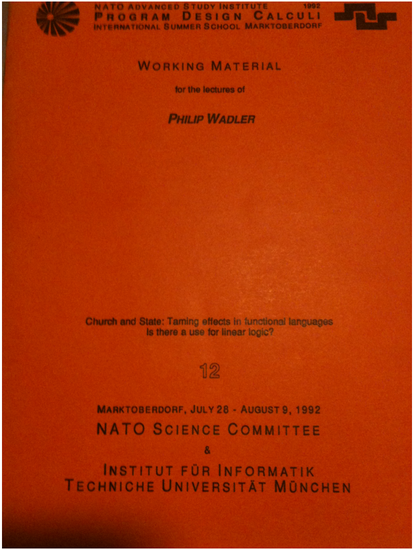
Subtyping Supports Safe Session Substitution

Simon Gay
School of Computing Science, University of Glasgow

EPSRC EP/K034413 & EP/L00058X
COST Action IC1201

Meeting Phil



NATO ADVANCED STUDY INSTITUTE 1992
PROGRAM DESIGN CALCULI
INTERNATIONAL SUMMER SCHOOL MARKTOBERDORF

WORKING MATERIAL

for the lectures of

PHILIP WADLER

Church and State: Taming effects in functional languages
Is there a use for linear logic?

12

MARKTOBERDORF, JULY 28 - AUGUST 9, 1992

NATO SCIENCE COMMITTEE

&

INSTITUT FÜR INFORMATIK
TECHNICHE UNIVERSITÄT MÜNCHEN

Session Types

- Describe a communication protocol as a type, and use type checking to guarantee correctness of communication.

Session Types

- Describe a communication protocol as a type, and use type checking to guarantee correctness of communication.
- The original papers:

  Honda, "Types for Dyadic Interaction", CONCUR 1993.

  Takeuchi, Honda & Kubo, "An Interaction-Based Language and its Typing System", PARLE 1994.

  Honda, Vasconcelos & Kubo, "Language Primitives and Type Discipline for Structured Communication-Based Programming", ESOP 1998.

Session Types

- During the last 20 years, session types have developed into a significant theme in programming languages.

Session Types

- During the last 20 years, session types have developed into a significant theme in programming languages.
- Computing has moved from the era of data processing to the era of communication.

Session Types

- During the last 20 years, session types have developed into a significant theme in programming languages.
- Computing has moved from the era of data processing to the era of communication.
- Data types codify the structure of data and make it available to programming tools.

Session Types

- During the last 20 years, session types have developed into a significant theme in programming languages.
- Computing has moved from the era of data processing to the era of communication.
- Data types codify the structure of data and make it available to programming tools.
- Session types codify the structure of communication and make it available to programming tools.

Session Types

- During the last 20 years, session types have developed into a significant theme in programming languages.
- Computing has moved from the era of data processing to the era of communication.
- Data types codify the structure of data and make it available to programming tools.
- Session types codify the structure of communication and make it available to programming tools.
- EPSRC Programme Grant "From Data Types to Session Types: A Basis for Concurrency and Distribution" (SG, Phil Wadler and Nobuko Yoshida).

The Maths Server: Types / Protocols

- The session type of the server's channel endpoint:

$$S \quad = \quad \&\langle\ \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \qquad\qquad \mathsf{eq} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{bool}].\mathsf{end}\ \rangle$$

The Maths Server: Types / Protocols

- The session type of the server's channel endpoint:

$$S = \&\langle \text{ add} : ?[\text{int}].?[\text{int}].![\text{int}].\text{end}, \\ \text{eq} : ?[\text{int}].?[\text{int}].![\text{bool}].\text{end} \rangle$$

- The session type of the client's channel endpoint:

$$C = \oplus\langle \text{ add} : ![\text{int}].![\text{int}].?[\text{int}].\text{end}, \\ \text{eq} : ![\text{int}].![\text{int}].?[\text{bool}].\text{end} \rangle$$

