Session-based Communication Optimisation for Higher-Order Mobile Processes

TLCA 2009

Dimitris Mostrous, Nobuko Yoshida

Imperial College London

Thursday, July 2, 2009

Overview

- Session Typing for Higher-Order π-calculus with Buffered communications.
- Subtyping for Asynchrony / Session action permutation.
- Integration with mobile code (functions as values).
- Techniques from Linear Type Theory (Linear λ-calculus).
- Simulation technique for Subtyping: Recursive types, Linear Types, Partial Permutations.
- Example: Type-safe optimisation of a mobile application.

Session Types

A binary session describes a communication protocol between two parties, taking place over a single connection between them.

Kaku Takeuchi and Kohei Honda and Makoto Kubo. *An Interaction-based Language and its Typing System*. PARLE'94, LNCS 817, pages 398-413, Springer-Verlag, 1994

Many works on Sessions for Pi, Corba I/f, Functional m/t languages, Subtyping & Par. Polymorphism, Ambients, Objects, Web services.

More recently: increasingly many works on multi-party sessions.

Basic Sessions

Session type for channel a:

$$a \; : \; \langle ![\mathit{int}].![\mathit{int}].?[\mathit{bool}].\mathsf{end} \rangle$$

Basic Sessions

Session type for channel a:

$$a \; : \; \langle ![int].![int].?[bool].\mathsf{end}\rangle$$
$$a \; : \; \langle ?[int].?[int].![bool].\mathsf{end}\rangle$$

agrees with $\mathrm{HO}\pi^{s}$ code:

$$P = \overline{a}(x).x!\langle 5\rangle.x!\langle 5\rangle.x(y\!:\!bool).\mathbf{0}$$

$$Q = a(x).x(y\!:\!int).x(z\!:\!int).x!\langle \mathsf{T}\rangle.\mathbf{0}$$

$P \mid Q$ is typable

Sessions with Branching

Session type for channel a:

$$a : \langle \oplus [add : ![int].![int].?[int].\mathsf{end}, \\ grt : ![int].![int].?[bool].\mathsf{end} \rangle$$

Sessions with Branching

Session type for channel a:

$$a : \langle \oplus [add :![int].![int].?[int].\mathsf{end},\ grt :![int].![int].?[bool].\mathsf{end}\rangle$$

$$a : \langle \& [add :?[int].?[int].![int].\mathsf{end},\ grt :?[int].?[int].![bool].\mathsf{end}]\rangle$$

agrees with HOπ^s code:

$$P = \overline{a}(x).x \triangleleft add.x!\langle 5\rangle.x!\langle 5\rangle.x(y)$$

