Session Types Revisited

Ornela Dardha

School of Computing Science
University of Glasgow

January 10, 2014

(Joint work with Elena Giachino and Davide Sangiorgi)
Gentle Intro

- In complex distributed systems communicating participants agree on a protocol to follow, specifying type and direction of data exchanged.

- **Session types** are a formalism to model structured communication-based programming.

- Designed for
  - process calculi
  - multithreaded functional languages
  - object-oriented languages
  - ...

- Guarantee privacy, communication safety and session fidelity.
Example of Session Types

Distributed Auction System:
sellers, that want to sell items,
auctioneers, that sell items on their behalf,
bidders, that bid for an item being auctioned.

seller: $\oplus\{\texttt{selling} : !\text{Item} . !\text{Price} . \& \{\texttt{sold} : ?\text{Price} . \text{end}, \texttt{not} : \text{end}\}\}$

auctioneer: $\&\{\texttt{selling} : ?\text{Item} . ?\text{Price} . \oplus \{\texttt{sold} : !\text{Price} . \text{end}, \texttt{not} : \text{end}\}\}$,

register: $?\text{Id} . !\text{Item} . ?\text{Bid} . \text{end}$

bidder: $\oplus\{\texttt{register} : !\text{Id} . ?\text{Item} . !\text{Bid} . \text{end}\}$
Key words for sessions

1. **Sequentiality** of input/output operations explicitly indicating type of data transmitted.  
   (Guarantees *session fidelity*)
Key words for sessions

1. **Sequentiality** of input/output operations explicitly indicating type of data transmitted.
   (Guarantees *session fidelity*)

2. **Duality** of session types corresponding to opposite endpoints of a session channel.
   (Guarantees *communication safety*)
Key words for sessions

1. **Sequentiality** of input/output operations explicitly indicating type of data transmitted. (Guarantees *session fidelity*)

2. **Duality** of session types corresponding to opposite endpoints of a session channel. (Guarantees *communication safety*)

3. **Connection** establishes a fresh session channel between two parties. (Guarantees *privacy*)
Background on $\pi$ types

- Channel type $\#T$: types a channel used in input or output to transmit values of type $T$, many times.
Background on $\pi$ types

- Channel type $\#T$: types a channel used in input or output to transmit values of type $T$, many times.
- Input/output channel type $iT/oT$: types a channel used *only* in input/output to transmit values of type $T$, many times.
Background on $\pi$ types

- Channel type $\# T$: types a channel used in input or output to transmit values of type $T$, many times.
- Input/output channel type $iT/oT$: types a channel used only in input/output to transmit values of type $T$, many times.
- Linear input/output type $\ell_i T/\ell_o T$: types a channel used only in input/output and exactly once to transmit values of type $T$. 
Background on $\pi$ types

- Channel type $\# T$: types a channel used in input or output to transmit values of type $T$, many times.
- Input/output channel type $iT/oT$: types a channel used only in input/output to transmit values of type $T$, many times.
- Linear input/output type $\ell_i T/\ell_o T$: types a channel used only in input/output and exactly once to transmit values of type $T$.
- Linearized channel: linear channel used multiple times but only in a sequential manner ($?, !$) and with the same carried type.
Background on $\pi$ types

- Channel type $\# T$: types a channel used in input or output to transmit values of type $T$, many times.
- Input/output channel type $iT/oT$: types a channel used only in input/output to transmit values of type $T$, many times.
- Linear input/output type $\ell_i T/\ell_o T$: types a channel used only in input/output and exactly once to transmit values of type $T$.
- Linearized channel: linear channel used multiple times but only in a sequential manner (?!, !) and with the same carried type.
- Variant type: labelled disjoint union of types ($\&$, $\oplus$).
Key words for $\pi$

We saw:

1. Sequentiality
2. Duality
3. Connection
Key words for $\pi$

We saw:

1. Sequentiality
2. Duality
3. Connection

4. Linearity forces a $\pi$ channel to be used exactly once.
Key words for $\pi$

We saw:

1. Sequentiality
2. Duality
3. Connection

1. **Linearity** forces a $\pi$ channel to be used exactly once.
2. **Capability** of input/output of the same $\pi$ channel split between two partners.
Key words for $\pi$

We saw:

