On the Expressiveness of Higher-Order Sessions (Draft of October 1, 2014)

No Author Given

Imperial College London

1 The Full Higher-Order Session π-Calculus

We present the syntax and operational semantics for the *Full Higher-Order Session π-calculus* or $\mathsf{HO}\pi$. $\mathsf{HO}\pi$ is a simpler version of the higher-order calculus developed in [1], that name and process abstraction application, as well as implicit application of abstractions. We believe that the higher-order semantics of the calculus is small and yet expressive enough to capture in a simple way the principles of session types.

1.1 $\mathsf{HO}\pi$ Syntax

The syntax for $\mathsf{HO}\pi$ processes is given in Figure 1. We assume a set of names S that range over $s, s_1, \ldots$ and a dual set of $\overline{S}$ that ranges over $\overline{s}, \overline{s_1}, \ldots$. Intuitively, s and $\overline{s}$ are dual *endpoints*. Both sets S and $\overline{S}$ constitute the set of names N. Name variables $x, y, z, \ldots$ are taken from set Var and process variable $X, Y, Z, \ldots$ from set PVar. Set RVar is the set of recursive variables that range over $\mathcal{X}, \mathcal{Y}, \ldots$. Abstractions, written $(x)P$, are processes P with a bound parameter x; the set of all abstractions is denoted Abs. We denote either names or name variables with $k, k_1, \ldots$. We write $V, V', \ldots$ to denote name variables, process variables, recursive variables, or abstractions. Intuitively, these are transmittable *values*. Note that set N includes session and shared names. We sometimes denote shared names with $a, b, c, \ldots$, although $a, b, c, \cdots \in S$.

Processes include the standard π-calculus prefixes for sending and receiving names. Prefix $k!\langle k' \rangle; P$ denotes the sending of name k over channel k and then continuing as P whereas prefix $k?(x); P$ denotes the reception of a value on channel k over variable x and then continue as P. Recursion is expressed on the primitive recursor $\mu\mathcal{X}.P$ that binds the recursive variable/process $\mathcal{X}$ in the structure of P.

Higher-order syntax is composed by the sending prefix $k!\langle (x)Q \rangle; P$ that denotes the sending of abstraction $(x)Q$ over channel k and then continuing as P. On the receiving side prefix $k?(X); P$ denotes the reception of an abstraction on channel k and over the process variable X. Process $X\langle k \rangle$ is the application process which is used to bind channel k on the process substituting process variable X.

We assume the standard session syntax for selection and branching. Process $k \triangleleft l; P$ selects label l on channel k and then behaves as P. Given $i \in I$, process $k \triangleright \{l_i : P_i\}_{i \in I}$ offers a choice on labels l_i with corresponding continuation P_i. The calculus is completed with the standard constructs for inaction $\mathbf{0}$, parallel composition $P_1 \mid P_2$, and name restriction $(\nu\ s)P$, which simultaneously binds endpoints s and $\overline{s}$ in P.

$$
\begin{array}{lcl}
P,Q & ::= & k!\langle k'\rangle;P \mid k?(x);P \mid X \mid \mu X.P \\
 & \mid & k!\langle (x)Q\rangle;P \mid k?(X);P \mid X\langle k\rangle \\
 & \mid & k \triangleleft l;P \mid k \triangleright \{l_i : P_i\}_{i\in I} \mid P_1 \mid P_2 \mid (\nu\ s)P \mid \mathbf{0} \\
\text{Names} & : & S = \{s, s_1, \ldots\} \qquad \overline{S} = \{\overline{s} \mid s \in S\} \qquad N = S \cup \overline{S} \\
\text{Variables} & : & \mathsf{Var} = \{x, y, z, \ldots\} \qquad \mathsf{PVar} = \{X, Y, Z, \ldots\} \qquad \mathsf{RVar} = \{\mathcal{X}, \mathcal{Y}, \ldots\} \\
\text{Abstractions} & : & \mathsf{Abs} = \{(x)P \mid P \text{ is a process}\} \\
 & & k \in N \cup \mathsf{Var} \qquad V \in N \cup \mathsf{Var} \cup \mathsf{PVar} \cup \mathsf{RVar} \cup \mathsf{Abs}
\end{array}
$$

Fig. 1. Syntax for HOπ

A well-formed process relies on assumptions for guarded recursive processes. A process is called a *program* if it contains no free recursion variables nor free name/process variables.