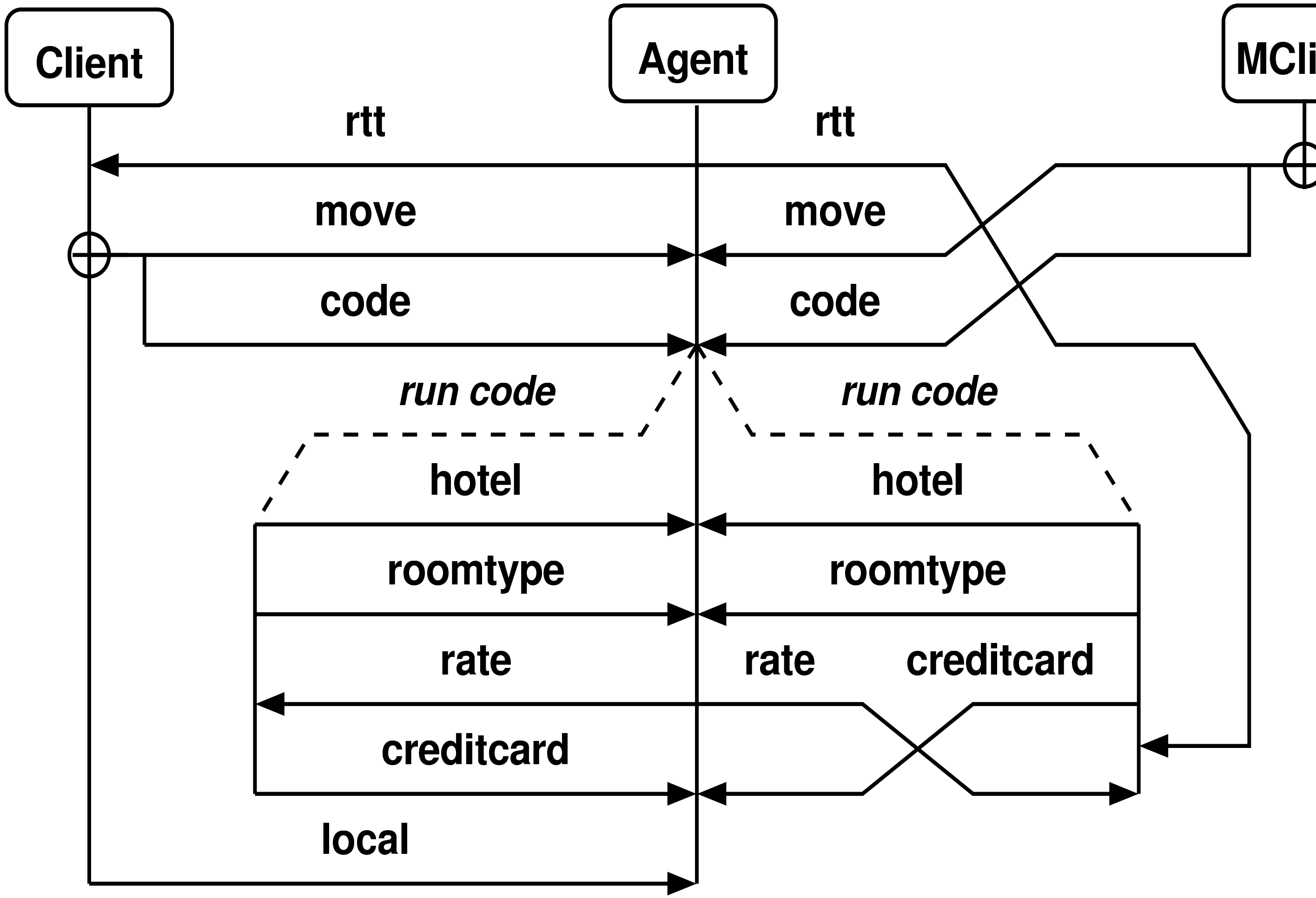
$$Q = a?(x).x \triangleright \{add : x(y).x(z).x!\langle x+z\rangle,\ grt : x(y).x(z).x!\langle x > z\rangle\}$$

Permutation

$$!\,[nat].?[int].\mathsf{end} \ll ?[int].!\,[nat].\mathsf{end}$$

$$?[int].!\,[nat].\mathsf{end} \not\ll !\,[nat].?[int].\mathsf{end}$$

Example Scenario



MClient makes use of asynchronous subtyping.

Example Scenario: The Code

$$\texttt{Client} \quad = \quad \overline{a}(x).\mathbf{x}?(\mathbf{rtt}).\mathbf{x} \lhd \mathsf{move}.$$

$$x!\langle \ulcorner x!\langle \texttt{ritz}\rangle.x!\langle \texttt{suite}\rangle.\mathbf{x}?(\mathbf{rate}).x!\langle \texttt{card}\rangle.\ldots.\urcorner\rangle$$

$$\texttt{Agent} \quad = \quad a(x).\mathbf{x}!\langle \texttt{rtt}\rangle.x \rhd \{\mathsf{move} : x?(code).(run\ code \mid Q),$$

$$\mathsf{local} : Q\,\}$$

$$Q \quad = \quad x?(hotel).x?(roomtype).\mathbf{x}!\langle \texttt{rate}\rangle.x?(creditcard)\ldots$$

$$\texttt{MClient} \quad = \quad \overline{a}(x).\mathbf{x} \lhd \mathsf{move}.$$

$$x!\langle \ulcorner x!\langle \texttt{ritz}\rangle.x!\langle \texttt{suite}\rangle.x!\langle \texttt{card}\rangle.\mathbf{x}?(\mathbf{rtt}).\mathbf{x}?(\mathbf{rate})\ldots\urcorner\rangle$$

The Language (some rules)

(beta) $(\lambda x.P)V \longrightarrow P\{^V/_x\}$

(rec) $(\mu y.\lambda x.P)V \longrightarrow P\{^V/_x\}\{^{\mu y.\lambda x.P}/_y\}$

(conn) $a(x).P \mid \overline{a}(z).Q \longrightarrow (\nu\, s)\,(P\{^s/_x\} \mid Q\{^{\overline{s}}/_z\} \mid \mathbf{s}:\varepsilon \mid \overline{\mathbf{s}}:\varepsilon)$