1. Sequentiality
2. Duality
3. Connection

1. **Linearity** forces a $\pi$ channel to be used exactly once.
2. **Capability** of input/output of the same $\pi$ channel split between two partners.
3. **Restriction** construct permits the creation of fresh private $\pi$ channels.
Standard $\pi$-types

$\tau ::= \emptyset[\tilde{T}]$ channel with no capability
$l_i [\tilde{T}]$ linear input
$l_o [\tilde{T}]$ linear output
$l_\# [\tilde{T}]$ linear connection

$T ::= \tau$ linear channel type
$\langle l_i - T_i \rangle_{i \in I}$ variant type
$\# T$ standard channel type
$\text{Bool}$ boolean type
$\cdots$ other constructs
Standard $\pi$-processes

\[ P, Q ::= \begin{array}{ll}
0 & \text{inaction} \\
x!(\tilde{v}).P & \text{output} \\
x?(\tilde{y}).P & \text{input} \\
P \mid Q & \text{composition} \\
(\nu x)P & \text{channel restriction} \\
\text{case } v \text{ of } \{l_i x_i \triangleright P_i\}_{i \in I} & \text{case process} \\
\end{array} \]

\[ v ::= x \quad \text{variable} \\
b \quad \text{boolean values} \\
l.v \quad \text{variant value} \]
Semantics

Just to understand *case normalisation*...

\[(R_{\pi-\text{COM}})\]

\[x!\langle\tilde{v}\rangle.P \mid x?(\tilde{z}).Q \rightarrow P \mid Q[\tilde{v}/\tilde{z}]\]

\[(R_{\pi-\text{CASE}})\]

\[\text{case } l_j\cdot v \text{ of } \{l_i\cdot x_i \triangleright P_i\}_{i \in I} \rightarrow P_j[v/x_j] \quad j \in I\]
Session Types

\[
S ::= \begin{array}{ll}
\text{end} & \text{termination} \\
\bang T.S & \text{send} \\
\question T.S & \text{receive} \\
\uplus \{l_i : S_i\}_{i \in I} & \text{select} \\
\& \{l_i : S_i\}_{i \in I} & \text{branch}
\end{array}
\]

\[
T ::= \begin{array}{ll}
S & \text{session type} \\
\# T & \text{standard channel type} \\
\text{Bool} & \text{boolean type} \\
\ldots & \text{other constructs}
\end{array}
\]
Session Processes

\[ P, Q ::= \begin{align*} & 0 \quad \text{inaction} \\
& x!\langle v \rangle.P \quad \text{output} \\
& x?(y).P \quad \text{input} \\
& x \triangle l_j.P \quad \text{selection} \\
& x > \{ l_i : P_i \}_{i \in I} \quad \text{branching} \\
& P \mid Q \quad \text{composition} \\
& (\nu x y)P \quad \text{session restriction} \\
& (\nu x)P \quad \text{channel restriction} \end{align*} \]

\[ \nu ::= \begin{align*} & x \quad \text{variable} \\
& b \quad \text{boolean values} \end{align*} \]
Types Encoding

[Bool]  def  =  Bool
[end]    def  =  ∅[
[!T.S]   def  =  l_o [[[T], [S]]
[?T.S]   def  =  l_i [[[T], [S]]
[⊕{l_i : T_i}_{i ∈ I}] def  =  l_o [⟨l_i¬[[T_i]]⟩_{i ∈ I}]
[&{l_i : T_i}_{i ∈ I}] def  =  l_i [⟨l_i¬[T_i]⟩_{i ∈ I}]
Example of Encoding: Types 1/2

Let \( x : T \) and \( y : \overline{T} \) where

\[
T = ?\text{Int.}?\text{Int.}!\text{Bool}.\text{end}
\]

and

\[
\overline{T} = !\text{Int.}!\text{Int.}?\text{Bool}.\text{end}
\]
Example of Encoding: Types 2/2

The encoding of these types is as follows:

\[
[T] = \ell_i [\text{Int}, \ell_i [\text{Int}, \ell_o [\text{Bool}, \emptyset]]]]
\]

and

\[
[\overline{T}] = \ell_o [\text{Int}, \ell_i [\text{Int}, \ell_o [\text{Bool}, \emptyset]]]]
\]
Example of Encoding: Types 2/2