The Maths Server: Types / Protocols

- The session type of the server's channel endpoint:

$$S \quad = \quad \&\langle\ \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \quad \mathsf{eq} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{bool}].\mathsf{end}\ \rangle$$

- The session type of the client's channel endpoint:

$$C \quad = \quad \oplus\langle\ \mathsf{add} : ![\mathsf{int}].![\mathsf{int}].?[\mathsf{int}].\mathsf{end}, \quad \mathsf{eq} : ![\mathsf{int}].![\mathsf{int}].?[\mathsf{bool}].\mathsf{end}\ \rangle$$

- Duality: $S = \overline{C}$

Upgrading the Maths Server

- newserver adds a new service and extends an existing service:

$$
\begin{aligned}
S' \quad = \quad \&\langle\ & \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\
& \mathsf{mul} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\
& \mathsf{eq} : ?[\mathsf{float}].?[\mathsf{float}].![\mathsf{bool}].\mathsf{end}\ \rangle
\end{aligned}
$$

Upgrading the Maths Server

- newserver adds a new service and extends an existing service:

$$S' \quad = \quad \&\langle \begin{array}{l} \mathsf{add}:?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{mul}:?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{eq}:?[\mathsf{float}].?[\mathsf{float}].![\mathsf{bool}].\mathsf{end} \; \rangle \end{array}$$

- Interaction with a client of type $C = \overline{S}$ ($\neq \overline{S'}$) is semantically safe, assuming that int is a subtype of float:

$$C \quad = \quad \oplus\langle \begin{array}{l} \mathsf{add}:![\mathsf{int}].![\mathsf{int}].?[\mathsf{int}].\mathsf{end}, \\ \mathsf{eq}:![\mathsf{int}].![\mathsf{int}].?[\mathsf{bool}].\mathsf{end} \; \rangle \end{array}$$

Upgrading the Maths Server

- newserver adds a new service and extends an existing service:

$$S' \quad = \quad \&\langle \begin{array}{l} \mathsf{add} :?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{mul} :?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{eq} :?[\mathsf{float}].?[\mathsf{float}].![\mathsf{bool}].\mathsf{end} \ \rangle \end{array}$$

- Interaction with a client of type $C = \overline{S}$ ($\neq \overline{S'}$) is semantically safe, assuming that int is a subtype of float:

$$C \quad = \quad \oplus\langle \begin{array}{l} \mathsf{add} :![\mathsf{int}].![\mathsf{int}].?[\mathsf{int}].\mathsf{end}, \\ \mathsf{eq} :![\mathsf{int}].![\mathsf{int}].?[\mathsf{bool}].\mathsf{end} \ \rangle \end{array}$$

- A theory of subtyping needs to allow this interaction to be typechecked.

Two Definitions of Subtyping

- Gay and Hole (1999, 2005) define

$$\&\left\langle \begin{array}{l} \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{eq} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{bool}].\mathsf{end} \end{array} \right\rangle \leqslant \&\left\langle \begin{array}{l} \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{mul} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{eq} : ?[\mathsf{float}].?[\mathsf{float}].![\mathsf{bool}].\mathsf{end} \end{array} \right\rangle$$

Two Definitions of Subtyping

- Gay and Hole (1999, 2005) define

$$\&\langle \begin{array}{l} \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{eq} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{bool}].\mathsf{end} \end{array} \rangle \quad \leqslant \quad \&\langle \begin{array}{l} \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{mul} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{eq} : ?[\mathsf{float}].?[\mathsf{float}].![\mathsf{bool}].\mathsf{end} \end{array} \rangle$$

- Honda et al. (2007 onwards) define

$$\&\langle \begin{array}{l} \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{eq} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{bool}].\mathsf{end} \end{array} \rangle \quad \sqsupseteq \quad \&\langle \begin{array}{l} \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{mul} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ \mathsf{eq} : ?[\mathsf{float}].?[\mathsf{float}].![\mathsf{bool}].\mathsf{end} \end{array} \rangle$$