$s, \overline{s} \notin \mathrm{fn}(P, Q)$

(send) $s!\langle V\rangle.P \mid \overline{s}:\vec{h} \longrightarrow P \mid \overline{s}:\vec{h}\cdot V$

(recv) $s?(x).P \mid s:V\cdot\vec{h} \longrightarrow P\{^V/_x\} \mid s:\vec{h}$

(sel) $s \triangleleft l.P \mid \overline{s}:\vec{h} \longrightarrow P \mid \overline{s}:\vec{h}\cdot l$

(bra) $s \triangleright \{l_1:P_1,\ldots,l_n:P_n\} \mid s:l_m\cdot\vec{h} \longrightarrow P_m \mid s:\vec{h}$

Types

Term	T	$::=$	$U \mid \diamond$
Value	U	$::=$	$H \mid S$
HO-value	H	$::=$	$\mathsf{unit} \mid U \rightarrow T \mid U \multimap T \mid \langle S \rangle$
Session	S	$::=$	$![U].S \mid ?[U].S$
		$\mid$	$\oplus[l_1 : S_1, \ldots, l_n : S_n] \mid \&[l_1 : S_1, \ldots, l_n : S_n]$
		$\mid$	$\mu \mathbf{t}.S \mid \mathbf{t} \mid \mathsf{end}$

Duality: $\overline{![U].S} = ?[U].\overline{S}$

$$\overline{\oplus[l_1 : S_1, \ldots, l_n : S_n]} = \&[l_1 : \overline{S_1}, \ldots, l_n : \overline{S_n}]$$

$$\overline{\mathbf{t}} = \mathbf{t} \qquad \overline{\mu \mathbf{t}.S} = \mu \mathbf{t}.\overline{S} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Linearity

- Take $f = \lambda(x{:}S \multimap \diamond).(x \cdot s \mid x \cdot s')$
- Take $V = \lambda(y{:}S).(y!\langle 5\rangle \mid s''!\langle 2\rangle)$
- Then $f \cdot V$ is unsafe.
- $\longrightarrow (s!\langle 5\rangle \mid s'!\langle 5\rangle \mid s''!\langle 2\rangle \mid s''!\langle 2\rangle)$
- A linear function is consumed when applied.
- Similar restrictions for sessions, e.g. we should not have $s = s'$ above.
- No copying of linear functions and session endpoints (i.e. no contraction in typing).
- No forgetting (i.e. no weakening).

Partial Permutations

- $s!\langle 2\rangle.s!\langle \texttt{true}\rangle.s?(x).\mathbf{0} \mid \bar{s}?(y).\bar{s}?(z).\bar{s}!\langle y+2\rangle.\mathbf{0}$
- If we permute the outputs to get $s!\langle \texttt{true}\rangle.s!\langle 2\rangle.s?(x).\mathbf{0}$, then the above parallel composition causes a type-error.
- Also, one direction causes deadlock, losing progress: consider exchanging $s!\langle \texttt{true}\rangle$ and $s?(z)$ in $s!\langle \texttt{true}\rangle.s?(z).\mathbf{0}$, and $\bar{s}?(y).\bar{s}!\langle 2\rangle.\mathbf{0}$.
- $s?(z).s!\langle \texttt{true}\rangle.\mathbf{0} \mid \bar{s}?(y).\bar{s}!\langle 2\rangle.\mathbf{0} \quad \not\longrightarrow$
- Note that partial permutation is only applied to finite parts of the top-level actions *without* unfolding recursive types.
- The principle is simple: outputs / selections can be done in advance — not the other way around. Input dependency can cause deadlock.

Permutation Rules

$$\text{(OI)} \quad ![U].?[U'].S \ll ?[U'].![U].S$$

$$\text{(SI)} \quad \oplus[l_j : ?[U].S_j]_{j\in J} \ll ?[U].\oplus[l_j : S_j]_{j\in J}$$

$$\text{(OB)} \quad ![U].\&[l_j : S_j]_{j\in J} \ll \&[l_j : ![U].S_j]_{j\in J}$$

$$\text{(SB)} \quad \oplus[l_i : \&[l'_j : S_{ij}]_{j\in J}]_{i\in I} \ll \&[l'_j : \oplus[l_i : S_{ij}]_{i\in I}]_{j\in J}$$

