Pi Calculus with Sessions is Type-Safe

M. Giunti

joint work with

K.Honda, N.Yoshida and V.T.Vasconcelos

SENSORIA Workshop, IMT Lucca, Sept. 17-18 2008

Abstract

Starting point

- [HVK-ESOP98] introduces a typed pi calculus for structured programming (dual types)
- Calculus admits *fresh session passing*

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

- Removing the side condition breaks subject reduction
  - Type safety?

Abstract

Starting point

- [HVK-ESOP98] introduces a typed pi calculus for structured programming (dual types)
- Calculus admits *fresh session passing*

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

- Removing the side condition breaks subject reduction
  - Type safety?

This talk

- [HVK-ESOP98] with general session passing is type-safe

Plan of the Talk

- Background: typed pi,session passing and subject reduction
- Pi calculus with runtime sessions
- Proof of type-safety

Pi with sessions [HVK-ESOP98]

P	::= request $a(x).P$	session request
	$\mid$ accept $a(x).P$	session acceptance
	$\mid u![e].P$	value sending
	$\mid u?(x).P$	value reception
	$\mid u \lhd l.P$	label selection
	$\mid u \rhd \{l_i : P_i\}_{i \in I}$	label branching
	$\mid (\nu u)P$	restriction
	$\mid \cdots$	pi calculus

- u range over names $a, \cdots, x$ (unstructured comm.) and channels k (struct. comm.)

Semantics and typing system

Runtime generation of sessions:

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

Session passing

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

Semantics and typing system

Runtime generation of sessions:

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

Session passing

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

Type system

- $\Gamma \vdash P \triangleright \Delta$ with $a \in \mathrm{dom}(\Gamma)$, $\{a,k\} \subseteq \mathrm{dom}(\Delta)$
- Duality: $\overline{?[S].T} = ![S].\overline{T} \quad \overline{\oplus\{l_i : T_i\}_{i\in I}} = \&\{l_i : \overline{T_i}\}_{i\in I}$
- Composition

$$\Gamma \vdash P \mid Q \triangleright \Delta_P \circ \Delta_Q \text{ provided } \Delta_P \asymp \Delta_Q$$

$$\Delta \asymp \Delta' \text{ iff } \forall u \in (\mathrm{dom}(\Delta) \cap \mathrm{dom}(\Delta')) \,.\, \Delta(u) = \overline{\Delta'(u)}$$

Discussion

What happened after [HVK-ESOP98]?

- many papers used
  - general session passing (session channel transmitted may occur in free channels of receiver)
  - same type system
- subject reduction lost

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

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

Discussion

What happened after [HVK-ESOP98]?

- many papers used
  - general session passing (session channel transmitted may occur in free channels of receiver)
  - same type system
- subject reduction lost

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

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

This means we loose type-safety?

- typed processes do not reduce to errors
- errors: $(k![k'].P \mid k![k''].Q)$, $(k?(x).P \mid k \triangleright \{l_i : P_i\}_{i \in I}), \cdots$

Extension [YV-SECRET06]

General session passing

- channels are polarized [GH-ActaInf05]
- accept $a(x).P \mid$ request $a(x).P \rightarrow (\nu k)(P[k^+/x] \mid Q[k^-/y])$
- $k^+![k^p].P \mid k^-?(x).Q \rightarrow P \mid Q[k^p/x]$

Extension [YV-SECRET06]

General session passing

- channels are polarized [GH-ActaInf05]
- accept $a(x).P \mid$ request $a(x).P \rightarrow (\nu k)(P[k^+/x] \mid Q[k^-/y])$
- $k^+![k^p].P \mid k^-?(x).Q \rightarrow P \mid Q[k^p/x]$

Type system changed

- Composition

$$\frac{\Gamma \vdash P \triangleright \Delta_P \qquad \Gamma \vdash Q \triangleright \Delta_Q \quad \Delta_P, \Delta_Q \text{ disjoint}}{\Gamma \vdash P \mid Q \triangleright \Delta_P, \Delta_Q}$$

- New channel

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot k^+ : T, k^- : \overline{T}}{\Gamma \vdash (\nu k)P \triangleright \Delta}$$

Type safety for [YV-SECRET06]