Two significant sub-calculi of HOπ will form the basis of our study. The first sub-calculus involves name-passing but no process-passing constructs; it is defined by lines 1 and 3 of the syntax in Figure 1. This is essentially a standard session π-calculus or session-π as defined in the bibliography []. The second sub-calculus features abstraction passing and application but no name-passing; it is defined by lines 2 and 3 of the same syntax. This language calculus is called the higher-order session calculus or HO.

1.2 Operational Semantics

Structural Congruence We define structural congruence as the least congruence that satisfies the commutative monoid ($\mid$,$\mathbf{0}$) and the rules:

$(\nu\ s)\mathbf{0} \equiv \mathbf{0} \quad s \notin \mathtt{fn}(P_1)$ implies $P_1 \mid (\nu\ s)P_2 \equiv (\nu\ s)(P_1 \mid P_2) \quad \mu X.P \equiv P\{\mu X.P/X\}$

We define the reduction semantics in Figure 2. Figure 2 first describes the process variable substitution through the semantics of name substitution. Substitution of application process $X\langle k\rangle$ over abstraction $(x)Q$ substitutes free variable x in Q with k and replaces X with the resulting process. There is no effect on variable substitution for the inactive process. In all the other cases process variable substitution is homorphic on the structure of the process.

There are three communication rules for the HOπ. The name passing rule, [NPass] where a send prefix on channel s sends channel s' to be received by parallel receive prefixed process on the dual endpoint $\overline{s}$. After the reduction the continuation of the latter process substitutes s' on the receive prefix object x. The second communication axiom, [APass], describes abstraction passing. An abstraction is sent on a send prefixed process over channel s and it is being received by a parallel $\overline{s}$-received prefixed process. The latter process follows a process variable substitution. The third axiom is the standard select/branch action where a process select label $l_k, k \in I$ on channel s over a set of labels $\{l_i\}_{i\in I}$ that are offered from a parallel process on the dual session endpoint $\overline{s}$. The continuation of the latter process is the corresponding process P_k. The last three rules are congruence rules that preserve reduction on the parallel, rule [Par] and name restriction, rule [Ses], operators as well as reduction is closed under structural congruence, rule [Cong].

$$s!\langle s'\rangle;P \mid \overline{s}?(x);Q \longrightarrow P \mid Q\{s'/x\}\ [\text{NPass}]$$
$$s!\langle (x)P_1\rangle;P \mid \overline{s}?(X);Q \longrightarrow P \mid Q\{(x)P_1/X\}\ [\text{APass}]$$
$$\frac{j \in I}{s \triangleleft l_j;Q \mid \overline{s} \triangleright \{l_i : P_i\}_{i\in I} \longrightarrow Q \mid P_j}\ [\text{Sel}] \qquad \frac{P \longrightarrow P'}{(\nu\ s)P \longrightarrow (\nu\ s)P'}\ [\text{Sess}]$$
$$\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}\ [\text{Par}] \qquad \frac{P \equiv \longrightarrow \equiv P'}{P \longrightarrow P'}\ [\text{Cong}]$$

$$X\langle k\rangle\{(x)Q/X\} = Q\{k/x\} \qquad \mathbf{0}\{(x)Q/X\} = \mathbf{0}$$
$$s?(Y);P\{(x)Q/X\} = s?(Y);(P\{(x)Q/X\})$$
$$(s!\langle (y)P_1\rangle;P_2)\{(x)Q/X\} = s!\langle (y)P_1\{(x)Q/X\}\rangle;(P_2\{(x)Q/X\})$$
$$s \triangleleft l;P\{(x)Q/X\} = s \triangleleft l;(P\{(x)Q/X\})$$
$$s \triangleright \{l_i : P_i\}_{i\in I}\{(x)Q/X\} = s \triangleright \{l_i : P_i\{(x)Q/X\}\}_{i\in I}$$
$$(P_1 \mid P_2)\{(x)Q/X\} = P_1\{(x)Q/X\} \mid P_2\{(x)Q/X\} \qquad (\nu\ s)P\{(x)Q/X\} = (\nu\ s)(P\{(x)Q/X\})$$

Fig. 2. Upper part: Reduction semantics for HOπ. Lower part: Process variable substitution.

