type

Formalising contracts with session types

Session types

A type system for a dialect of the π-calculus adding primitives to handle sessions

Session

A session is an abstraction of a sequence of communications through a private channel between two parties over a session channel in such a way that both privacy and duality are guaranteed.

Goals of the type system:

- to abstract a discipline of the interaction into a session type
- to ensure safe handshaking-communications

Formalising contracts with session types

Session types

A type system for a dialect of the π-calculus adding primitives to handle sessions

Session

A session is an abstraction of a sequence of communications through a private channel between two parties over a session channel in such a way that both privacy and duality are guaranteed.

Goals of the type system:

- to abstract a discipline of the interaction into a session type
- to ensure safe handshaking-communications

Reference

Language Primitives and Type Disciplines for Structured Communication-based Programming - Honda Vasconcelos and Kubo - ESOP'98

Example

Example

In the original system sessions have to be symmetric:

Example

In the original system sessions have to be symmetric:

$$
\begin{aligned}
\mathit{CalcServer}_1 =_{\text{def}} \text{accept } a(x).x \triangleright \{ \; & \text{add} : x?(n).x?(m).x![n+m], \\
& \dots \\
& \text{div} : x?(n).x?(m).x![n \text{ div } m]\}
\end{aligned}
$$

$$
\mathit{CalcClient}_1 =_{\text{def}} \text{request } a(x).x \triangleleft \text{div}.x![21,5].x?(n)
$$

Example

In the original system sessions have to be symmetric:

$$\begin{aligned} \mathit{CalcServer}_1 =_{\mathrm{def}} \text{accept } a(x).x \triangleright \{ & \text{add} : x?(n).x?(m).x![n+m], \\ & \ldots \\ & \text{div} : x?(n).x?(m).x![n \text{ div } m]\} \end{aligned}$$

$$\mathit{CalcClient}_1 =_{\mathrm{def}} \text{request } a(x).x \triangleleft \text{div}.x![21, 5].x?(n)$$

Associated session types:

Example

In the original system sessions have to be symmetric:

$$\begin{array}{lll} \mathit{CalcServer}_1 =_{\mathrm{def}} \text{accept } a(x).x \triangleright \{ & \text{add} : x?(n).x?(m).x![n+m], \\ & \ldots \\ & \text{div} : x?(n).x?(m).x![n \text{ div } m]\} \end{array}$$

$$\mathit{CalcClient}_1 =_{\mathrm{def}} \text{request } a(x).x \triangleleft \text{div}.x![21,5].x?(n)$$

Associated session types:

$$S_{\mathrm{server1}} = \&\langle \text{add} : ?(\text{int})\,?(\text{int})\,![\text{int}]\text{end}, \ldots \text{div} : ?(\text{int})\,?(\text{int})\,![\text{int}]\text{end}\rangle,$$

Example

In the original system sessions have to be symmetric:

$$
\begin{aligned}
\mathit{CalcServer}_1 =_{\text{def}} \text{accept } a(x).x \triangleright \{ & \text{ add} : x?(n).x?(m).x![n+m], \\
& \dots \\
& \text{div} : x?(n).x?(m).x![n \text{ div } m]\}
\end{aligned}
$$

$$
\mathit{CalcClient}_1 =_{\text{def}} \text{request } a(x).x \triangleleft \text{div}.x![21,5].x?(n)
$$

Associated session types:

$$
S_{\text{server1}} = \&\langle \text{add} : ?(\text{int})\, ?(\text{int})\, ![\text{int}]\text{end}, \dots \text{div} : ?(\text{int})\, ?(\text{int})\, ![\text{int}]\text{end}\rangle,
$$

$$
S_{\text{client1}} = \overline{S_{\text{server1}}} = \oplus\langle \text{add} : ![\text{int}]\, ![\text{int}]\, ?(\text{int})\text{end}, \dots \text{div} : ![\text{int}]\, ![\text{int}]\, ?(\text{int})\text{end}\rangle
$$

Compliance

Compliance

However the service request of $CalcClient_1$ would be satisfied also by the server