$$\text{(Tr)} \ \frac{S_1 \ll S_2 \quad S_2 \ll S_3}{S_1 \ll S_3} \qquad \text{(CB)} \ \frac{\forall i \in I.\ S_i \ll S'_i}{\&[l_i : S_i]_{i\in I} \ll \&[l_i : S'_i]_{i\in I}}$$

$$\text{(CI)} \ \frac{S \ll S'}{?[U].S \ll ?[U].S'} \qquad \text{(CO)} \ ![U].S \ll ![U].S$$

$$\text{(CS)} \ \oplus[l_i : S_i]_{i\in I} \ll \oplus[l_i : S_i]_{i\in I} \qquad \text{(E)} \ \mathsf{end} \ll \mathsf{end} \qquad \text{(M)} \ \mu\mathbf{t}.\mathbf{S} \ll \mu\mathbf{t}.\mathbf{S}$$

n-time Unfolding

- Unfolding needs to be "deep":

$$\mathsf{unfold}^1(?[U].\mu\mathbf{t}.![U'].\mathbf{t}) \quad = \quad ?[U].![U'].\mu\mathbf{t}.![U'].\mathbf{t}$$

$$\mathsf{unfold}^2(?[U].\mu\mathbf{t}.?[U].\mu\mathbf{t}'.![U'].\mathbf{t}') \quad =$$

$$?[U].?[U].![U'].\mu\mathbf{t}'.![U'].\mathbf{t}'$$

- Allows output $![U']$ to appear at *top-level*
- Then the rules of $\ll$ can apply (OI) and we can obtain:

$$![U'].?[U].\mu\mathbf{t}.![U'].\mathbf{t}$$

Coinductive Subtyping I

$$!\,[U_1].\mu\mathbf{t}.!\,[U_1].?[U_2].\mathbf{t} \quad \leqslant_c \quad \mu\mathbf{t}.?[U_2].!\,[U_1].\mathbf{t}$$

- Subtyping is equi-recursive.
- Valid permutations are also allowed.
- The technique uses Type Simulations.

Coinductive Subtyping II

Variance Switch:

$$(H,H')^{\circledast} = (H,H') \qquad (S,S')^{\circledast} = (S',S) \qquad (\diamond,\diamond)^{\circledast} = (\diamond,\diamond)$$

$\Re$ is a Type Simulation if $(T_1,T_2) \in \Re$ implies:

1. If $T_1 = \diamond$, then $T_2 = \diamond$.
2. If $T_1 = \mathsf{unit}$, then $T_2 = \mathsf{unit}$.
3. If $T_1 = U_1 \to T_1'$, then $T_2 = U_2 \to T_2'$ or $T_2 = U_2 \multimap T_2'$ with $(U_2,U_1)^{\circledast} \in \Re$ and $(T_1',T_2')^{\circledast} \in \Re$.
4. If $T_1 = U_1 \multimap T_1'$, then $T_2 = U_2 \multimap T_2'$ with $(U_2,U_1)^{\circledast} \in \Re$ and $(T_1',T_2')^{\circledast} \in \Re$.
5. If $T_1 = \langle S_1 \rangle$, then $T_2 = \langle S_2 \rangle$ and $(S_1,S_2) \in \Re$ and $(S_2,S_1) \in \Re$.

Coinductive Subtyping III

6. If $T_1 = \text{end}$, then $\text{unfold}^n(T_2) = \text{end}$.
7. If $T_1 = ![U_1].S_1$, then $\text{unfold}^n(T_2) \gg ![U_2].S_2$, $(U_1, U_2)^{\circledast} \in \mathfrak{R}$ and $(S_1, S_2) \in \mathfrak{R}$.
8. If $T_1 = ?[U_1].S_1$, then $\text{unfold}^n(T_2) = ?[U_2].S_2$, $(U_2, U_1)^{\circledast} \in \mathfrak{R}$ and $(S_1, S_2) \in \mathfrak{R}$.
9. If $T_1 = \oplus[l_i : S_{1i}]_{i \in I}$, then $\text{unfold}^n(T_2) \gg \oplus[l_j : S_{2j}]_{j \in J}$, $I \subseteq J$ and $\forall i \in I.(S_{1i}, S_{2i}) \in \mathfrak{R}$.
10. If $T_1 = \&[l_i : S_{1i}]_{i \in I}$, then $\text{unfold}^n(T_2) = \&[l_j : S_{2j}]_{j \in J}$, $J \subseteq I$ and $\forall j \in J.(S_{1j}, S_{2j}) \in \mathfrak{R}$.
11. If $T_1 = \mu\mathbf{t}.S$, then $(\text{unfold}^1(T_1), T_2) \in \mathfrak{R}$.