2 Types for HOπ

We define a session type system for HOπ, which is based on the type system developed by Mostrous and Yoshida in [1].

2.1 Session Types

We consider a minimal type structure, a fragment of that defined in [1]. The only (but fundamental) differences are in the types for values: we focus on having $S \to \diamond$ and $S \multimap \diamond$, whereas the structure in [1] supports general functions $U \to T$ and $U \multimap T$.

$$\begin{array}{llllll} \textit{Values} & U ::= S \mid S \multimap \diamond \mid S \to \diamond \mid \langle S \rangle & \textit{Terms} & T ::= U \mid \diamond \\ \textit{Sessions} & S ::= !\langle U\rangle;S \mid ?(U);S \mid \oplus\{l_i : S_i\}_{i\in I} \mid \&\{l_i : S_i\}_{i\in I} \mid \mu \mathsf{t}.S \mid \mathsf{t} \mid \mathsf{end} & & \end{array}$$

There are four different value types U; session value S, linear higher order value $S \multimap \diamond$, shared higher order value $S \to \diamond$; shared channel $\langle S \rangle$. Terms can either have a value type U or a process type $\diamond$.

Session types follow the standard binary session types syntax []. Session send prefix $!\langle U\rangle;S$ denotes a session type that sends a value of type U and continues as S. Dually receive prefix $?(U);S$ denotes a session type that receives a value of type U and continues as S. Set ST is the space of all session types.

Definition 2.1 (Duality). *Let function $F(R) : \mathsf{ST} \longrightarrow \mathsf{ST}$ to be defined as:*

$$\begin{aligned} F(R) = \{&(\mathsf{end},\mathsf{end}),(\mathsf{t},\mathsf{t})\} \\ &\cup \{(!\langle U\rangle;;S_1,?(U);;S_2),(?(U);;S_1,!\langle U\rangle;;S_2) \mid S_1\ R\ S_2\} \\ &\cup \{(\oplus\{l_i : S_i\}_{i\in I},\&\{l_j : S'_j\}_{j\in J}) \mid I \subseteq J, S_i\ R\ S'_j\} \\ &\cup \{(\&\{l_i : S_i\}_{i\in I},\oplus\{l_j : S'_j\}_{j\in J}) \mid J \subseteq I, S_j\ R\ S'_i\} \\ &\cup \{(\mu \mathsf{t}.S_1,\mu \mathsf{t}.S_2) \cup (S_1\{\mu \mathsf{t}.S/\mathsf{t}\},S_2),(S_1,S_2\{\mu \mathsf{t}.S/\mathsf{t}\}) \mid S_1\ R\ S_2)\} \end{aligned}$$

Standard arguments ensure that F is monotone, thus the greatest fix point of F exists and let duality defined as $\mathsf{dual} = \nu X.F(X)$.

Following our decision of focusing on functions $S \to \diamond$ and $S \multimap \diamond$, our environments are also simpler than those in [1]:

$$\begin{array}{lllll} \text{Shared} & \Gamma & ::= & \emptyset \mid \Gamma \cdot X : S \to \diamond \mid \Gamma \cdot k : \langle S \rangle \mid \Gamma \cdot \mathcal{X} : \Sigma \\ \text{Linear} & \Lambda & ::= & \emptyset \mid \Lambda \cdot X : S \multimap \diamond \\ \text{Session} & \Sigma & ::= & \emptyset \mid \Sigma \cdot k : S \end{array}$$

With these environments the shape of judgments is exactly the same as in Mostrous and Yoshida's system:

$$\Gamma; \Lambda; \Sigma \vdash P \triangleright T$$

As expected, weakening, contraction, and exchange principles apply to Γ; environments Λ and Σ behave linearly, and are only subject to exchange. We require that the domains of Γ, Λ and Σ are pairwise distinct.

