Session-based Type Discipline for Pi Calculus with Matching

Marco Giunti

joint work with

K.Honda,V.Vasconcelos and N.Yoshida

PLACES, ETAPS, March 22 2009

Background

Pi calculi with session-based type discipline [HVK-98]

- Semantics feature *fresh session passing*

$$k![k'].P \mid k?(x).Q \longrightarrow P \mid Q[k'/x] \quad k' \notin \mathrm{fc}(Q)$$

- Type system requires k' not used in P ($k' \notin \mathrm{fc}(P)$)

Background

Pi calculi with session-based type discipline [HVK-98]

- Semantics feature *fresh session passing*

$$k![k'].P \mid k?(x).Q \longrightarrow P \mid Q[k'/x] \quad k' \notin \mathrm{fc}(Q)$$

- Type system requires k' not used in P ($k' \notin \mathrm{fc}(P)$)

Semantics

- Removing the side condition breaks subject reduction
- Many works used polarized channels to recover type safety [Gay&Hole-05]

Background

Pi calculi with session-based type discipline [HVK-98]

- Semantics feature *fresh session passing*

$$k![k'].P \mid k?(x).Q \longrightarrow P \mid Q[k'/x] \quad k' \notin \mathrm{fc}(Q)$$

- Type system requires k' not used in P ($k' \notin \mathrm{fc}(P)$)

Semantics

- Removing the side condition breaks subject reduction
- Many works used polarized channels to recover type safety [Gay&Hole-05]

Type system

- k' used for communication in P unsound
- still could be other uses of k'

Example: Subject Reduction Lost

Passed session in fc of receiver

$$P = k![k'] \mid k?(x).x?().k'![] \qquad \emptyset \vdash P \triangleright k : \perp, k' : \perp$$

$$P \to k'?().k'![] \qquad \emptyset \not\vdash k'?().k'![] \triangleright k : \perp, k' : \perp$$

Example: Subject Reduction Lost

Passed session in fc of receiver

$$P = k![k'] \mid k?(x).x?().k'![] \qquad \emptyset \vdash P \triangleright k : \perp, k' : \perp$$

$$P \rightarrow k'?().k'![] \qquad \emptyset \not\vdash k'?().k'![] \triangleright k : \perp, k' : \perp$$

Passed session in fc of continuation

$$P = k![k'].k'?() \mid k?(x).k_1![x] \rightarrow k'() \mid k_1[k'^T]$$

This talk

1. Polarity-free [HVK-98] with general session passing is type-safe

$$k![k'].P \mid k?(x).Q \longrightarrow P \mid Q[k'/x]$$

2. Generalization of session types with limited use of passed sessions

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot u : U \cdot k' : T'_T}{\Gamma \vdash k![k'].Q \triangleright \Delta \cdot k : ![T].U \cdot k' : T}$$

Safe use of passed sessions

Non-communication operations, e.g.

- check the identity of sessions,
- store sessions in data structures

Safe use of passed sessions

Non-communication operations, e.g.

- check the identity of sessions,
- store sessions in data structures

Example: database of malicious sessions

$\mathsf{DB}(d) = \mathsf{accept}\ d(x).x \triangleright$

$\{\mathsf{contains} : x?(y^{\mathsf{end}}).\mathsf{if}\ y \in \mathsf{set}\ \mathsf{then}\ x \triangleleft \mathsf{no}.\mathsf{DB}(d) \mathsf{else}\ x \triangleleft \mathsf{yes}.\mathsf{DB}(d),$

$\mathsf{put} : x?(y^{\mathsf{end}}).\mathsf{Store}[y].\mathsf{DB}(d)\}$

$\mathsf{CLIENT}(d) =$

$x?(y).\mathsf{request}\ d(z).z \triangleleft \mathsf{contains}.z![y^{\mathsf{end}}].z \triangleright \{\mathsf{yes} : P(y^T), \mathsf{no} : Q(y^T)\}$

$\mid x![y^T].\mathsf{request}\ d(z).z \triangleleft \mathsf{put}.z![y^{\mathsf{end}}]$

Pi calculus with sessions

Top-level syntax

$$P ::= u![e].P \mid u?(x).P \mid P \mid P' \mid (\nu a)P \mid \mathbf{0}$$

$\mid$ def $\{X_i(\tilde{x}_i) = P_i\}_{i \in I}$ in $P \mid X[\tilde{e}]$ — recursion