Coinductive Subtyping IV

- **[Theorem] $\leqslant_c$ is a preorder.**
- *Transitivity* is the crucial property.
- Need to find the right relation (wrt $\Re_1$, $\Re_2$), and prove it is a simulation.
- We include the transitivity connections:

$$\Re = \Re_{12} \cdot \Re_{21} \cup \Re_{21} \cdot \Re_{12} \qquad \text{with} \qquad \Re_{ij} = \Re_i \cup \mathbf{trc}(\Re_j, \Re_i)$$

- Each $\Re_{ij}$ is a simulation (union of simulations)
- Then given $(T_1, T_2) \in \Re_1$ and $(T_2, T_3) \in \Re_2$, we construct $\Re$ and it contains (T_1, T_3). It is also a simulation.
- To prove $\Re$ is a simulation we take cases on an arbitrary pair from within the relation: we need to check that the rules of simulation still hold.

Coinductive Subtyping V

Unfolding and permutation preserve subtyping (Lemma):

$$\begin{array}{rcc} S_1 & \Re_1 & S_2 \\ & \vdots & \\ \mathsf{unfold}^n(S_1) \gg S_1' & \Re & S_2 \end{array}$$

The *Asynchrony Relation* $\mathcal{A}(S_1, S_2)$, given $S_1 \Re_1 S_2$, is the union:

- For all $n \in \mathbb{N}$
  - for each n, take all valid permutations using $\gg$ (finite)
    - for each, add simulation $\Re$ which exists by Lemma above.

The *Transitivity Connection* $\mathbf{trc}(\Re_1, \Re_2)$ is also a union:

- Whenever $S_1 \; \Re_1 \; S_2 \; \Re_2 \; S_3$, we have $\mathcal{A}(S_2, S_3) \subseteq \mathbf{trc}(\Re_1, \Re_2)$.

Typing System

Typing Judgement:

$$\Gamma; \Sigma; \mathcal{L} \vdash P : T$$

Typing Environments:

$$\Gamma ::= \emptyset \mid \Gamma, u : H \qquad \Sigma ::= \emptyset \mid \Sigma, k : S \qquad \mathcal{L} \text{ is a set.}$$

Some Rules:

(Shared)

$$\frac{H \neq U \multimap T}{\Gamma, u : H; \emptyset; \emptyset \vdash u : H}$$

(Session)

$$\frac{}{\Gamma; k : S; \emptyset \vdash k : S}$$

(LVar)

$$\frac{}{\Gamma, x : U \multimap T; \emptyset; \{x\} \vdash x : U \multimap T}$$

(Base)

$$\frac{}{\Gamma; \emptyset; \emptyset \vdash () : \mathtt{unit}}$$

Typing System (cont.)

Some Rules (cont.):

$$\text{(Abs)}\quad \frac{\Gamma, x:H;\Sigma;\mathcal{L} \vdash P : T \qquad \text{if } H = U \multimap T' \text{ then } x \in \mathcal{L}}{\Gamma;\Sigma;\mathcal{L} \setminus x \vdash \lambda(x:H).P : H \to T}$$

$$\text{(Abs}_S\text{)}\quad \frac{\Gamma;\Sigma, x:S;\mathcal{L} \vdash P : T}{\Gamma;\Sigma;\mathcal{L} \vdash \lambda(x:S).P : S \to T}$$

Typing System (cont.)

Some Rules (cont.):

$$\frac{\text{(App) } \Gamma;\Sigma_1;\mathcal{L}_1 \vdash P : U \multimap T \quad \Gamma;\Sigma_2;\mathcal{L}_2 \vdash Q : U \quad \text{if } U = U' \to T' \text{ then } \Sigma_2 = \mathcal{L}_2 = \emptyset}{\Gamma;\Sigma_1,\Sigma_2;\mathcal{L}_1,\mathcal{L}_2 \vdash P\,Q : T}$$

$$\frac{\text{(Sub) } \Gamma;\Sigma;\mathcal{L} \vdash P : H \quad \Sigma \leqslant_c \Sigma' \quad H \leqslant_c H'}{\Gamma;\Sigma';\mathcal{L} \vdash P : H'}$$

Typing System (cont.)