Compliance

However the service request of *$CalcClient_1$* would be satisfied also by the server

$$\begin{aligned} CalcServer_2 =_{\mathrm{def}} \text{accept } a(x).x \triangleright \{ & \text{ add} : x?(n).x?(m).x![n+m] \\ & \ \dots \\ & \text{ div} : x?(n).x?(m).x![n \text{ div } m].x![n \text{ mod } m]\} \end{aligned}$$

Compliance

However the service request of $\mathit{CalcClient}_1$ would be satisfied also by the server

$$\begin{array}{rl} \mathit{CalcServer}_2 =_{\mathrm{def}} \mathsf{accept}\ a(x).x \triangleright \{ & \mathsf{add} : x?(n).x?(m).x![n+m] \\ & \ldots \\ & \mathsf{div} : x?(n).x?(m).x![n\ \mathsf{div}\ m].x![n\ \mathsf{mod}\ m] \} \end{array}$$

whose typing of x is just longer than S_{server1}:

Compliance

However the service request of $CalcClient_1$ would be satisfied also by the server

$$
\begin{aligned}
CalcServer_2 =_{\text{def}} \text{accept } a(x).x \triangleright \{\ & \text{add} : x?(n).x?(m).x![n+m] \\
& \dots \\
& \text{div} : x?(n).x?(m).x![n \text{ div } m].x![n \text{ mod } m]\}
\end{aligned}
$$

whose typing of x is just longer than S_{server1}:

$$S_{\text{server2}} = \&\langle \text{add} : ?(\text{int})\,?(\text{int})\,![\text{int}]\text{end}, \dots \text{div} : ?(\text{int})\,?(\text{int})\,![\text{int}]\,![\text{int}]\text{end}\rangle$$

$$S_{\text{server1}} = \&\langle \text{add} : ?(\text{int})\,?(\text{int})\,![\text{int}]\text{end}, \dots \text{div} : ?(\text{int})\,?(\text{int})\,![\text{int}]\text{end}\rangle,$$

Compliance

However the service request of *CalcClient*$_1$ would be satisfied also by the server

$$CalcServer_2 =_{\mathsf{def}} \mathsf{accept}\ a(x).x \triangleright \{\ \begin{array}{l} \mathsf{add} : x?(n).x?(m).x![n+m] \\ \dots \\ \mathsf{div} : x?(n).x?(m).x![n \ \mathsf{div}\ m].x![n \ \mathsf{mod}\ m]\} \end{array}$$

whose typing of x is just longer than S_{server1}:

$$S_{\mathsf{server2}} = \&\langle \mathsf{add} : ?(\mathsf{int})\, ?(\mathsf{int})\, ![\mathsf{int}]\mathsf{end}, \dots \mathsf{div} : ?(\mathsf{int})\, ?(\mathsf{int})\, ![\mathsf{int}]\, ![\mathsf{int}]\mathsf{end}\rangle$$

$$S_{\mathsf{server1}} = \&\langle \mathsf{add} : ?(\mathsf{int})\, ?(\mathsf{int})\, ![\mathsf{int}]\mathsf{end}, \dots \mathsf{div} : ?(\mathsf{int})\, ?(\mathsf{int})\, ![\mathsf{int}]\mathsf{end}\rangle,$$

- A client comply with a service if it successfully terminate any interaction with a service offering more

Compliance

However the service request of $CalcClient_1$ would be satisfied also by the server

$$
\begin{aligned}
CalcServer_2 =_{\text{def}} \text{accept } a(x).x \triangleright \{ \; & \text{add} : x?(n).x?(m).x![n+m] \\
& \ldots \\
& \text{div} : x?(n).x?(m).x![n \text{ div } m].x![n \text{ mod } m]\}
\end{aligned}
$$

whose typing of x is just longer than S_{server1}:

$$S_{\text{server2}} = \&\langle \text{add} : ?(\text{int})\,?(\text{int})\,![\text{int}]\text{end}, \ldots \text{div} : ?(\text{int})\,?(\text{int})\,![\text{int}]\,![\text{int}]\text{end}\rangle$$

$$S_{\text{server1}} = \&\langle \text{add} : ?(\text{int})\,?(\text{int})\,![\text{int}]\text{end}, \ldots \text{div} : ?(\text{int})\,?(\text{int})\,![\text{int}]\text{end}\rangle,$$

- A client comply with a service if it successfully terminate any interaction with a service offering more
- the compliance of the client with the server can be checked by formally proving that the dual of the client contract is the initial part of the server contract

Subtyping VS Prefix

Subtyping VS Prefix

Can the concept of being longer be caught by means of subtyping?