Subject reduction and type safety for balanced typings

$$\Delta(k^+) = \overline{\Delta(k^-)}$$

- Counter-example to [HVK-ESOP98] recovered

$$P = q^+![k^+] \mid q^-?(x).x?().k^-![]$$
$$\Delta_1 = q^+ :![?], k^+ :?$$
$$\Delta_2 = q^- :?[?], k^- :!$$
$$\Delta = \Delta_1, \Delta_2$$

- $\emptyset \vdash P \triangleright \Delta$ with Δ balanced and $P \longrightarrow P' = k^+?().k^-![]$
- There is Δ' balanced s.t. $\emptyset \vdash P' \triangleright \Delta'$
  - $\Delta' = k^+ :?, k^- :!$

Comparison

ESOP'98	Syntax: names and channels Fresh session passing $\Gamma \vdash P \triangleright \Delta$ and $P \longrightarrow^* P'$ imply P' not an error
SECRET'06	Syntax: names and polarized channels General session passing $\Gamma \vdash P \triangleright \Delta$, Δ balanced, $P \longrightarrow^* P'$ imply P' not an error
THIS TALK	Top-level syntax: names Runtime syntax: names and channels General session passing $\Gamma \vdash P \triangleright \Delta$ (P top-level) and $P \longrightarrow^* P' \Rightarrow P'$ not an error

Pi calculus with runtime sessions

- Top-level syntax defined over names u

$$
\begin{array}{rll}
P & ::= \text{request } a(x).P & \text{session request} \\
& \mid \text{accept } a(x).P & \text{session acceptance} \\
& \mid u![e].P & \text{value sending} \\
& \mid u?(x).P & \text{value reception} \\
& \mid u \lhd l.P & \text{label selection} \\
& \mid u \rhd \{l_i : P_i\}_{i \in I} & \text{label branching} \\
& \mid (\nu u)P & \text{restriction} \\
& \mid \cdots & \text{pi calculus}
\end{array}
$$

- Runtime syntax: u range over names and channels

Semantics and type system

Runtime generation of sessions

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

Session passing:

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

Semantics and type system

Runtime generation of sessions

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

Session passing:

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

Type system for top-level processes

- $\Gamma \vdash P \triangleright \Delta$ (Δ containing only names)

Composition and duality:

$$\frac{\Gamma \vdash P \triangleright \Delta_P \qquad \Gamma \vdash Q \triangleright \Delta_Q \quad \Delta_P, \Delta_Q \text{ disjoint}}{\Gamma \vdash P \mid Q \triangleright \Delta_P, \Delta_Q}$$

$$\frac{\Gamma \vdash a\colon \langle T \rangle \qquad \Gamma \vdash P \triangleright \Delta, x\colon T}{\Gamma \vdash \text{accept } a(x).P \triangleright \Delta} \qquad \frac{\Gamma \vdash a\colon \langle T \rangle \qquad \Gamma \vdash P \triangleright \Delta, x\colon \overline{T}}{\Gamma \vdash \text{request } a(x).P \triangleright \Delta}$$

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.

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.

Outline:

- We define a new syntax and semantics for runtime processes
- We prove subject reduction and type safety for the new language
- We encode newer runtime processes into older ones
- We prove operational and error correspondence
  - $\exists R \,.\, P \longrightarrow^*_{\mathsf{db}} R \;\wedge\; [\![R]\!] \equiv Q$
  - R not an error *implies* $[\![R]\!]$ not an error

The double binder runtime language

New runtime syntax

- Binder $(\nu k)P$ subtituted with a *double binder* $(\nu k_1 k_2)P$

  (this is not $(\nu k_1, k_2)P$!)
- The double binder associate the two end-points of the session (the channels k_1, k_2)

The double binder runtime language

New runtime syntax

- Binder $(\nu k)P$ subtituted with a *double binder* $(\nu k_1 k_2)P$

  (this is not $(\nu k_1, k_2)P$!)
- The double binder associate the two end-points of the session (the channels k_1, k_2)