Some Rules (cont.):

(Req)

$$\frac{\Gamma;\emptyset;\emptyset \vdash u : \langle S \rangle \quad \Gamma;\Sigma, x : S; \mathcal{L} \vdash P : \diamond}{\Gamma;\Sigma;\mathcal{L} \vdash \overline{u}(x).P : \diamond}$$

(Rec)

$$\frac{\Gamma, x : H; \Sigma, k : S; \mathcal{L} \vdash P : \diamond \quad (\star)}{\Gamma;\Sigma, k : ?[H].S; \mathcal{L} \setminus x \vdash k?(x).P : \diamond}$$

Session Remainder

$$\boxed{S - \vec{\tau} = S'}$$

- S is the type of an endpoint s in a program fragment.
- $\vec{\tau}$ is a sequence of types corresponding to the values $\vec{h}$ in the buffer $s : \vec{h}$.
- S' is the result of subtracting the buffer types from the original session type — it is the *session remainder*.

Session Remainder Rules

(Empty)

$$\frac{}{S - \varepsilon = S}$$

(Get)

$$\frac{S - \vec{\tau} = S'}{?[U].S - U\vec{\tau} = S'}$$

(Put)

$$\frac{S - \vec{\tau} = S'}{![U].S - \vec{\tau} = ![U].S'}$$

(Branch)

$$\frac{S_k - \vec{\tau} = S' \qquad k \in I}{\&[l_i : S_i]_{i\in I} - l_k\vec{\tau} = S'}$$

(Select)

$$\frac{S_i - \vec{\tau} = S'_i \qquad \forall i \in I}{\oplus[l_i : S_i]_{i\in I} - \vec{\tau} = \oplus[l_i : S'_i]_{i\in I}}$$

Some examples:

$$?[U].\mathsf{end} - U = \mathsf{end} \qquad ?[U].![U'].\mathsf{end} - U = ![U'].\mathsf{end}$$

$$![U'].?[U].\mathsf{end} - U = ![U'].\mathsf{end}$$

Runtime Typing I

Extended Session Environment:

$$\Delta \ ::= \ \Sigma \mid \Delta, s : \vec{\tau} \mid \Delta, s : (S, \vec{\tau})$$

Composition:

$$\Delta_1 \odot \Delta_2 \quad = \quad \{s : (\Delta_1(s), \Delta_2(s)) \mid s \in \mathsf{dom}(\Delta_1) \cap \mathsf{dom}(\Delta_2)\}$$

$$\cup \Delta_1 \setminus \mathsf{dom}(\Delta_2) \cup \Delta_2 \setminus \mathsf{dom}(\Delta_1)$$

Example:

$$s?(x).\mathbf{0} \ \mid \ s : V$$

$$\{s : ?[U].\mathsf{end}\} \odot \{s : U\} \quad = \quad \{s : (?[U].\mathsf{end}, U)\}$$

Runtime Typing II

Some Rules:

(Par)

$$\frac{\Gamma; \Delta_{1,2}; \mathcal{L}_{1,2} \vdash P_{1,2} : \diamond}{\Gamma; \Delta_1 \odot \Delta_2; \mathcal{L}_1, \mathcal{L}_2 \vdash P_1 \mid P_2 : \diamond}$$

(Queue) if $\tau_i = U \rightarrow T$ then $\Sigma_i = \emptyset$

$$\frac{\Gamma; \Sigma_i; \emptyset \vdash h_i : \tau_i \quad i \in 1..n \quad \Sigma_0 = \{\vec{s} : \overrightarrow{\text{end}}\}}{\Gamma; (\Sigma_0, .., \Sigma_n) \odot s : \tau_1..\tau_n; \emptyset \vdash s : h_1..h_n : \diamond}$$

(New$_s$) $S_1 - \vec{\tau}_1 = S_1' \quad S_2 - \vec{\tau}_2 = S_2'$

$$\frac{\Gamma; \Delta, s : (S_1, \vec{\tau}_1), \bar{s} : (S_2, \vec{\tau}_2); \emptyset \vdash P : \diamond \quad S_1' \leqslant_c \overline{S_2'}}{\Gamma; \Delta; \emptyset \vdash (\nu\, s)P : \diamond}$$

Some Definitions

$\mathsf{balanced}(\Delta)$ if whenever $s:(S_1,\vec{\tau}_1), \bar{s}:(S_2,\vec{\tau}_2) \in \Delta$ with $S_1 - \vec{\tau}_1 = S_1'$ and $S_2 - \vec{\tau}_2 = S_2'$, then $\mathbf{S_1'} \leqslant_\mathbf{c} \overline{\mathbf{S_2'}}$.