$\mid$ request $u(x).P \mid$ accept $u(x).P$ — session install

$\mid u \triangleleft l.P \mid u \triangleright \{l_i : P_i\}_{i \in I}$ — label select & branch

$\mid$ if $[u = v]$ then P else P' — matching

$u, v ::= a \mid x \mid$ true $\mid$ false — values

Pi calculus with sessions

Top-level syntax

$$
\begin{array}{rll}
P & ::= u![e].P \mid u?(x).P \mid P \mid P' \mid (\nu a)P \mid \mathbf{0} & \\
& \mid \text{def } \{X_i(\tilde{x}_i) = P_i\}_{i\in I} \text{ in } P \mid X[\tilde{e}] & \text{recursion} \\
& \mid \text{request } u(x).P \mid \text{accept } u(x).P & \text{session install} \\
& \mid u \lhd l.P \mid u \rhd \{l_i : P_i\}_{i\in I} & \text{label select \& branch} \\
& \mid \text{if } [u = v] \text{ then } P \text{ else } P' & \text{matching} \\
u, v & ::= a \mid x \mid \text{true} \mid \text{false} & \text{values}
\end{array}
$$

Runtime syntax

$$
\begin{array}{rll}
Q & ::= \cdots \mid (\nu k)Q & \text{channel binder} \\
u, v & ::= \cdots \mid k & \text{session channel}
\end{array}
$$

Semantics

Runtime generation of sessions

- accept $a(x).Q \mid$ request $a(y).Q' \longrightarrow (\nu k)(Q[k/x] \mid Q'[k/y])$

Session passing:

- $k![k'].Q \mid k?(x).Q' \longrightarrow Q \mid Q'[k'/x]$

Semantics

Runtime generation of sessions

- accept $a(x).Q \mid$ request $a(y).Q' \longrightarrow (\nu k)(Q[k/x] \mid Q'[k/y])$

Session passing:

- $k![k'].Q \mid k?(x).Q' \longrightarrow Q \mid Q'[k'/x]$

Label selection

- $k \lhd l_j.Q \mid k \rhd \{l_i : Q_i\}_{i \in I} \longrightarrow Q \mid \; Q_j \qquad (j \in I)$

Matching:

- if $[a = a]$ then Q else $Q' \longrightarrow Q$

Struct:

- $Q' \equiv Q_1$ and $Q_1 \longrightarrow Q_2$ and $Q_2 \equiv Q'' \; \Rightarrow \; Q' \longrightarrow Q''$

Type System for Top-level Processes

Session Types

$$T \ ::= \ ?[S].T \ \mid ?[T].T \ \mid \ \&\{l_i\colon T_i\}_{i\in I} \ \mid \ \text{end}$$
$$\mid \ ![S].T \ \mid \ ![T].T \ \mid \ \oplus\{l_i\colon T_i\}_{i\in I} \ \mid \ t \ \mid \ \mu t.T$$

Type System for Top-level Processes

Session Types

$$T \ ::= ?[S].T \ \mid ?[T].T \ \mid \ \&\{l_i \colon T_i\}_{i\in I} \ \mid \ \text{end}$$
$$\mid \ ![S].T \ \mid ![T].T \ \mid \ \oplus\{l_i \colon T_i\}_{i\in I} \ \mid \ t \ \mid \ \mu t.T$$

Duality

- $\overline{T}$ exchanges ! with ?, and & with $\oplus$

$$\overline{?[\alpha].T} = ![\alpha].\overline{T} \quad \overline{\oplus\{l_i \colon T_i\}_{i\in I}} = \&\{l_i \colon \overline{T_i}\}_{i\in I} \qquad \overline{\text{end}} = \text{end}$$
$$\overline{![\alpha].T} = ?[\alpha].\overline{T} \quad \overline{\&\{l_i \colon T_i\}_{i\in I}} = \oplus\{l_i \colon \overline{T_i}\}_{i\in I} \quad \overline{\mu X.T} = \mu X.\overline{T} \quad \overline{X} =$$