Semantics defined over configurations $\sigma \diamond P$

- σ binary relation over channels, keeps track of free end-points

$$\sigma \diamond \text{accept } a(x).P \mid \text{request } a(y).Q \longrightarrow \sigma \diamond (\nu k_1 k_2)(P[k_1/x] \mid Q[k_2/y])$$

$$\sigma \models k_1 \leftrightarrow k_2 \Rightarrow \sigma \diamond k_1![k].P \mid \; k_2?(y).Q \longrightarrow \sigma \diamond P \mid Q[k/y]$$

$$\sigma, k_1 \leftrightarrow k_2 \diamond P \longrightarrow \sigma, k_1 \leftrightarrow k_2 \diamond P' \Rightarrow \sigma \diamond (\nu k_1 k_2)P \longrightarrow \sigma \diamond (\nu k_1 k_2)P'$$

Subject reduction and type safety

Typing system

$$\frac{\Gamma \vdash \sigma, k_1 \leftrightarrow k_2 \diamond P \triangleright \Delta, k_1 : T, k_2 : \overline{T}}{\Gamma \vdash \sigma \diamond (\nu k_1 k_2) P \triangleright \Delta}$$

Subject reduction

- Let $\Gamma \vdash \sigma \diamond P \triangleright \Delta$ and $\sigma \models \Delta$. If $\sigma \diamond P \rightarrow \sigma \diamond Q$, then there is Δ' inferred from Δ, σ, P and Q s.t. $\Gamma \vdash \sigma \diamond Q \triangleright \Delta'$ and $\sigma \models \Delta'$.
  - $\sigma \models \Delta$ balances typings of the channels related by σ

Type safety

- Let $\sigma \diamond P$ be a configuration and assume $\Gamma \vdash \sigma \diamond P \triangleright \Delta$ with $\sigma \models \Delta$. If $\sigma \diamond P \rightarrow^* \sigma \diamond Q$, then $\sigma \diamond Q$ is not an error.
  - By using subject reduction and $\Gamma \vdash \sigma \diamond Q \triangleright \Delta'$ with $\sigma \models \Delta'$ implies $\sigma \diamond Q$ not an error.

The encoding

We let Σ be an injective map from σ to channels not in $\mathrm{fc}(\sigma)$ (noted $\Sigma \bowtie \sigma$)

$$
\begin{array}{rcll}
[\![\sigma \diamond k?(x).P]\!]_\Sigma & = & [\![k]\!]_\Sigma?(x).[\![\sigma \diamond P]\!]_\Sigma & \\
[\![\sigma \diamond k_1![k_2].P]\!]_\Sigma & = & [\![k_1]\!]_\Sigma![[\![k_2]\!]_\Sigma].[\![\sigma \diamond P]\!]_\Sigma & \\
[\![\sigma \diamond (\nu k_1 k_2)P]\!]_\Sigma & = & (\nu k)[\![\sigma, k_1 \leftrightarrow k_2 \diamond P]\!]_{\Sigma,(k_1 \leftrightarrow k_2)\to k} & k \notin \mathrm{fc}(P)
\end{array}
$$

where $[\![k_1]\!]_\Sigma = k$ when $\Sigma(k_1 \leftrightarrow k_2) = k \vee \Sigma(k_2 \leftrightarrow k_1) = k$.

The encoding

We let Σ be an injective map from σ to channels not in $\mathrm{fc}(\sigma)$ (noted $\Sigma \bowtie \sigma$)

$$
\begin{array}{rcll}
[\![\sigma \diamond k?(x).P]\!]_\Sigma & = & [\![k]\!]_\Sigma?(x).[\![\sigma \diamond P]\!]_\Sigma & \\
[\![\sigma \diamond k_1![k_2].P]\!]_\Sigma & = & [\![k_1]\!]_\Sigma![[\![k_2]\!]_\Sigma].[\![\sigma \diamond P]\!]_\Sigma & \\
[\![\sigma \diamond (\nu k_1 k_2)P]\!]_\Sigma & = & (\nu k)[\![\sigma, k_1 \leftrightarrow k_2 \diamond P]\!]_{\Sigma,(k_1 \leftrightarrow k_2)\to k} & k \notin \mathrm{fc}(P)
\end{array}
$$

where $[\![k_1]\!]_\Sigma = k$ when $\Sigma(k_1 \leftrightarrow k_2) = k \vee \Sigma(k_2 \leftrightarrow k_1) = k$.