Δ Ordering

$$s : ?[U].S \odot s : U\vec{\tau} \ \sqsubseteq_s \ s : S \odot s : \vec{\tau}$$

$$s : ![U].S \odot \bar{s} : \vec{\tau} \ \sqsubseteq_s \ s : S \odot \bar{s} : \vec{\tau}U$$

$$\vdots$$

$$s : \mu\mathbf{t}.S \odot s' : \vec{\tau} \ \sqsubseteq_s \ s : S' \odot s' : \vec{\tau}'$$

$$\text{if} \quad s : S[\mu\mathbf{t}.S/\mathbf{t}] \odot s' : \vec{\tau} \ \sqsubseteq_s \ s : S' \odot s' : \vec{\tau}'$$

$$\Delta \odot \Delta_1 \sqsubseteq_s \Delta \odot \Delta_2 \quad \text{if} \quad \Delta_1 \sqsubseteq_s \Delta_2 \text{ and } \Delta \odot \Delta_1 \text{ defined}$$

Properties

Type Soundness

1. Suppose $\Gamma;\Delta;\mathcal{L} \vdash P : \diamond$. Then $P \equiv P'$ implies $\Gamma;\Delta;\mathcal{L} \vdash P' : \diamond$.
2. Suppose $\Gamma;\Delta;\emptyset \vdash P : T$ with $\mathsf{balanced}(\Delta)$. Then $P \longrightarrow P'$ implies $\Gamma;\Delta';\emptyset \vdash P' : T$ and either $\Delta = \Delta'$ or $\Delta \sqsubseteq_s \Delta'$.

Type Safety

If $\Gamma;\Delta;\mathcal{L} \vdash P : \diamond$ with $\mathsf{balanced}(\Delta)$, then P never reduces into an error.

Error $P \equiv (\nu\,\vec{a})(\nu\,\vec{s})(Q \mid R)$ where Q is e.g.:

- $s!\langle V\rangle.R_1 \mid s!\langle V'\rangle.R_2$
- $\ldots$

Types for Scenario

$$S_{\texttt{Agent}} = ![\texttt{int}].\&[\mathsf{move} : ?[\mathsf{unit} \multimap \diamond].S'_{\texttt{Agent}}\,,\ \mathsf{local} : S'_{\texttt{Agent}}]$$

with $\quad S'_{\texttt{Agent}} = ?[\texttt{string}].?[\texttt{string}].![\texttt{double}].?[\texttt{int}].\mathsf{end}$

$$S_{\texttt{client}} = \overline{S_{\texttt{Agent}}}$$

$$S_{\texttt{MClient}} = \oplus[\mathsf{move} : ![\mathsf{unit} \multimap \diamond].![\texttt{string}].![\texttt{string}].\\ ![\texttt{int}].?[\texttt{int}].?[\texttt{double}].\mathsf{end}]$$

$S_{\texttt{MClient}} \leqslant_c \overline{S_{\texttt{Agent}}}$ and $S_{\texttt{MClient}} \leqslant_c S_{\texttt{Client}}$

Algorithmic Subtyping

- Decidability of $S \ll S'$.
- Rewriting rule $S \overset{!}{\mapsto} S'$ and $S \overset{\oplus}{\mapsto} S'$ – moves the action to the head.
- $\Sigma \vdash T \leqslant T'$
- $\mathcal{T}[S_h]^{h \in H}$ is a h-hole context.
- $T \bowtie T'$ means that T and T' have the same session constructors under matching recursions; and labels in each type are distinct.

$$\text{(Out)}\ \frac{\Sigma \vdash ![U_1].S_1 \leqslant ![U_2].\mathcal{T}[S_{2h}]^{h \in H} \qquad \mathcal{T}[![U_2].S_{2h}]^{h \in H} \overset{!}{\mapsto} ![U_2].\mathcal{T}[S_{2h}]^{h \in H} \qquad S_1 \bowtie \mathcal{T}[S_{2h}]^{h \in H}}{\Sigma \vdash ![U_1].S_1 \leqslant \mathcal{T}[![U_2].S_{2h}]^{h \in H}}$$

Algorithmic Subtyping II

Theorem For all closed types T and T' with $T \bowtie T'$, $T \leqslant_c T'$ if and only if $\emptyset \vdash T \leqslant T'$.