Two Definitions of Subtyping

- Gay and Hole (1999, 2005) define

$$\&\langle\ \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end},\ \mathsf{eq} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{bool}].\mathsf{end}\ \rangle \leqslant \&\langle\ \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end},\ \mathsf{mul} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end},\ \mathsf{eq} : ?[\mathsf{float}].?[\mathsf{float}].![\mathsf{bool}].\mathsf{end}\ \rangle$$

- Honda et al. (2007 onwards) define

$$\&\langle\ \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end},\ \mathsf{eq} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{bool}].\mathsf{end}\ \rangle \sqsupseteq \&\langle\ \mathsf{add} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end},\ \mathsf{mul} : ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end},\ \mathsf{eq} : ?[\mathsf{float}].?[\mathsf{float}].![\mathsf{bool}].\mathsf{end}\ \rangle$$

- How can both definitions be correct?

Justifying Subtyping: Safe Substitutability

- Liskov and Wing (1994): T is a subtype of U if an expression of type T can be used wherever an expression of type U is expected, without violating the runtime safety property guaranteed by the type system.

Justifying Subtyping: Safe Substitutability

- Liskov and Wing (1994): T is a subtype of U if an expression of type T can be used wherever an expression of type U is expected, without violating the runtime safety property guaranteed by the type system.
- For session types, runtime safety means that all messages are understood.

Justifying Subtyping: Safe Substitutability

- Liskov and Wing (1994): T is a subtype of U if an expression of type T can be used wherever an expression of type U is expected, without violating the runtime safety property guaranteed by the type system.
- For session types, runtime safety means that all messages are understood.
- We have to understand which expressions we are interested in.

Justifying Subtyping: Safe Substitutability

- Liskov and Wing (1994): T is a subtype of U if an expression of type T can be used wherever an expression of type U is expected, without violating the runtime safety property guaranteed by the type system.
- For session types, runtime safety means that all messages are understood.
- We have to understand which expressions we are interested in.
- Gay and Hole: safe substitutability of channels.

Justifying Subtyping: Safe Substitutability

- Liskov and Wing (1994): T is a subtype of U if an expression of type T can be used wherever an expression of type U is expected, without violating the runtime safety property guaranteed by the type system.
- For session types, runtime safety means that all messages are understood.
- We have to understand which expressions we are interested in.
- Gay and Hole: safe substitutability of channels.
- Honda et al.: safe substitutability of processes.

Justifying Subtyping: Safe Substitutability

- Liskov and Wing (1994): T is a subtype of U if an expression of type T can be used wherever an expression of type U is expected, without violating the runtime safety property guaranteed by the type system.
- For session types, runtime safety means that all messages are understood.
- We have to understand which expressions we are interested in.
- Gay and Hole: safe substitutability of channels.
- Honda et al.: safe substitutability of processes.
- This has become folklore in the session types community.

Channel-Oriented Subtyping (Gay and Hole)

- Substitution of a channel (endpoint) can be achieved by passing it as a function parameter or by sending it as a message on another channel.

Channel-Oriented Subtyping (Gay and Hole)

- Substitution of a channel (endpoint) can be achieved by passing it as a function parameter or by sending it as a message on another channel.
- newserver has been implemented on the assumption that it will use a channel of type $S' = \&\langle \mathsf{add} : \ldots, \mathsf{mul} : \ldots, \mathsf{eq} : \ldots \rangle$.

Channel-Oriented Subtyping (Gay and Hole)

- Substitution of a channel (endpoint) can be achieved by passing it as a function parameter or by sending it as a message on another channel.
- newserver has been implemented on the assumption that it will use a channel of type $S' = \&\langle \mathsf{add} : \ldots, \mathsf{mul} : \ldots, \mathsf{eq} : \ldots \rangle$.
- newserver implements the add, mul and eq services.