Type environments

- Δ is a session environment, Γ is unrestricted

Typing Session Delegation

Merge

- Commutative relation s.t. $T \otimes \text{end} = T$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot u : U \cdot v : T_1 \quad T_1 = T \otimes T_1}{\Gamma \vdash u![v].P \triangleright \Delta \cdot u : ![T].U \cdot v : T}$$

Typing Session Delegation

Merge

- Commutative relation s.t. $T \otimes \text{end} = T$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot u : U \cdot v : T_1 \quad T_1 = T \otimes T_1}{\Gamma \vdash u![v].P \triangleright \Delta \cdot u : ![T].U \cdot v : T}$$

Example

- Client query for trust of a channel :

$$\Gamma \vdash z![y^{\text{end}}].z \triangleright \{\text{yes} : P(y^{T_1}), \text{no} : Q(y^{T_1})\} \triangleright z :![\text{end}].U, y : \text{end}$$

- Client signaling a channel :

$$\Gamma \vdash x![y^{T}].\text{request } d(z).z \triangleleft \text{put}.z![y^{\text{end}}] \triangleright x :![T].U, y : T$$

Typing Composition

Rule

$$\frac{\Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash P' \triangleright \Delta'}{\Gamma \vdash P \mid P' \triangleright \Delta \otimes \Delta'}$$

Typing Composition

Rule

$$\frac{\Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash P' \triangleright \Delta'}{\Gamma \vdash P \mid P' \triangleright \Delta \otimes \Delta'}$$

- We type more processes
  - $P = x![y^T].P' \mid z![y^{\text{end}}].P''$
  - $P = x?().P_1 \mid$ if $[x^{\text{end}} = y^T]$ then P' else P''
- T and end in general not dual

Typing matching

Rules for Values

$$\frac{\Gamma \vdash u\colon S, v\colon S}{\Gamma \vdash [u = v]\colon \text{bool}} \qquad \frac{\Delta \vdash u\colon T, v\colon T' \quad (T \otimes T') \downarrow \text{ or } T = T'}{\Delta \vdash [u = v]\colon \text{bool}}$$

Typing matching

Rules for Values

$$\frac{\Gamma \vdash u\colon S, v\colon S}{\Gamma \vdash [u = v]\colon \text{bool}} \qquad \frac{\Delta \vdash u\colon T, v\colon T' \quad (T \otimes T')\downarrow \text{ or } T = T'}{\Delta \vdash [u = v]\colon \text{bool}}$$

Typing processes

$$\frac{\Delta \vdash [u = v]\colon \text{bool} \quad \Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash P' \triangleright \Delta}{\Gamma \vdash \text{if } [u = v] \text{ then } P \text{ else } P' \triangleright \Delta}$$

Example

- $\Gamma \vdash \text{if } [u = v] \text{ then } P \text{ else } P' \triangleright \Delta \cdot u : T, v : \text{end}$

Other Distinguishing Rules

Typing Inhert Process

- $\Gamma \vdash \mathbf{0} \triangleright \Delta$

More liberal

- We do not require linear resources to be depleted (at type end)
- Example: $\Gamma \vdash \mathbf{0} \triangleright x : T$

Other Distinguishing Rules

Typing Inhert Process

- $\Gamma \vdash \mathbf{0} \triangleright \Delta$

More liberal

- We do not require linear resources to be depleted (at type end)
- Example: $\Gamma \vdash \mathbf{0} \triangleright x : T$

Typing Recursion

$$\frac{\Gamma \vdash \tilde{e} : \tilde{S}}{\Gamma \cdot X : \tilde{S}\tilde{T} \vdash X[\tilde{e}\tilde{u}] \triangleright \Delta \cdot \tilde{u} : \tilde{T}}$$

- Same considerations

Type-Safety

[Theorem] If $\Gamma \vdash P \triangleright \Delta$ and $P \longrightarrow^* Q$, then Q is not an error.

- *Proof* non-standard without using subject reduction
- Mapping $[\![\cdot]\!]$ from double-binder to base runtime language
- We use operational and error correspondence

Type-Safety

[Theorem] If $\Gamma \vdash P \triangleright \Delta$ and $P \longrightarrow^* Q$, then Q is not an error.

- *Proof* non-standard without using subject reduction
- Mapping $[\![\cdot]\!]$ from double-binder to base runtime language
- We use operational and error correspondence

Double binder runtime language

- $(\nu k)Q$ replaced by $(\nu cd)R$ with c, d distinct identifiers
- Type system require c, d to have dual types

Type-Safety

[Theorem] If $\Gamma \vdash P \triangleright \Delta$ and $P \longrightarrow^* Q$, then Q is not an error.