- Restriction $T \bowtie T'$ is necessary as assumptions can grow indefinitely.
- The problem is that deep unfolding may need to be used repeatedly creating new types larger than before.
- These new types are not in the assumptions.
- But, we can relax the restriction to allow the constructors to have the same rate wrt inputs and outputs, between subtype and supertype.
- Proof outline for basic restriction.

Conclusions

- Session actions permutation statically checked as sybtyping.
- Co-inductive simulation.
- Integration with Linear Higher-order Code.
- Runtime typing & Safety properties.
- Algorithmic subtyping.

Related Work

- Typing and Subtyping for Mobile Processes Benjamin Pierce and Davide Sangiorgi.
- Subtyping for Session Types in the Pi Calculus S. Gay and M. Hole.
- Linear λ-calculus. Ch. 1 of “Advanced Topics in Types and Programming Languages” David Walker, ed. B.C. Pierce.
- Linear Type Theory for Asynchronous Session Types S. Gay and V. T. Vasconcelos.

The End

?

Appendix

Encoding Replicated Processes

Definition:

$$!a(x).P \overset{\text{def}}{=} (\mu y.\lambda z.z(x).(P \mid y\,z))\,a \qquad \text{taking} \quad y, z \notin \mathsf{fv}(P)$$

Reduction:

$$!a(x).P \mid \overline{a}(z).Q$$

$$\longrightarrow \quad a(x).(P \mid !a(x).P) \;\mid\; \overline{a}(z).Q \qquad \text{(rec)}$$

$$\longrightarrow \quad (\nu\, s)\,(P\{^{s}/_{x}\} \mid !a(x).P \mid Q\{^{\overline{s}}/_{z}\} \mid s:\varepsilon \mid \overline{s}:\varepsilon) \qquad \text{(conn)}$$

$$\equiv \quad (\nu\, s)\,(P\{^{s}/_{x}\} \mid Q\{^{\overline{s}}/_{z}\} \mid s:\varepsilon \mid \overline{s}:\varepsilon) \;\mid\; !a(x).P$$

n-time Unfolding Definition

$$\text{unfold}^0(S) = S \text{ for all } S$$

$$\text{unfold}^{1+n}(S) = \text{unfold}^1(\text{unfold}^n(S))$$

$$\text{unfold}^1(![U].S) = ![U].\text{unfold}^1(S)$$

$$\text{unfold}^1(\oplus[l_i : S_i]_{i\in I}) = \oplus[l_i : \text{unfold}^1(S_i)]_{i\in I}$$

$$\text{unfold}^1(?[U].S) = ?[U].\text{unfold}^1(S)$$

$$\text{unfold}^1(\&[l_i : S_i]_{i\in I}) = \&[l_i : \text{unfold}^1(S_i)]_{i\in I}$$

$$\text{unfold}^1(\mathbf{t}) = \mathbf{t}$$

$$\text{unfold}^1(\mu\mathbf{t}.S) = S[\mu\mathbf{t}.S/\mathbf{t}]$$

$$\text{unfold}^1(\text{end}) = \text{end}$$

Coinductive Subtyping VI

- [Lemma] If $S_1 \leqslant_c S_2$ and $S_1' \ll \mathsf{unfold}^n(S_1)$ then $S_1' \leqslant_c S_2$.
- [Definition] When $S_1 \, \mathfrak{R} \, S_2$ for type simulation $\mathfrak{R}$, we define the asynchrony relation of S_1 and S_2 as:

$$\mathcal{A}(S_1, S_2) =$$

$$\bigcup_{n \in \mathbb{N}} \{(S_1', S_2') \mid \mathsf{unfold}^n(S_1) \gg S \ \wedge \ S \ \mathfrak{R}' \, S_2 \ \wedge \ \mathfrak{R}' \subseteq \leqslant_c \\ \wedge \ (S_1', S_2') \in \mathfrak{R}'\}$$

- [Definition] Transitivity Connection $\mathbf{trc}(\mathfrak{R}_1, \mathfrak{R}_2)$ is the smallest relation such that whenever $S_1 \, \mathfrak{R}_1 \, S_2$ and $S_2 \, \mathfrak{R}_2 \, S_3$, we have $\mathcal{A}(S_2, S_3) \subseteq \mathbf{trc}(\mathfrak{R}_1, \mathfrak{R}_2)$.