Subtyping VS Prefix

Can the concept of being longer be caught by means of subtyping?

We could extend the subtyping theory of session types S_{client1} is a subtype of S_{client2}

Subtyping VS Prefix

Can the concept of being longer be caught by means of subtyping?

We could extend the subtyping theory of session types S_{client1} is a subtype of S_{client2}

Reference

Subtyping for Session Types in the Pi-Calculus - Gay, Hole -Acta Informatica 2005

Subtyping VS Prefix

Can the concept of being longer be caught by means of subtyping?

We could extend the subtyping theory of session types S_{client1} is a subtype of S_{client2}

Reference

Subtyping for Session Types in the Pi-Calculus - Gay, Hole -Acta Informatica 2005

NO!

Subtyping VS Prefix

Can the concept of being longer be caught by means of subtyping?

We could extend the subtyping theory of session types S_{client1} is a subtype of S_{client2}

Reference

Subtyping for Session Types in the Pi-Calculus - Gay, Hole -Acta Informatica 2005

NO!

Theorem

There is no consistent theory of subtyping, extending the above theory, which includes the axiom $\mathrm{end} <: S$ *and satisfies the principle that if* $S <: S'$ *then* $\overline{S'} <: \overline{S}$.

Subtyping VS Prefix

Can the concept of being longer be caught by means of subtyping?

We could extend the subtyping theory of session types S_{client1} is a subtype of S_{client2}

Reference

Subtyping for Session Types in the Pi-Calculus - Gay, Hole -Acta Informatica 2005

NO!

Theorem

There is no consistent theory of subtyping, extending the above theory, which includes the axiom end $<: S$ *and satisfies the principle that if* $S <: S'$ *then* $\overline{S'} <: \overline{S}$.

The key property of subtyping (if $S <: S'$ then $\overline{S'} <: \overline{S}$) incompatible with the axiom end $<: S$

Prefix

Prefix

We introduce a new relation among session types:

$$S \preccurlyeq S'$$

prefix relation: if $S \preccurlyeq S'$ then any interaction pattern typed by S is the initial part of a pattern typed by S'

Prefix

We introduce a new relation among session types:

$$S \preccurlyeq S'$$

prefix relation: if $S \preccurlyeq S'$ then any interaction pattern typed by S is the initial part of a pattern typed by S'
Changing the interpretation of the typing $x : S$

Prefix

We introduce a new relation among session types:

$$S \preccurlyeq S'$$

prefix relation: if $S \preccurlyeq S'$ then any interaction pattern typed by S is the initial part of a pattern typed by S'

Changing the interpretation of the typing $x : S$

- In the server case S represents its duties, the commitment to an interaction which is at least of the shape (and the length) represented by S

Prefix

We introduce a new relation among session types:

$$S \preccurlyeq S'$$

prefix relation: if $S \preccurlyeq S'$ then any interaction pattern typed by S is the initial part of a pattern typed by S'
Changing the interpretation of the typing $x : S$

- In the server case S represents its duties, the commitment to an interaction which is at least of the shape (and the length) represented by S
- If x is a client end, then S represents the client's rights, telling that it is entitled to ask at most an interaction of that shape.