The typing rules for our system are in Fig. 3.

2.2 Type Soundness

We state results for type safety: we report instances of more general statements already proved by Mostrous and Yoshida in the asynchronous case.

Lemma 2.1 (Substitution Lemma - Lemma C.10 in M&Y).

1. *Suppose* $\Gamma \cdot X : S \to \diamond; \Lambda; \Sigma \vdash P \triangleright T$ *and* $\Gamma; \emptyset; \emptyset \vdash V \triangleright S \to \diamond$. *Then* $\Gamma; \Lambda; \Sigma \vdash P\{V/x\} \triangleright T$.
2. *Assume* $\Gamma; \Lambda_1 \cdot X : S \multimap \diamond; \Sigma_1 \vdash P \triangleright T$ *and* $\Gamma; \Lambda_2; \Sigma_2 \vdash V \triangleright S \multimap \diamond$ *with* Λ_1, Λ_2 *and* Σ_1, Σ_2 *defined. Then* $\Gamma; \Lambda_1 \cup \Lambda_2; \Sigma_1 \cup \Sigma_2 \vdash P\{V/X\} \triangleright T$.
3. *Suppose* $\Gamma; \Lambda; \Sigma \cdot x : S \vdash P \triangleright T$ *and* $k \notin \mathtt{dom}(\Gamma, \Lambda, \Sigma)$. *Then* $\Gamma; \Lambda; \Sigma \cdot k : S \vdash P\{k/x\} \triangleright T$.
4. *Suppose* $\Gamma \cdot x : \langle S \rangle; \Lambda; \Sigma \vdash P \triangleright T$ *and* $k \notin \mathtt{dom}(\Gamma, \Lambda, \Sigma)$. *Then* $\Gamma \cdot k : \langle S \rangle; \Lambda; \Sigma \vdash P\{k/x\} \triangleright T$.

Proof. In all four parts, we proceed by induction on the typing for P, with a case analysis on the last applied rule. Interesting parts are (1) and (2), where the definition of substitution in Fig. 2 and strengthening are required. (TBC) □

Definition 2.2 (Well-typed Session Environment). *Let* Σ *a session environment. We say that* Σ *is* well-typed *if whenever* $s : S_1, \overline{s} : S_2 \in \Sigma$ *then* $S_1 \ \mathsf{dual} \ S_2$.

Definition 2.3 (Session Environment Reduction). *We define relation* $\longrightarrow$ *on session environments as:*

- $\Sigma \cdot s : !\langle U \rangle; S_1 \cdot \overline{s} : ?(U); S_2 \longrightarrow \Sigma \cdot s : S_1 \cdot \overline{s} : S_2$
- $\Sigma \cdot s : \oplus\{l_i : S_i\}_{i \in I} \cdot \overline{s} : \&\{l_i : S'_i\}_{i \in I} \longrightarrow \Sigma \cdot s : S_k \cdot \overline{s} : S'_k, \quad k \in I.$

We now state the instance of type soundness that we can derived from the Mostrous and Yoshida system. It is worth noticing that M&Y have a slightly richer definition of structural congruence. Also, their statement for subject reduction relies on an ordering on typings associated to queues and other runtime elements (such extended typings are denoted Δ by M&Y). Since we are synchronous we can omit such an ordering.

Theorem 2.1 (Type Soundness - Theorem 7.3 in M&Y).

1. *(Subject Congruence) Suppose* $\Gamma; \Lambda; \Sigma \vdash P \triangleright \diamond$. *Then* $P \equiv P'$ *implies* $\Gamma; \Lambda; \Sigma \vdash P' \triangleright \diamond$.
2. *(Subject Reduction) Suppose* $\Gamma; \emptyset; \Sigma \vdash P \triangleright T$ *with well-typed* Σ. *Then* $P \longrightarrow P'$ *implies* $\Gamma; \emptyset; \Sigma_2 \vdash P' \triangleright T$ *and* $\Sigma_1 \longrightarrow \Sigma_2$ *or* $\Sigma_1 = \Sigma_2$.