Channel-Oriented Subtyping (Gay and Hole)

- Substitution of a channel (endpoint) can be achieved by passing it as a function parameter or by sending it as a message on another channel.
- newserver has been implemented on the assumption that it will use a channel of type $S' = \&\langle \mathsf{add} : \ldots, \mathsf{mul} : \ldots, \mathsf{eq} : \ldots \rangle$.
- newserver implements the add, mul and eq services.
- If newserver is given a channel of type $S = \&\langle \mathsf{add} : \ldots, \mathsf{eq} : \ldots \rangle$ then execution is safe: the mul service is never used, because a client of type $\overline{S}$ can't send mul.

Channel-Oriented Subtyping (Gay and Hole)

- Substitution of a channel (endpoint) can be achieved by passing it as a function parameter or by sending it as a message on another channel.
- newserver has been implemented on the assumption that it will use a channel of type $S' = \&\langle \mathsf{add} : \ldots, \mathsf{mul} : \ldots, \mathsf{eq} : \ldots \rangle$.
- newserver implements the add, mul and eq services.
- If newserver is given a channel of type $S = \&\langle \mathsf{add} : \ldots, \mathsf{eq} : \ldots \rangle$ then execution is safe: the mul service is never used, because a client of type $\overline{S}$ can't send mul.
- $S \leqslant S'$ (covariant in the set of labels)

Channel-Oriented Subtyping (Gay and Hole)

- Substitution of a channel (endpoint) can be achieved by passing it as a function parameter or by sending it as a message on another channel.
- newserver has been implemented on the assumption that it will use a channel of type $S' = \&\langle \mathsf{add} : \ldots, \mathsf{mul} : \ldots, \mathsf{eq} : \ldots \rangle$.
- newserver implements the add, mul and eq services.
- If newserver is given a channel of type $S = \&\langle \mathsf{add} : \ldots, \mathsf{eq} : \ldots \rangle$ then execution is safe: the mul service is never used, because a client of type $\overline{S}$ can't send mul.
- $S \leqslant S'$ (covariant in the set of labels)
- In Gay and Hole's pi-calculus session type system, this is how an old client can safely connect to a new server.

Other Derivations of Channel-Oriented Subtyping

- Castagna et al. (2009): semantic subtyping for session types.

Other Derivations of Channel-Oriented Subtyping

- Castagna et al. (2009): semantic subtyping for session types.
- Dardha et al. (2012): translate session types into linear pi types + variants, and derive subtyping.

Other Derivations of Channel-Oriented Subtyping

- Castagna et al. (2009): semantic subtyping for session types.
- Dardha et al. (2012): translate session types into linear pi types + variants, and derive subtyping.
- Gay (2016): derive the definition of subtyping from the structure of the type safety proof.

Process-Oriented Subtyping (Honda et al.)

- View the session environment as the type of a process:
  $\mathsf{server}(x^+) \vdash x^+ : S \qquad S = \&\langle \mathsf{add} : \ldots, \mathsf{eq} : \ldots \rangle$

Process-Oriented Subtyping (Honda et al.)

- View the session environment as the type of a process:
  $\mathsf{server}(x^+) \vdash x^+ : S \qquad S = \&\langle \mathsf{add} : \ldots, \mathsf{eq} : \ldots \rangle$
- $\mathsf{server}(x^+)$ can execute in an environment in which x^- allows choices within the set of labels of S, i.e. add and eq.

Process-Oriented Subtyping (Honda et al.)

- View the session environment as the type of a process:
  $\mathsf{server}(x^+) \vdash x^+ : S \qquad S = \&\langle \mathsf{add} : \ldots, \mathsf{eq} : \ldots \rangle$
- $\mathsf{server}(x^+)$ can execute in an environment in which x^- allows choices within the set of labels of S, i.e. add and eq.
- We have $\qquad \mathsf{newserver}(x^+) \vdash x^+ : S'$
  $S' = \&\langle \mathsf{add} : \ldots, \mathsf{mul} : \ldots, \mathsf{eq} : \ldots \rangle$

Process-Oriented Subtyping (Honda et al.)