Prefix

Prefix

Syntax of session types

$$
\begin{array}{rrcl}
\text{Type} & T & ::= & \mathsf{bool} \mid \mathsf{nat} \mid \mathsf{int} \mid \mathsf{real} \mid \ S \ \mid \ \uparrow[S] \\
\text{Session type} & S & ::= & ?(T)S \mid ![T]S \mid \&\langle l_1 : S_1, \dots, l_n : S_n\rangle \\
 & & \mid & \oplus\langle l_1 : S_1, \dots, l_n : S_n\rangle \mid \mathsf{end}
\end{array}
$$

Prefix

Syntax of session types

$$
\begin{array}{rrl}
\text{Type} & T ::= & \mathsf{bool} \mid \mathsf{nat} \mid \mathsf{int} \mid \mathsf{real} \mid \ S \ \mid \ \uparrow[S] \\
\text{Session type} & S ::= & ?(T)S \mid ![T]S \mid \&\langle l_1 : S_1, \ldots, l_n : S_n\rangle \\
 & \mid & \oplus\langle l_1 : S_1, \ldots, l_n : S_n\rangle \mid \mathsf{end}
\end{array}
$$

Prefix Relation over First Order Session Types

The *prefix* relation over first-order session types, $S \preccurlyeq S'$ (read "S is a prefix of S'") is defined as the least preorder satisfying the following axiom and rules

$$
\frac{}{\mathsf{end} \preccurlyeq S} \qquad \frac{S \preccurlyeq S'}{?(T)S \preccurlyeq ?(T)S'} \qquad \frac{S \preccurlyeq S'}{![T]S \preccurlyeq ![T]S'}
$$

$$
\frac{S_i \preccurlyeq S'_i \quad (\forall i \leq n)}{\&\langle l_1 : S_1, .., l_n : S_n\rangle \preccurlyeq \&\langle l_1 : S'_1, .., l_n : S'_n\rangle} \qquad \frac{S_i \preccurlyeq S'_i \quad (\forall i \leq n)}{\oplus\langle l_1 : S_1, .., l_n : S_n\rangle \preccurlyeq \oplus\langle l_1 : S'_1, .., l_n : S'_n\rangle}
$$

Prefix: typing rules

Prefix: typing rules

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{+} : S \quad S' \preccurlyeq S}{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{+} : S'} \text{T-PREFS}$$

Prefix: typing rules

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{+} : S \quad S' \preccurlyeq S}{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{+} : S'}\ \textsc{T-PrefS}$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{-} : S \quad S \preccurlyeq S'}{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{-} : S'}\ \textsc{T-PrefC}$$

Prefix: typing rules

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{+} : S \quad S' \preccurlyeq S}{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{+} : S'} \text{ T-PrefS}$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{-} : S \quad S \preccurlyeq S'}{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{-} : S'} \text{ T-PrefC}$$

Prefix duality

For any $S, S' \in \mathcal{ST}$, if $S \preccurlyeq S'$ then $\overline{S} \preccurlyeq \overline{S'}$.

Prefix: typing rules

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{+} : S \quad S' \preccurlyeq S}{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{+} : S'}\ \textsc{T-PrefS}$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{-} : S \quad S \preccurlyeq S'}{\Gamma \vdash P \triangleright \Delta \cdot \kappa^{-} : S'}\ \textsc{T-PrefC}$$

Prefix duality

For any $S, S' \in \mathcal{ST}$, if $S \preccurlyeq S'$ then $\overline{S} \preccurlyeq \overline{S'}$.

Polarities: - for client side, $+$ for server side

Weak compliance

Weak compliance

- a client is "strongly compliant" with a service whenever it completes all direct interaction sessions with the service

Weak compliance

- a client is "strongly compliant" with a service whenever it completes all direct interaction sessions with the service
- session types do not enforce deadlock freeness in general: a client might be not strongly compliant because a deadlock occurs that prevents the session to proceed properly

Weak compliance

- a client is "strongly compliant" with a service whenever it completes all direct interaction sessions with the service
- session types do not enforce deadlock freeness in general: a client might be not strongly compliant because a deadlock occurs that prevents the session to proceed properly
- we can only expect a weaker concept of compliance to be warranted for typable systems, up to deadlock occurrences

Weak compliance

- a client is "strongly compliant" with a service whenever it completes all direct interaction sessions with the service
- session types do not enforce deadlock freeness in general: a client might be not strongly compliant because a deadlock occurs that prevents the session to proceed properly
- we can only expect a weaker concept of compliance to be warranted for typable systems, up to deadlock occurrences

Weak Compliance Property

A server cannot exhaust its actions on a channel before the corresponding client does.