Single-step correspondence

- Let $\Gamma \vdash \sigma \diamond P \triangleright \Delta$. If $[\![\sigma \diamond P]\!]_\Sigma \to Q$, then there is R such that $\sigma \diamond P \to \sigma \diamond R$ with $[\![\sigma \diamond R]\!]_\Sigma \equiv Q$.
  - Standard proof by induction of the lenght of $[\![\sigma \diamond P]\!]_\Sigma \to Q$.

Example

Counter-example to [HVK-ESOP'98]

$$\sigma \models q_1 \leftrightarrow q_2, k_1 \leftrightarrow k_2 \qquad P = q_1![k_1] \mid q_2?(x).x?().k_2![]$$

$$\sigma \diamond P \rightarrow \sigma \diamond k_1?().k_2![]$$

Encoding: $\Sigma(q_1 \leftrightarrow q_2) = q,\ \Sigma(k_1 \leftrightarrow k_2) = k$

$$[\![\sigma \diamond P]\!]_\Sigma = q![k] \mid q?(x).x?().k![]$$

$$[\![\sigma \diamond P]\!]_\Sigma \rightarrow k?().k![]$$

Op. correspondence

$$[\![\sigma \diamond k_1?().k_2![]]\!]_\Sigma = k?().k![]$$

The encoding /2

Typing correspondence

- Let P be a top-level process such that $\Gamma \vdash P \triangleright \Delta$. Then $\Gamma \vdash \emptyset \diamond P \triangleright \Delta$.

The encoding /2

Typing correspondence

- Let P be a top-level process such that $\Gamma \vdash P \triangleright \Delta$. Then $\Gamma \vdash \emptyset \diamond P \triangleright \Delta$.

Operational correspondence

- Let P be a top-level process such that $\Gamma \vdash P \triangleright \Delta$. If $P \longrightarrow^n Q$, then there is P' such that $\emptyset \diamond P \longrightarrow^n \emptyset \diamond P'$ and $[\![\emptyset \diamond P']\!]_\emptyset = Q$.

The encoding /2

Typing correspondence

- Let P be a top-level process such that $\Gamma \vdash P \triangleright \Delta$. Then $\Gamma \vdash \emptyset \diamond P \triangleright \Delta$.

Operational correspondence

- Let P be a top-level process such that $\Gamma \vdash P \triangleright \Delta$. If $P \rightarrow^n Q$, then there is P' such that $\emptyset \diamond P \rightarrow^n \emptyset \diamond P'$ and $[\![\emptyset \diamond P']\!]_\emptyset = Q$.

Error Correspondence

- Let $\sigma \diamond P$ be a double binder configuration and assume $\Sigma \bowtie \sigma$. If $\sigma \diamond P$ is not an error then $[\![\sigma \diamond P]\!]_\Sigma$ is not an error.

The encoding /2

Typing correspondence

- Let P be a top-level process such that $\Gamma \vdash P \triangleright \Delta$. Then $\Gamma \vdash \emptyset \diamond P \triangleright \Delta$.

Operational correspondence

- Let P be a top-level process such that $\Gamma \vdash P \triangleright \Delta$. If $P \rightarrow^n Q$, then there is P' such that $\emptyset \diamond P \rightarrow^n \emptyset \diamond P'$ and $[\![\emptyset \diamond P']\!]_\emptyset = Q$.

Error Correspondence

- Let $\sigma \diamond P$ be a double binder configuration and assume $\Sigma \bowtie \sigma$. If $\sigma \diamond P$ is not an error then $[\![\sigma \diamond P]\!]_\Sigma$ is not an error.

[Type safety for top-level processes] Let $\Gamma \vdash P \triangleright \Delta$ with P a top-level process. If $P \rightarrow^* Q$, then Q is not an error.

- By using typing, operational and error correspondence and type safety of the double binder language

Conclusion

Main result

- We proved type-safety for pi with sessions
  - limitation(?): programmer use names, session generation and control due to the system
- Goal: provide a type-safe language for sessions with clean syntax and semantics (pi calculus)
  - double bindings/polarities low-level details for type safety

Conclusion

Main result

- We proved type-safety for pi with sessions
  - limitation(?): programmer use names, session generation and control due to the system
- Goal: provide a type-safe language for sessions with clean syntax and semantics (pi calculus)
  - double bindings/polarities low-level details for type safety

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