The encoding of these types is as follows:

$$\llbracket T \rrbracket = \ell_i [\text{Int}, \ell_i [\text{Int}, \ell_o [\text{Bool}, \emptyset[]]]]$$

and

$$\llbracket \overline{T} \rrbracket = \ell_o [\text{Int}, \ell_i [\text{Int}, \ell_o [\text{Bool}, \emptyset[]]]]$$

**NB**

*duality on session types boils down to opposite capabilities (i/o) of channel types, only in the outermost level!***
Terms Encoding

\[
\begin{align*}
\llbracket x \rrbracket_f &= f_x \\
\llbracket b \rrbracket_f &= b \\
\llbracket 0 \rrbracket_f &= 0 \\
\llbracket x ! \langle v \rangle . P \rrbracket_f &= (\nu c) f_x ! \langle v, c \rangle . \llbracket P \rrbracket_f, \{ x \mapsto c \} \\
\llbracket x ? (y) . P \rrbracket_f &= f_x ? (y, c) . \llbracket P \rrbracket_f, \{ x \mapsto c \} \\
\llbracket x \triangleleft l_j . P \rrbracket_f &= (\nu c) f_x ! \langle l_j . c \rangle . \llbracket P \rrbracket_f, \{ x \mapsto c \} \\
\llbracket x \triangleright \{ l_i : P_i \}_{i \in I} \rrbracket_f &= f_x ? (y) . \text{case } y \text{ of } \{ l_i . c \triangleright \llbracket P_i \rrbracket_f, \{ x \mapsto c \} \}_{i \in I} \\
\llbracket P \mid Q \rrbracket_f &= \llbracket P \rrbracket_f \mid \llbracket Q \rrbracket_f \\
\llbracket (\nu x y) P \rrbracket_f &= (\nu c) \llbracket P \rrbracket_f, \{ x, y \mapsto c \}
\end{align*}
\]
Example of Encoding: Terms 1/2

server \overset{\text{def}}{=} x?(nr1).x?(nr2).x!(nr1 == nr2).0

client \overset{\text{def}}{=} y!(3).y!(5).y?(eq).0

The system is given by

$$(\nu xy)(\text{server} | \text{client})$$

The encoding of the above system is

$$[(\nu xy)(\text{server} | \text{client})]_f = (\nu z)[(\text{server} | \text{client})]_f,\{x,y\mapsto z\}$$
Example of Encoding: Terms 2/2

Where the encodings of server and client processes are as follows:

\[
\begin{align*}
\llbracket \text{server} \rrbracket_{f, \{x, y \mapsto z\}} & \overset{\text{def}}{=} \ z? (nr1, c). c? (nr2, c'). (\nu c'') c! \langle nr1 == nr2, c'' \rangle. 0 \\
\llbracket \text{client} \rrbracket_{f, \{x, y \mapsto z\}} & \overset{\text{def}}{=} (\nu c) z! \langle 3, c \rangle. (\nu c') c! \langle 5, c' \rangle. c'? (eq, c''). 0
\end{align*}
\]

Output actions create new channels \( c, c', c'' \) which are sent to the communicating party along with the value.
Guaranteeing Communication Properties

- Privacy is guaranteed because a channel is used \textit{at most} once.
- Communication safety is guaranteed because a channel is used \textit{at least} once.
- Session fidelity is guaranteed because of continuation-passing.
Theorem (Correctness of the Encoding)

\[ \Gamma \vdash P \text{ if and only if } [\Gamma]_f \vdash [P]_f. \]
Theorem (Correctness of the Encoding)

\[ \Gamma \vdash P \text{ if and only if } [\Gamma]_f \vdash [P]_f. \]

Theorem (Operational Correspondence)

Let \( P \) be a process in the \( \pi \)-calculus with sessions. The following hold.

1. If \( P \rightarrow P' \) then \( \exists Q \) such that \( [P]_f \rightarrow Q \) and \( Q \hookrightarrow [P']_f \), where \( \hookrightarrow \) denotes a structural congruence possibly extended with a case normalisation.

2. If \( [P]_f \rightarrow^{\equiv} Q \) then, \( \exists P' \) such that \( P \rightarrow P' \) and \( Q \rightarrow^* \equiv [P']_f \).
Theorem (Correctness of the Encoding)

\[ \Gamma \vdash P \text{ if and only if } \llbracket \Gamma \rrbracket_f \vdash \llbracket P \rrbracket_f. \]

Theorem (Operational Correspondence)

Let \( P \) be a process in the \( \pi \)-calculus with sessions. The following hold.