Higher order sessions

Higher order sessions

- **Delegation** the ability for a process to pass a session to some third party which is in charge of continuing the interaction.

Higher order sessions

- **Delegation** the ability for a process to pass a session to some third party which is in charge of continuing the interaction.
- implemented by primitives throw and catch

Higher order sessions

- **Delegation** the ability for a process to pass a session to some third party which is in charge of continuing the interaction.
- implemented by primitives throw and catch

throw and catch naive typing

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \kappa_1^p : S \quad S' \neq \mathrm{end}}{\Gamma \vdash \mathrm{throw}\ \kappa_1^p[\kappa_2^q].P \triangleright \Delta \cdot \kappa_1^p : ![S']S \cdot \kappa_2^q : S'}$$

$$\frac{\Gamma \vdash \{\kappa_2^q/x\}P \triangleright \Delta \cdot \kappa_1^p : S \cdot \kappa_2^q : S'}{\Gamma \vdash \mathrm{catch}\ \kappa_1^p(x).P \triangleright \Delta \cdot \kappa_1^p : ?(S')S}$$

Higher order sessions

- **Delegation** the ability for a process to pass a session to some third party which is in charge of continuing the interaction.
- implemented by primitives throw and catch

throw and catch naive typing

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \kappa_1^p : S \qquad S' \neq \mathsf{end}}{\Gamma \vdash \mathsf{throw}\ \kappa_1^p[\kappa_2^q].P \triangleright \Delta \cdot \kappa_1^p : ![S']S \cdot \kappa_2^q : S'}$$

$$\frac{\Gamma \vdash \{\kappa_2^q/x\}P \triangleright \Delta \cdot \kappa_1^p : S \cdot \kappa_2^q : S'}{\Gamma \vdash \mathsf{catch}\ \kappa_1^p(x).P \triangleright \Delta \cdot \kappa_1^p : ?(S')S}$$

the above typing rules force $![S']S$ and $?(S')S$ to behave invariantly in S' w.r.t. prefix relation **TOO RESTRICTIVE!**

Higher order sessions: typing

Higher order sessions: typing

$$\text{throw } \kappa_1^{\overline{p}}[\kappa_2^q] \mid \text{catch } \kappa_1^p(x)P$$

Higher order sessions: typing

$$\text{throw } \kappa_1^{\overline{p}}[\kappa_2^q] \mid \text{catch } \kappa_1^p(x)P$$

We have to establish:

Higher order sessions: typing

$$\text{throw } \kappa_1^{\overline{p}}[\kappa_2^{q}] \mid \text{catch } \kappa_1^{p}(x)P$$

We have to establish:

- a relation between the p in the conclusion and the q in the premise of rule `TCatT`

Higher order sessions: typing

$$\text{throw } \kappa_1^{\overline{p}}[\kappa_2^q] \mid \text{catch } \kappa_1^p(x)P$$

We have to establish:

- a relation between the p in the conclusion and the q in the premise of rule `TCatT`
- variance: assume $![S']S$ be contravariant in S' and covariant in S, and that $?(S')S$ be covariant both in S' and in S

Higher order sessions: typing

$$\text{throw } \kappa_1^{\overline{p}}[\kappa_2^q] \mid \text{catch } \kappa_1^p(x)P$$

We have to establish:

- a relation between the p in the conclusion and the q in the premise of rule `TCatT`
- variance: assume $![S']S$ be contravariant in S' and covariant in S, and that $?(S')S$ be covariant both in S' and in S
- If $\mathbf{p} \neq \mathbf{q}$ problems arise because of an inner incoherence of the principle of delegation for those particular client/server asymmetric interactions:

Higher order sessions: typing

$$\text{throw } \kappa_1^{\overline{p}}[\kappa_2^{q}] \mid \text{catch } \kappa_1^{p}(x)P$$

We have to establish:

- a relation between the p in the conclusion and the q in the premise of rule `TCatT`
- variance: assume $![S']S$ be contravariant in S' and covariant in S, and that $?(S')S$ be covariant both in S' and in S
- If $\mathbf{p} \neq \mathbf{q}$ problems arise because of an inner incoherence of the principle of delegation for those particular client/server asymmetric interactions:
  - a client receiving a server can declare that the received server "does more" than it actually can do