- *Proof* non-standard without using subject reduction
- Mapping $[\![\cdot]\!]$ from double-binder to base runtime language
- We use operational and error correspondence

Double binder runtime language

- $(\nu k)Q$ replaced by $(\nu cd)R$ with c, d distinct identifiers
- Type system require c, d to have dual types

Encoding

- $(\nu cd)R$ mapped to $(\nu k)Q$
- Occurrencies of c, d in R mapped to k in Q

Outline of the proof

1. *Typing Correspondence*.
   Typed top level are typed double binder processes.
2. *Operational Correspondence*

$$P \longrightarrow^* Q \implies \exists R \text{ such that } P \hookrightarrow^* R \text{ and } [\![R]\!] \equiv Q$$

3. *Subject Reduction of Double Binder Language*.
   We apply this result to (1,2) and infer that R *not an error*
4. *Error Correspondence*.

$$R \text{ not an error} \implies [\![R]\!] \text{ not an error}$$

5. *Error Congruence*. We glue (2) and (4) and show that

$$\left([\![R]\!] \equiv Q \text{ and } [\![R]\!] \text{ not an error}\right) \implies Q \text{ not an error}$$

The double binder runtime language

Reduction semantics defined over configurations $\sigma \diamond P$

- σ symmetric, irreflexive, functional binary relation over channels
- Store connections among free end-points

Dynamics

$$\sigma \diamond (\text{accept } a(x).R \mid \text{request } a(y).R) \longrightarrow \sigma \diamond ((\nu cd)(R[c/x] \mid R'[d/y]))$$

$$\sigma \cdot (c,d) \diamond (c![v].R \mid d?(y).R') \longrightarrow \sigma \cdot (c,d) \diamond (R \mid R'[v/y])$$

$$\sigma \cdot (c,d) \diamond (c \lhd l_j.R \mid d \rhd \{l_i : R_i\}_{i \in I}) \longrightarrow \sigma \cdot (c,d) \diamond (R \mid R_j)$$

$$\sigma \cdot (c,d) \diamond R \rightarrow \sigma \cdot (c,d) \diamond R' \Rightarrow \sigma \diamond (\nu cd)R\sigma \diamond (\nu cd)R'$$

Subject Reduction for DB

Type system adds a single new rule

$$\frac{\Gamma \vdash \sigma \cdot (c, d) \diamond R \triangleright \Delta \cdot c : T \cdot d : \overline{T}}{\Gamma \vdash \sigma \diamond (\nu cd) R \triangleright \Delta}$$

Subject Reduction for DB

Type system adds a single new rule

$$\frac{\Gamma \vdash \sigma \cdot (c,d) \diamond R \triangleright \Delta \cdot c : T \cdot d : \overline{T}}{\Gamma \vdash \sigma \diamond (\nu cd) R \triangleright \Delta}$$

Balanced environments

- Δ *balanced* by σ whenever $c\,\sigma\,d$ and $\{c,d\} \subseteq \mathrm{dom}(\Delta)$ implies that or (i) $\Delta(c) = \overline{\Delta(d)}$ or (ii) $\Delta(c) \otimes \Delta(d) \downarrow$.

Theorem

- Let $\Gamma \vdash \sigma \diamond R \triangleright \Delta$ with Δ balanced by σ. If $\sigma \diamond R \longrightarrow \sigma \diamond R'$, then there is Δ' balanced by σ s.t. $\Gamma \vdash \sigma \diamond R' \triangleright \Delta'$.
- *Proof* involved because of session passing and rule for composition merging overlapping environments