1. If \( P \rightarrow P' \) then \( \exists Q \) such that \( \llbracket P \rrbracket_f \rightarrow Q \) and \( Q \leftrightarrow \llbracket P' \rrbracket_f \), where \( \leftrightarrow \) denotes a structural congruence possibly extended with a case normalisation.

2. If \( \llbracket P \rrbracket_f \rightarrow^{\equiv} Q \) then, \( \exists P' \) such that \( P \rightarrow P' \) and \( Q \rightarrow^{*\equiv} \llbracket P' \rrbracket_{f'} \).

Corollary

*Subject Reduction* and *Type Soundness* on session types.
Extensions of the Encoding of Sessions
Subtyping

Theorem

\( T <: T' \text{ if and only if } \llbracket T \rrbracket \leq \llbracket T' \rrbracket. \)
Subtyping

Theorem

\( T <: T' \text{ if and only if } \llbracket T \rrbracket \leq \llbracket T' \rrbracket. \)

Derived from the encoding:

- Reflexivity and Transitivity of Subtyping.
- Lemmas (ex. Substitution, Narrowing...) follow from the corresponding ones in \( \pi \).
- Nothing to prove for other type constructs added.
Encoding Parametric Polymorphism

\[
\begin{align*}
[X] & \overset{\text{def}}{=} X \\
[[X; T]] & \overset{\text{def}}{=} \langle X; [T] \rangle \\
\langle T; v \rangle_f & \overset{\text{def}}{=} \langle [T]; f_v \rangle \\
\text{open } v \text{ as } (X; x) \text{ in } P_f & \overset{\text{def}}{=} \text{open } f_v \text{ as } (X; f_x) \text{ in } [P]_f
\end{align*}
\]
Encoding Bounded Polymorphism

$$[B] \overset{\text{def}}{=} B$$

$$\left[ \oplus \{ l_i(X_i <: B_i) : T_i \} \right]_{i \in I} \overset{\text{def}}{=} \ell_0 \left[ \langle l_i(X_i \leq B_i) \}_i \overset{\text{def}}{=} \ell_1 \left[ \langle l_i(X_i \leq B_i) \}_i \right.$$}

$$\left[ x \triangleleft l_j(B).P \right]_f \overset{\text{def}}{=} (\nu c).f_x! \langle l_j(B) \}_c \right).\left[ P \right]_f,\{x \mapsto c\}$$

$$\left[ x \triangleright \{ l_i(X_i <: B_i) : P_i \} \right]_{i \in I} \overset{\text{def}}{=} f_x?(y).$$

**case** y **of** \{ l_i(X_i \leq B_i) \}_c \triangleright \left[ P_i \right]_f,\{x \mapsto c\} \}
Theorem (Correctness of Typing Unpacking)

\[ \Gamma; \Delta \vdash \text{open } v \text{ as } (X; x) \text{ in } P \text{ if and only if } \]  
\[ \llbracket \Gamma; \Delta \rrbracket_f \vdash \llbracket \text{open } v \text{ as } (X; x) \text{ in } P \rrbracket_f. \]
Parametric and Bounded Polymorphism

Theorem (Correctness of Typing Unpacking)

\[ \Gamma; \Delta \vdash \text{open } v \text{ as } (X; x) \text{ in } P \text{ if and only if } \llbracket \Gamma; \Delta \rrbracket_f \vdash \llbracket \text{open } v \text{ as } (X; x) \text{ in } P \rrbracket_f. \]

Theorem (Correctness of Typing Bounded Polymorphic Processes)

\[ \Gamma; \Delta \vdash Q \text{ if and only if } \llbracket \Gamma; \Delta \rrbracket_f \vdash \llbracket Q \rrbracket_f, \text{ where either } \]
\[ Q = x \downarrow l_j(B).P, \text{ or } Q = x \uparrow \{l_i(X_i \leq B_i) : P_i\}_{i \in I}. \]
Observations

Derived from the encoding:

- Modification in the calculus as expected
- Encoding of polymorphism constructs: an homomorphism.
- Again, **Subject Reduction** and **Type Soundness** derived for free (considering just the constructs added).
Higher-Order

\[ \sigma ::= \begin{array}{l}
T \quad \text{general type} \\
\Diamond \quad \text{process type}
\end{array} \]

\[ T ::= \begin{array}{l}
T \rightarrow \sigma \quad \text{functional type} \\
T \mathbf{1} \rightarrow \sigma \quad \text{linear functional type}
\end{array} \]

\[ P ::= \begin{array}{l}
PQ \quad \text{application} \\
\nu \quad \text{values}
\end{array} \]

\[ \nu ::= \begin{array}{l}
\lambda x : T.P \quad \text{abstraction}
\end{array} \]

And encoding is an homomorphism...
Higher-Order Results

**Theorem (Correctness: Typing HO$_\pi$ Processes)**

\[ \Phi; \Gamma; S \vdash P : \sigma, \text{ if and only if } [\Phi; \Gamma; S]_f \vdash [P]_f : [\sigma]. \]
Higher-Order Results

Theorem (Correctness: Typing HO$\pi$ Processes)

$\Phi; \Gamma; S \vdash P : \sigma$, if and only if $[\Phi; \Gamma; S]_f \vdash [P]_f : [\sigma]$.

Derived from the encoding:

- Session $\pi$ augmented with cbv $\lambda$.
- Encoding of the new constructs: an homomorphism.
- Again, Subject Reduction and Type Soundness derived for free.
Presented an encoding of session types into ordinary $\pi$-types. Encoding proved faithful, in that it allows us to derive all the basic properties of session types from $\pi$-types. Encoding proved robust (Subtyping, Polymorphism, HO).
Conclusions 1/2

- Presented an encoding of session types into ordinary \( \pi \) types.
Conclusions 1/2

- Presented an encoding of session types into ordinary $\pi$ types.
- Encoding proved faithful, in that it allows us to derive all the basic properties of session types from $\pi$ types.
Conclusions 1/2

- Presented an encoding of session types into ordinary $\pi$ types.
- Encoding proved faithful, in that it allows us to derive all the basic properties of session types from $\pi$ types.
- Encoding proved robust (Subtyping, Polymorphism, HO).
Elimination of redundancy: syntax of types and terms in sessions.

Derivation of properties: subject reduction and type soundness in sessions come as straightforward corollaries from the theory of $\pi$.

Duality on session types boils down to opposite capabilities of standard channel types.

Robustness of the encoding allows us to easily obtain extensions of the session calculus.
**Conclusions 2/2**

- Elimination of redundancy: syntax of types and terms in sessions.
Conclusions 2/2

- Elimination of redundancy: syntax of types and terms in sessions.
- Derivation of properties: subject reduction and type soundness in sessions come as straightforward corollaries from the theory of $\pi$. 
Conclusions 2/2

- Elimination of redundancy: syntax of types and terms in sessions.
- Derivation of properties: subject reduction and type soundness in sessions come as straightforward corollaries from the theory of $\pi$.
- Duality on session types boils down to opposite capabilities of standard channel types.
Conclusions 2/2

- Elimination of redundancy: syntax of types and terms in sessions.
- Derivation of properties: subject reduction and type soundness in sessions come as straightforward corollaries from the theory of $\pi$.
- Duality on session types boils down to opposite capabilities of standard channel types.
- Robustness of the encoding allows us to easily obtain extensions of the session calculus.
Questions?
About the encoding...

Theorem (Operational Correspondence)

Let \( P \) be a process in the \( \pi \)-calculus with sessions. The following hold.

1. If \( P \rightarrow P' \) then, \([P]_f \rightarrow^* \equiv [P']_f\);

2. If \([P]_f \rightarrow \equiv Q\) then, \( \exists P', \mathcal{E}[] \) such that \( \mathcal{E}[P] \rightarrow \mathcal{E}[P'] \) and \( Q \rightarrow^* \equiv [P']_{f'} \), where either \( f' = f \) or \( \text{dom}(f') = \text{dom}(f) \cup \text{BV}(\mathcal{E}[]) \).
Subtyping in standard $\pi$-calculus

\[
\frac{T \leq T}{(S\pi-\text{REFL})} \quad \frac{T \leq T'}{T \leq T''} \quad \frac{T' \leq T''}{(S\pi-\text{TRANS})}
\]

\[
\frac{\tilde{T} \leq \tilde{T}'}{\ell_i [\tilde{T}] \leq \ell_i [\tilde{T}']} \quad \frac{\tilde{T}' \leq \tilde{T}}{\ell_o [\tilde{T}] \leq \ell_o [\tilde{T}']} \quad \frac{\tilde{T}' \leq \tilde{T}}{(S\pi-\text{OO})}
\]