$$[\text{Shared}]\ \Gamma \cdot k : \langle S \rangle; \emptyset; \emptyset \vdash k \triangleright \langle S \rangle \qquad [\text{LVar}]\ \Gamma; \{X : S \multimap \diamond\}; \emptyset \vdash X \triangleright S \multimap \diamond \qquad [\text{Session}]\ \Gamma; \emptyset; \{k : S\} \vdash k \triangleright S$$

$$[\text{Prom}]\ \frac{\Gamma; \emptyset; \emptyset \vdash P \triangleright S \multimap \diamond}{\Gamma; \emptyset; \emptyset \vdash P \triangleright S \to \diamond} \qquad [\text{Derelic}]\ \frac{\Gamma; \Lambda \cdot X{:}S \multimap \diamond; \Sigma \vdash P \triangleright T}{\Gamma \cdot X : S \to \diamond; \Lambda; \Sigma \vdash P \triangleright T}$$

$$[\text{Abs}]\ \frac{\Gamma; \Lambda; \Sigma \cdot x : S \vdash P \triangleright \diamond}{\Gamma; \Lambda; \Sigma \vdash (x)P \triangleright S \multimap \diamond}$$

$$[\text{App}]\ \frac{(U = S \multimap \diamond) \vee (U = S \to \diamond) \quad \Gamma; \Lambda_1; \Sigma_1 \vdash X \triangleright U \quad \Gamma; \Lambda_2; \Sigma_2 \vdash k \triangleright S}{\Gamma; \Lambda_1 \cup \Lambda_2; \Sigma_1 \cup \Sigma_2 \vdash X\langle k \rangle \triangleright \diamond}$$

$$[\text{Send}]\ \frac{\Gamma; \Lambda_1; \Sigma_1 \vdash P \triangleright \diamond \quad \Gamma; \Lambda_2; \Sigma_2 \vdash V \triangleright U \quad (k : S \in \Sigma_1 \cup \Sigma_2)}{\Gamma; \Lambda_1 \cup \Lambda_2; (\Sigma_1 \cup \Sigma_2) \backslash \{k : S\} \cdot k : !\langle U \rangle; S \vdash k!\langle V \rangle; P \triangleright \diamond}$$

$$[\text{Conn}]\ \frac{\Gamma; \Lambda; \Sigma \cdot x : S \vdash P \triangleright \diamond \quad \Gamma; \emptyset; \emptyset \vdash k \triangleright \langle S \rangle}{\Gamma; \Lambda; \Sigma \vdash k?(x); P \triangleright \diamond}$$

$$[\text{ConnDual}]\ \frac{\Gamma; \Lambda; \Sigma \cdot s : S_1 \vdash P \triangleright \diamond \quad \Gamma; \emptyset; \emptyset \vdash k \triangleright \langle S_2 \rangle \quad S_1 \text{ dual } S_2}{\Gamma; \Lambda; \Sigma \vdash k!\langle s \rangle; P \triangleright \diamond}$$

$$[\text{NewSh}]\ \frac{\Gamma \cdot s : \langle S \rangle; \Lambda; \Sigma \vdash P \triangleright \diamond}{\Gamma; \Lambda; \Sigma \vdash (\nu\ s)P \triangleright \diamond} \qquad [\text{NewSes}]\ \frac{\Gamma; \Lambda; \Sigma \cdot s : S_1 \cdot \overline{s} : S_2 \vdash P \triangleright \diamond \quad S_1 \text{ dual } S_2}{\Gamma; \Lambda; \Sigma \vdash (\nu\ s)P \triangleright \diamond}$$

$$[\text{RecvS}]\ \frac{\Gamma; \Lambda; \Sigma \cdot k : S_1 \cdot x : S_2 \vdash P \triangleright \diamond}{\Gamma; \Lambda; \Sigma, k : ?(S_2); S_1 \vdash k?(x); P \triangleright \diamond} \qquad [\text{RecvL}]\ \frac{\Gamma; \Lambda \cdot X : S \multimap \diamond; \Sigma \cdot k : S_1 \vdash P \triangleright \diamond}{\Gamma; \Lambda; \Sigma \cdot k : ?(S \multimap \diamond); S_1 \vdash k?(X); P \triangleright \diamond}$$