Encoding

- Σ injection over σ s.t. $(c\,\sigma\,d \wedge \Sigma(c) = k) \Rightarrow \Sigma(d) = k$

$$
\begin{aligned}
[\![\sigma \diamond \text{accept } u(x).R]\!]_\Sigma &= \text{accept } [\![u]\!]_\Sigma(x).[\![\sigma \diamond R]\!]_\Sigma \\
[\![\sigma \diamond u?(x).R]\!]_\Sigma &= [\![u]\!]_\Sigma?(x).[\![\sigma \diamond R]\!]_\Sigma \\
[\![\sigma \diamond u![e].R]\!]_\Sigma &= [\![u]\!]_\Sigma![[\![e]\!]_\Sigma].[\![\sigma \diamond R]\!]_\Sigma \\
[\![\sigma \diamond (\nu a)R]\!]_\Sigma &= (\nu a)[\![\sigma \diamond R]\!]_\Sigma \\
[\![\sigma \diamond (\nu cd)R]\!]_\Sigma &= (\nu k)[\![\sigma \cdot (c,d) \diamond R]\!]_{\Sigma\cdot(c\to k, d\to k)} \\
[\![\sigma \diamond u \lhd l_j.R]\!]_\Sigma &= [\![u]\!]_\Sigma \lhd l_j.[\![\sigma \diamond R]\!]_\Sigma \\
[\![\sigma \diamond \text{if } e \text{ then } R \text{ else } R']\!]_\Sigma &= \text{if } [\![e]\!]_\Sigma \text{ then } [\![R]\!]_\Sigma \text{ else } [\![R]\!]'_\Sigma \\
[\![\sigma \diamond R \mid R']\!]_\Sigma &= [\![\sigma \diamond R]\!]_\Sigma \mid [\![\sigma \diamond R']\!]_\Sigma
\end{aligned}
$$

Proof of main result

Type Safety

- Typed top level processes do not reduce to errors

$$k![] \mid k \triangleleft u.Q \qquad k![].Q \mid k?(x).Q' \mid k \triangleright \{l_i : Q_i\}_{i \in I}$$

Proof of main result

Type Safety

- Typed top level processes do not reduce to errors

$$k![\,] \mid k \lhd u.Q \qquad k![\,].Q \mid k?(x).Q' \mid k \rhd \{l_i : Q_i\}_{i \in I}$$

Proof. Let P top level, $P \longrightarrow^* Q$ and $\Gamma \vdash P \triangleright \Delta$

- (OC) $\exists R$ such $\emptyset \diamond P \longrightarrow^* \emptyset \diamond R$ and $[\![\emptyset \diamond R]\!] \equiv Q$
- (TC) $\Gamma \vdash \emptyset \diamond P \triangleright \Delta$
- (SR) $\emptyset \diamond R$ not an error (Δ balanced since $\emptyset$)
- (EC) Q not an error

Proof of main result

Type Safety

- Typed top level processes do not reduce to errors

$$k![] \mid k \lhd u.Q \qquad k![].Q \mid k?(x).Q' \mid k \rhd \{l_i : Q_i\}_{i \in I}$$

Proof. Let P top level, $P \longrightarrow^* Q$ and $\Gamma \vdash P \triangleright \Delta$

- (OC) $\exists R$ such $\emptyset \diamond P \longrightarrow^* \emptyset \diamond R$ and $[\![\emptyset \diamond R]\!] \equiv Q$
- (TC) $\Gamma \vdash \emptyset \diamond P \triangleright \Delta$
- (SR) $\emptyset \diamond R$ not an error (Δ balanced since $\emptyset$)
- (EC) Q not an error

Not errors:

$$k?(x).Q \mid \text{if } [k^{\text{end}} = v] \text{ then } Q' \qquad k_1![k^{\text{end}}].Q_1 \mid k?().Q \qquad k?().k![]$$

Contribution

Session-based pi calculi

- type safety for polarity-free pi calculus wirh sessions
- clean type theory for session-based communication

Generalization of session types

- typing rule for session delegation relaxed
- there is “life” after passing a session

Discussion

Polarities

- low-level details for type safety invisible to programmer
- inferring in presence of overlapping type environments in composition could be tricky
- more suitable for asynchronous implementations

Proof technique

- proof should be valid for session calculi based on pi (e.g., SSCC,CASPIS)
- only typings and semantics for new session generation are involved

Discussion

Typing inhert processes

- many works require linear resources to be depleted (at type end)
- to preserve typing congruence?

$$P \mid \mathbf{0} \equiv P \qquad \Gamma \vdash P \mid \mathbf{0} \triangleright \Delta \Leftrightarrow \Gamma \vdash P \triangleright \Delta$$

- our approach possible in many systems
  - e.g. non-overlapping type environments in composition

Session Types

- Future work: merge operation could be more flexible