- View the session environment as the type of a process:
  $\mathsf{server}(x^+) \vdash x^+ : S \qquad S = \&\langle \mathsf{add} : \ldots, \mathsf{eq} : \ldots \rangle$
- $\mathsf{server}(x^+)$ can execute in an environment in which x^- allows choices within the set of labels of S, i.e. add and eq.
- We have $\quad \mathsf{newserver}(x^+) \vdash x^+ : S'$
  $S' = \&\langle \mathsf{add} : \ldots, \mathsf{mul} : \ldots, \mathsf{eq} : \ldots \rangle$
- But also newserver can execute in an environment in which server can execute.

Process-Oriented Subtyping (Honda et al.)

- View the session environment as the type of a process:
  $\mathsf{server}(x^+) \vdash x^+ : S \qquad S = \&\langle \mathsf{add} : \ldots, \mathsf{eq} : \ldots \rangle$
- $\mathsf{server}(x^+)$ can execute in an environment in which x^- allows choices within the set of labels of S, i.e. add and eq.
- We have $\qquad \mathsf{newserver}(x^+) \vdash x^+ : S'$
  $S' = \&\langle \mathsf{add} : \ldots, \mathsf{mul} : \ldots, \mathsf{eq} : \ldots \rangle$
- But also newserver can execute in an environment in which server can execute.
- So safe substitutability of processes means that $S' \sqsubseteq S$ (contravariant in the set of labels).

Process-Oriented Subtyping (Honda et al.)

- View the session environment as the type of a process:
  $\mathsf{server}(x^+) \vdash x^+ : S \qquad S = \&\langle \mathsf{add} : \ldots, \mathsf{eq} : \ldots \rangle$
- $\mathsf{server}(x^+)$ can execute in an environment in which x^- allows choices within the set of labels of S, i.e. add and eq.
- We have $\quad \mathsf{newserver}(x^+) \vdash x^+ : S'$
  $S' = \&\langle \mathsf{add} : \ldots, \mathsf{mul} : \ldots, \mathsf{eq} : \ldots \rangle$
- But also newserver can execute in an environment in which server can execute.
- So safe substitutability of processes means that $S' \sqsubseteq S$ (contravariant in the set of labels).
- This approach is natural if processes can be sent on channels (higher-order pi) or when combining pi and lambda.

Unifying Channel-Oriented and Process-Oriented Subtyping

- Typing judgements à la Gay and Hole: $\Gamma \vdash P : \mathsf{proc}$

Unifying Channel-Oriented and Process-Oriented Subtyping

- Typing judgements à la Gay and Hole: $\Gamma \vdash P : \mathsf{proc}$
- $x : S \vdash \mathsf{server}(s) : \mathsf{proc}$

Unifying Channel-Oriented and Process-Oriented Subtyping

- Typing judgements à la Gay and Hole: $\Gamma \vdash P : \mathsf{proc}$
- $x : S \vdash \mathsf{server}(s) : \mathsf{proc}$
- Mostrous and Yoshida (2007): $\lambda x.\mathsf{server}(x) : S \to \mathsf{proc}$

Unifying Channel-Oriented and Process-Oriented Subtyping

- Typing judgements à la Gay and Hole: $\Gamma \vdash P : \mathsf{proc}$
- $x : S \vdash \mathsf{server}(s) : \mathsf{proc}$
- Mostrous and Yoshida (2007): $\lambda x.\mathsf{server}(x) : S \to \mathsf{proc}$
- $\lambda x.\mathsf{newserver}(x) : S' \to \mathsf{proc}$