$$[\text{RecvSh}]\ \frac{\Gamma \cdot X : S \to \diamond; \Lambda; \Sigma \cdot k : S_1 \vdash P \triangleright \diamond}{\Gamma; \Lambda; \Sigma \cdot k : ?(S \to \diamond); S_1 \vdash k?(X); P \triangleright \diamond} \qquad [\text{RecvShN}]\ \frac{\Gamma \cdot x : \langle S \rangle; \Lambda; \Sigma \cdot k : S_1 \vdash P \triangleright \diamond}{\Gamma; \Lambda; \Sigma \cdot k : ?(\langle S \rangle); S_1 \vdash k?(x); P \triangleright \diamond}$$

$$[\text{Nil}]\ \Gamma; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond$$

$$[\text{Par}]\ \frac{\Gamma; \Lambda_1; \Sigma_1 \vdash P_1 \triangleright \diamond \quad \Gamma; \Lambda_2; \Sigma_2 \vdash P_2 \triangleright \diamond}{\Gamma; \Lambda_1 \cup \Lambda_2; \Sigma_1 \cup \Sigma_2 \vdash P_1 \mid P_2 \triangleright \diamond} \qquad [\text{Close}]\ \frac{\Gamma; \Lambda; \Sigma \vdash P \triangleright T \quad k \notin \text{dom}(\Gamma, \Lambda, \Sigma)}{\Gamma; \Lambda; \Sigma \cdot k : \text{end} \vdash P \triangleright T}$$

$$[\text{Bra}]\ \frac{\forall i \in I \quad \Gamma; \Lambda; \Sigma \cdot k : S_i \vdash P_i \triangleright \diamond}{\Gamma; \Lambda; \Sigma \cdot k : \&\{l_i : S_i\}_{i \in I} \vdash k \triangleright \{l_i : P_i\}_{i \in I} \triangleright \diamond} \qquad [\text{Sel}]\ \frac{\Gamma; \Lambda; \Sigma \cdot k : S_j \vdash P \triangleright \diamond \quad j \in I}{\Gamma; \Lambda; \Sigma \cdot k : \oplus\{l_i : S_i\}_{i \in I} \vdash s \triangleleft l_j; P \triangleright T}$$

$$[\text{Var}]\ \frac{}{\Gamma \cdot X : \Sigma; \emptyset; \emptyset \vdash X \triangleright \diamond} \qquad [\text{Rec}]\ \frac{\Gamma \cdot X : \Sigma; \emptyset; \Sigma \vdash P \triangleright \diamond}{\Gamma; \emptyset; \Sigma \vdash \mu X.P \triangleright \diamond}$$

Fig. 3. Typing Rules for HOπ

3 Encodings

In this section we present a study of the expressiveness of the sub-calculi of HOπ.

We first define what is an encoding over typed calculus.

Definition 3.1. *Let $\langle L_1, T_1 \rangle$ be the calculus with process set L_1 and session type set T_1 and $\langle L_2, T_2 \rangle$ the calculus with process set L_2 and session type set T_2. Assuming mappings $[\![\cdot]\!] : L_1 \longrightarrow L_2$ and $(\!|\cdot|\!) : T_1 \longrightarrow T_2$ we write $\langle [\![\cdot]\!], (\!|\cdot|\!) \rangle : \langle L_1, T_1 \rangle \longrightarrow \langle L_2, T_2 \rangle$ for the encoding from $\langle L_1, T_1 \rangle$ to $\langle L_2, T_2 \rangle$.*

3.1 Encoding Properties

We require that a *good* encoding should transform preserve not only the syntax but also the operational, typing and behavioural semantics. Formally:

Definition 3.2 (Encoding Properties). *Let $\Gamma;\emptyset;\Sigma \vdash P \triangleright \diamond$ a process from calculus $\langle L_1, T_1\rangle$ and an encoding $\langle [\![\cdot]\!], \langle\!\langle\cdot\rangle\!\rangle\rangle : \langle L_1, T_1\rangle \longrightarrow \langle L_2, T_2\rangle$. Then*