\[
\frac{I \subseteq J \quad T_i \leq T'_j \quad \forall i \in I}{\langle l_i - T_i \rangle_{i \in I} \leq \langle l_j - T'_j \rangle_{j \in J}} \quad \frac{I \subseteq J \quad T_i \leq T'_j \quad \forall i \in I}{\langle l_i - T_i \rangle_{i \in I} \leq \langle l_j - T'_j \rangle_{j \in J}} \quad \frac{I \subseteq J \quad T_i \leq T'_j \quad \forall i \in I}{(S\pi-\text{VARIANT})}
\]
Semantics of Bounded Polymorphism

\[(\nu xy)(x \triangleleft l_j(B).P \mid y \triangleright \{ l_i(X_i \leq B_i) : P_i \}_{i \in I} \mid R) \rightarrow (\nu xy)(P \mid P_j[B/X_j] \mid R) \quad j \in I\]

\[\text{case } l_j(B) \_ \nu \text{ of } \{ l_i(X_i \leq B_i)_\_ x_i \triangleright P \}_{i \in I} \rightarrow P_j[B/X_j][\nu/x_j] \quad j \in I\]
Parametric Polymorphism

Example of polymorphism in pi with/without sessions:

\[ x : !\langle X; D \rangle.\text{end} \quad y : ?\langle X; D \rangle.\text{end} \]

\[ \vdash x!\langle \text{Int}; 5 \rangle \mid y?(z). \text{open } z \text{ as } (X; w) \text{ in } nj!\langle w \rangle \]

\[ \rightarrow \text{open } \langle \text{Int}; 5 \rangle \text{ as } (X; w) \text{ in } nj!\langle w \rangle \]

\[ \rightarrow nj!\langle 5 \rangle \]
Encoding Higher-Order

\[
\begin{align*}
\llbracket T \to \sigma \rrbracket & \overset{\text{def}}{=} \llbracket T \rrbracket \to \sigma \\
\llbracket T \to \sigma \rrbracket & \overset{\text{def}}{=} \llbracket T \rrbracket \to \sigma
\end{align*}
\]

\[
\begin{align*}
\llbracket \lambda x : T . P \rrbracket_f & \overset{\text{def}}{=} \lambda x : \llbracket T \rrbracket . [P]_f \\
\llbracket PQ \rrbracket_f & \overset{\text{def}}{=} \llbracket P \rrbracket_f \llbracket Q \rrbracket_f
\end{align*}
\]

Where \( \sigma ::= T \mid \Diamond \)
Up-side-down point of view

- Linear channel transmitting a value and a new linear channel
  \((\nu b)\bar{a}\langle\nu, b\rangle\ldots\)
Up-side-down point of view

- Linear channel transmitting a value and a new linear channel \((\nu b)\bar{a}\langle v, b \rangle \ldots\)
- Linear channel transmitting a value and itself \(\bar{a}\langle v, a \rangle \ldots\)
Up-side-down point of view

- Linear channel transmitting a value and a new linear channel $(\nu b)\overline{a}\langle v, b \rangle \ldots$
- Linear channel transmitting a value and itself $\overline{a}\langle v, a \rangle \ldots$
- Linear channel transmitting a value $\overline{a}\langle v \rangle \ldots$
Up-side-down point of view

- Linear channel transmitting a value and a new linear channel $(\nu b)\bar{a}\langle\nu, b\rangle \ldots$
- Linear channel transmitting a value and itself $\bar{a}\langle\nu, a\rangle \ldots$
- Linear channel transmitting a value $\bar{a}\langle\nu\rangle \ldots$

**NB**

*Session types are an optimisation of linear $\pi$ types.*
New typing rule for output

\[\Gamma_1 \vdash x : \ell_\circ [\tilde{T}] \quad \tilde{\Gamma}_2, x : \ell_\alpha [\tilde{S}] \vdash \tilde{v} : \tilde{T} \quad \Gamma_3, x : \ell_{\tilde{\alpha}} [\tilde{S}] \vdash P\]

\[\Gamma_1 \uplus \tilde{\Gamma}_2 \uplus \Gamma_3 \vdash x! \langle \tilde{v} \rangle . P\]