Unifying Channel-Oriented and Process-Oriented Subtyping

- Typing judgements à la Gay and Hole: $\Gamma \vdash P : \mathsf{proc}$
- $x : S \vdash \mathsf{server}(s) : \mathsf{proc}$
- Mostrous and Yoshida (2007): $\lambda x.\mathsf{server}(x) : S \to \mathsf{proc}$
- $\lambda x.\mathsf{newserver}(x) : S' \to \mathsf{proc}$
- Typing judgements à la Honda et al.: $P \vdash \mathsf{proc}(\Gamma)$

Unifying Channel-Oriented and Process-Oriented Subtyping

- Typing judgements à la Gay and Hole: $\Gamma \vdash P : \mathsf{proc}$
- $x : S \vdash \mathsf{server}(s) : \mathsf{proc}$
- Mostrous and Yoshida (2007): $\lambda x.\mathsf{server}(x) : S \to \mathsf{proc}$
- $\lambda x.\mathsf{newserver}(x) : S' \to \mathsf{proc}$
- Typing judgements à la Honda et al.: $P \vdash \mathsf{proc}(\Gamma)$
- Identify $\mathsf{proc}(x : S)$ with $S \to \mathsf{proc}$

Unifying Channel-Oriented and Process-Oriented Subtyping

- Typing judgements à la Gay and Hole: $\Gamma \vdash P : \mathsf{proc}$
- $x : S \vdash \mathsf{server}(s) : \mathsf{proc}$
- Mostrous and Yoshida (2007): $\lambda x.\mathsf{server}(x) : S \rightarrow \mathsf{proc}$
- $\lambda x.\mathsf{newserver}(x) : S' \rightarrow \mathsf{proc}$
- Typing judgements à la Honda et al.: $P \vdash \mathsf{proc}(\Gamma)$
- Identify $\mathsf{proc}(x : S)$ with $S \rightarrow \mathsf{proc}$
- An abstracted process is a self-contained entity that can be sent and then substituted into a context.

Unifying Channel-Oriented and Process-Oriented Subtyping

- Typing judgements à la Gay and Hole: $\Gamma \vdash P : \mathsf{proc}$
- $x : S \vdash \mathsf{server}(s) : \mathsf{proc}$
- Mostrous and Yoshida (2007): $\lambda x.\mathsf{server}(x) : S \to \mathsf{proc}$
- $\lambda x.\mathsf{newserver}(x) : S' \to \mathsf{proc}$
- Typing judgements à la Honda et al.: $P \vdash \mathsf{proc}(\Gamma)$
- Identify $\mathsf{proc}(x : S)$ with $S \to \mathsf{proc}$
- An abstracted process is a self-contained entity that can be sent and then substituted into a context.
- Taking $S \leqslant S'$ (channel-oriented) and using subtyping for function types gives $\mathsf{proc}(x : S') \leqslant \mathsf{proc}(x : S)$, corresponding to the process-oriented definition $S' \sqsubseteq S$.

Unifying Channel-Oriented and Process-Oriented Subtyping

- Typing judgements à la Gay and Hole: $\Gamma \vdash P : \mathsf{proc}$
- $x : S \vdash \mathsf{server}(s) : \mathsf{proc}$
- Mostrous and Yoshida (2007): $\lambda x.\mathsf{server}(x) : S \to \mathsf{proc}$
- $\lambda x.\mathsf{newserver}(x) : S' \to \mathsf{proc}$
- Typing judgements à la Honda et al.: $P \vdash \mathsf{proc}(\Gamma)$
- Identify $\mathsf{proc}(x : S)$ with $S \to \mathsf{proc}$
- An abstracted process is a self-contained entity that can be sent and then substituted into a context.
- Taking $S \leqslant S'$ (channel-oriented) and using subtyping for function types gives $\mathsf{proc}(x : S') \leqslant \mathsf{proc}(x : S)$, corresponding to the process-oriented definition $S' \sqsubseteq S$.
- The difference between channel-oriented and process-oriented subtyping is explained by contravariance of the function type constructor.

end