Higher order sessions: typing

$$\text{throw } \kappa_1^{\overline{p}}[\kappa_2^{q}] \mid \text{catch } \kappa_1^{p}(x)P$$

We have to establish:

- a relation between the p in the conclusion and the q in the premise of rule `TCatT`
- variance: assume $![S']S$ be contravariant in S' and covariant in S, and that $?(S')S$ be covariant both in S' and in S
- If $\mathbf{p} \neq \mathbf{q}$ problems arise because of an inner incoherence of the principle of delegation for those particular client/server asymmetric interactions:
  - a client receiving a server can declare that the received server "does more" than it actually can do
  - a server receiving a client can declare that the received client "ask for less" than it actually can do

Higher order sessions: typing

$$\text{throw } \kappa_1^{\overline{p}}[\kappa_2^q] \mid \text{catch } \kappa_1^p(x)P$$

We have to establish:

- a relation between the p in the conclusion and the q in the premise of rule `TCatT`
- variance: assume $![S']S$ be contravariant in S' and covariant in S, and that $?(S')S$ be covariant both in S' and in S
- If $\mathbf{p} \neq \mathbf{q}$ problems arise because of an inner incoherence of the principle of delegation for those particular client/server asymmetric interactions:
  - a client receiving a server can declare that the received server "does more" than it actually can do
  - a server receiving a client can declare that the received client "ask for less" than it actually can do
- If $\mathbf{p} = \mathbf{q}$: problems depend only on the contravariance of the output type $![\,]$

Higher order sessions: typing

Higher order sessions: typing

Solution

Higher order sessions: typing

Solution

1. Covariance of both input and output higher-order session types:

Higher order sessions: typing

Solution

1. Covariance of both input and output higher-order session types:

$$\frac{S_1' \preccurlyeq S_2' \quad S_1 \preccurlyeq S_2}{![S_1']S_1 \preccurlyeq ![S_2']S_2} \qquad \frac{S_1' \preccurlyeq S_2' \quad S_1 \preccurlyeq S_2}{?(S_1')S_1 \preccurlyeq ?(S_2')S_2}$$

Higher order sessions: typing

Solution

1. Covariance of both input and output higher-order session types:

$$\frac{S_1' \preccurlyeq S_2' \quad S_1 \preccurlyeq S_2}{![S_1']S_1 \preccurlyeq ![S_2']S_2} \qquad \frac{S_1' \preccurlyeq S_2' \quad S_1 \preccurlyeq S_2}{?(S_1')S_1 \preccurlyeq ?(S_2')S_2}$$

2. put the equality of polarities of the subject and the object of a catch action:

Higher order sessions: typing

Solution

1. Covariance of both input and output higher-order session types:

$$\frac{S_1' \preccurlyeq S_2' \quad S_1 \preccurlyeq S_2}{![S_1']S_1 \preccurlyeq ![S_2']S_2} \qquad \frac{S_1' \preccurlyeq S_2' \quad S_1 \preccurlyeq S_2}{?(S_1')S_1 \preccurlyeq ?(S_2')S_2}$$

2. put the equality of polarities of the subject and the object of a catch action:

$$\frac{\Gamma \vdash \{\kappa_2^p/x\}P \triangleright \Delta \cdot \kappa_1^p : S \cdot \kappa_2^p : S'}{\Gamma \vdash \text{catch } \kappa_1^p(x).P \triangleright \Delta \cdot \kappa_1^p : ?(S')S}$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \kappa_1^{\overline{p}} : S}{\Gamma \vdash \text{throw } \kappa_1^{\overline{p}}[\kappa_2^p].P \triangleright \Delta \cdot \kappa_1^{\overline{p}} : ![S']S \cdot \kappa_2^p : S'}$$

Higher order sessions:properties

Higher order sessions:properties

Theorem (Error Freeness)

If $\Gamma \vdash P \triangleright \Delta$ *then* P *is error free.*

Higher order sessions:properties

Theorem (Error Freeness)

If $\Gamma \vdash P \triangleright \Delta$ *then* P *is error free.*

where

Definition (Error Freeness)

A process P is an *error* if there exists a channel κ such that either two κ-processes which do not form a κ-redex occur in P in head position, or there are more than two k-processes in head position. A process P is *error free* if there exists no Q such that $P \stackrel{*}{\longrightarrow} Q$ which is an error.