1. *$\langle [\![\cdot]\!], \langle\!\langle\cdot\rangle\!\rangle\rangle$ is type preservering whenever $\Gamma;\emptyset;\Sigma \vdash P \triangleright \diamond$ implies $\langle\!\langle\Gamma\rangle\!\rangle;\emptyset;\langle\!\langle\Sigma\rangle\!\rangle \vdash [\![P]\!] \triangleright \diamond$*
2. *$\langle [\![\cdot]\!], \langle\!\langle\cdot\rangle\!\rangle\rangle$ satisfies operational correspondence whenever*
   - *If $P \longrightarrow P'$ then $[\![P]\!] \twoheadrightarrow [\![P']\!]$ and $\langle\!\langle\Gamma\rangle\!\rangle;\emptyset;\langle\!\langle\Sigma'\rangle\!\rangle \vdash [\![P']\!] \triangleright \diamond$.*
   - *If $[\![P]\!] \longrightarrow [\![Q]\!]$ then $\exists P'$ such that $P \longrightarrow P'$ and $\langle\!\langle\Gamma\rangle\!\rangle;\emptyset;\langle\!\langle\Sigma_1\rangle\!\rangle \vdash [\![Q]\!] \approx \Sigma_2 \vdash P' \triangleright \diamond$.*
3. *$\langle [\![\cdot]\!], \langle\!\langle\cdot\rangle\!\rangle\rangle$ is fully abstract whenever $\Gamma;\emptyset;\Sigma_1 \vdash P_1 \approx \Sigma_2 \vdash P_2 \triangleright \diamond$ if and only if $\langle\!\langle\Gamma\rangle\!\rangle;\emptyset;\langle\!\langle\Sigma_1\rangle\!\rangle \vdash [\![P_1]\!] \approx \langle\!\langle\Sigma_2\rangle\!\rangle \vdash [\![P_2]\!] \triangleright \diamond$.*

Encoding composition is closed on the above properties.

Proposition 3.1. *Let $\langle [\![\cdot]\!]^1, \langle\!\langle\cdot\rangle\!\rangle^1\rangle : \langle L_1, T_1\rangle \longrightarrow \langle L_2, T_2\rangle$ and $\langle [\![\cdot]\!]^2, \langle\!\langle\cdot\rangle\!\rangle^2\rangle : \langle L_2, T_2\rangle \longrightarrow \langle L_3, T_3\rangle$ encodings that respect the properties of definition 3.2. Then $\langle [\![\cdot]\!]^1, \langle\!\langle\cdot\rangle\!\rangle^1\rangle \cdot \langle [\![\cdot]\!]^2, \langle\!\langle\cdot\rangle\!\rangle^2\rangle$ also respect the properties of definition 3.2.*

Proof. Straightforward application of the definition of each property.

3.2 Encoding from session-π to HOπ

The semantics of the HOπ are powerfull enough to express the semantics of the standard session-π calculus.

Encoding from session-π without recursion to HOπ: We first encode the name passing semantics:

$$[\![k!\langle k'\rangle;P]\!] \stackrel{\text{def}}{=} k!\langle(z)z?(X);X\langle k'\rangle\rangle;[\![P]\!] \qquad \langle\!\langle !\langle S_1\rangle;S_2\rangle\!\rangle \stackrel{\text{def}}{=} !\langle \mathit{todo}\rangle;\langle\!\langle S_2\rangle\!\rangle$$

$$[\![k?(x);P]\!] \stackrel{\text{def}}{=} k?(X);(\nu\ s)(X\langle s\rangle \mid \overline{s}!\langle(x)[\![P]\!]\rangle;\mathbf{0}) \qquad \langle\!\langle ?(S_1);S_2\rangle\!\rangle \stackrel{\text{def}}{=} ?(\mathit{todo});\langle\!\langle S_2\rangle\!\rangle$$

References

1. Dimitris Mostrous and Nobuko Yoshida. Two session typing systems for higher-order mobile processes. In *TLCA'07*, volume 4583 of *LNCS*, pages 321–335. Springer, 2007.