Higher order sessions:properties

Higher order sessions:properties

Theorem (Higher-order Weak Compliance)

Let P be a running process which is a derivative of some typed initial process. If P contains a κ^- process in head position, then it includes either a dual κ^+-process (though not necessarily in head position) or a potential κ^+-process generator.

Higher order sessions:properties

Theorem (Higher-order Weak Compliance)

Let P be a running process which is a derivative of some typed initial process. If P contains a κ^- process in head position, then it includes either a dual κ^+-process (though not necessarily in head position) or a potential κ^+-process generator.

Higher order sessions:properties

Theorem (Higher-order Weak Compliance)

Let P be a running process which is a derivative of some typed initial process. If P contains a κ^- process in head position, then it includes either a dual κ^+-process (though not necessarily in head position) or a potential κ^+-process generator.

where:

Higher order sessions:properties

Theorem (Higher-order Weak Compliance)

Let P be a running process which is a derivative of some typed initial process. If P contains a κ^- process in head position, then it includes either a dual κ^+-process (though not necessarily in head position) or a potential κ^+-process generator.

where:

Definition (Potential κ^+-process generator)

A *potential κ^+-process generator* is any process of the form throw $k[\kappa^+].Q$

Higher order sessions:properties

Theorem (Higher-order Weak Compliance)

Let P be a running process which is a derivative of some typed initial process. If P contains a κ^- *process in head position, then it includes either a dual* κ^+*-process (though not necessarily in head position) or a* potential κ^+-process generator.

where:

Definition (Potential κ^+-process generator)

A *potential* κ^+*-process generator* is any process of the form throw $k[\kappa^+].Q$

A process P is *initial* if does not contain any channel name κ neither free nor bound.

Higher order sessions:properties

Theorem (Higher-order Weak Compliance)

Let P be a running process which is a derivative of some typed initial process. If P contains a κ^- process in head position, then it includes either a dual κ^+-process (though not necessarily in head position) or a potential κ^+-process generator.

where:

Definition (Potential κ^+-process generator)

A *potential κ^+-process generator* is any process of the form throw $k[\kappa^+].Q$

A process P is *initial* if does not contain any channel name κ neither free nor bound.

A process P is *running* if there exists an initial Q such that: $Q \xrightarrow{*} P$

Conclusions

Conclusions

- introducing the relation of prefix in the rules of the systems breaks the symmetry of session type systems studied so far

Conclusions

- introducing the relation of prefix in the rules of the systems breaks the symmetry of session type systems studied so far
- does not destroy the basic properties of the system, namely subject reduction and error freeness

Conclusions

- introducing the relation of prefix in the rules of the systems breaks the symmetry of session type systems studied so far
- does not destroy the basic properties of the system, namely subject reduction and error freeness
- obstacle: ordinary session types do not guarantee deadlock-freeness

Conclusions

- introducing the relation of prefix in the rules of the systems breaks the symmetry of session type systems studied so far
- does not destroy the basic properties of the system, namely subject reduction and error freeness
- obstacle: ordinary session types do not guarantee deadlock-freeness
- even more acute when admitting asymmetric sessions and typing, because the client can be prevented from performing all its actions because of the presence of a residual of some sessions abandoned by other clients

Conclusions

- introducing the relation of prefix in the rules of the systems breaks the symmetry of session type systems studied so far
- does not destroy the basic properties of the system, namely subject reduction and error freeness
- obstacle: ordinary session types do not guarantee deadlock-freeness
- even more acute when admitting asymmetric sessions and typing, because the client can be prevented from performing all its actions because of the presence of a residual of some sessions abandoned by other clients
- the processes that can be represented in the π-calculus with sessions are far richer than those considered in the theory of contracts, and hence closer to real world as much as they can be more difficult to master.