On the Relative Expressiveness of Higher-Order Session Processes

Dimitrios Kouzapas[1], Jorge A. Pérez[2(✉)], and Nobuko Yoshida[3]

[1] University of Glasgow, Glasgow, UK
[2] University of Groningen, Groningen, The Netherlands
j.a.perez@rug.nl
[3] Imperial College London, London, UK

Abstract. By integrating constructs from the λ-calculus and the π-calculus, in *higher-order process calculi* exchanged values may contain processes. This paper studies the relative expressiveness of HOπ, the higher-order π-calculus in which communications are governed by *session types*. Our main discovery is that HO, a subcalculus of HOπ which lacks name-passing and recursion, can serve as a new core calculus for session-typed higher-order concurrency. By exploring a new bisimulation for HO, we show that HO can encode HOπ fully abstractly (up to typed contextual equivalence) more precisely and efficiently than the first-order session π-calculus (π). Overall, under session types, HOπ, HO, and π are equally expressive; however, HOπ and HO are more tightly related than HOπ and π.

1 Introduction

Type-preserving compilations are important in the design of functional and object-oriented languages: type information has been used to, e.g., justify code optimizations and reason about programs [18,21,38]. A vast literature on *expressiveness* in concurrency theory also studies compilations (or *encodings*) [8,10,16,26,31]: they are used to transfer reasoning techniques across calculi, and to implement process constructs using simpler ones. In this work, we study *relative expressiveness* via *type-preserving encodings* for HOπ, a *higher-order* process language that integrates message-passing concurrency with functional features. We consider source and target calculi coupled with *session types* [11] denoting interaction protocols. Building on untyped frameworks for relative expressiveness [10], we propose type preservation as a new criterion for *precise encodings*. We identify HO, a new core calculus for higher-order session concurrency without name passing. We show that HO can encode HOπ precisely and efficiently. Requiring type preservation makes this encoding far from trivial: we crucially exploit advances on session type duality [2,3] and recent characterisations of typed contextual equivalence [14]. We develop a full hierarchy of variants of HOπ based on precise encodings: our encodings are type-preserving and fully abstract up to typed behavioural equalities. Figure 1 illustrates this hierarchy; the variants of HOπ are explained next.

P. Thiemann (Ed.): ESOP 2016, LNCS 9632, pp. 446–475, 2016.
DOI: 10.1007/978-3-662-49498-1_18

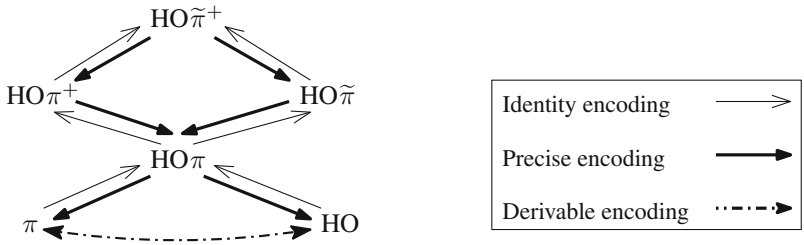

Fig. 1. Encodability in higher-order sessions. Precise encodings are defined in Definition 15.

Context. In *session-based concurrency*, interactions are organised into *sessions*, basic communication units. Interaction patterns can then be abstracted as *session types* [11], against which specifications may be checked. Session type $?(U);S$ (resp. $!\langle U\rangle;S$) describes a protocol that first receives (resp. sends) a value of type U and then continues as protocol S. Also, given an index set I, types $\&\{l_i : S_i\}_{i\in I}$ and $\oplus\{l_i : S_i\}_{i\in I}$ define, respectively, external and internal choice constructs for a labelled choice mechanism; types $\mu \mathsf{t}.S$ and end denote recursive and completed protocols, respectively. In the π-calculus, session types describe the intended interactive behaviour of the names in a process [11].

Session-based concurrency has also been casted in higher-order process calculi which, by combining features from the λ-calculus and the π-calculus, enable the exchange of values that may contain processes [9,22]. The higher-order calculus with sessions studied here, called $\mathsf{HO}\pi$, can specify protocols involving *code mobility*: it includes constructs for synchronisation along shared names, session communication (value passing, labelled choice) along linear names, recursion, (first-order) abstractions and applications. That is, values in communications include names but also (first-order) abstractions—functions from name identifiers to processes. (In contrast, we rule out *higher-order* abstractions—functions from processes to processes.) Abstractions can be linear or shared; their types are denoted $C\multimap\diamond$ and $C\to\diamond$, respectively (C denotes a name). In $\mathsf{HO}\pi$ we may have processes with a session type such as, e.g.,

$$S = \&\{up : ?(C\multimap\diamond); !\langle \mathsf{ok}\rangle; \mathsf{end}\ ,\ down : !\langle C\to\diamond\rangle; !\langle \mathsf{ok}\rangle; \mathsf{end}\ ,\ quit : !\langle \mathsf{bye}\rangle; \mathsf{end}\}\ .$$

S is the type of a server that offers (&) three different behaviours to a client: to *upload* a linear function, to *download* a shared function, or to *quit* the protocol. Following a client's selection ($\oplus$), the server sends a message (ok or bye) before closing the session.

Expressiveness of $\mathsf{HO}\pi$. We study the type-preserving, relative expressivity of $\mathsf{HO}\pi$. As expected from known literature in the untyped setting [32], the first-order session π-calculus [11] (here denoted π) can encode $\mathsf{HO}\pi$ preserving session types. In this paper, our *main discovery* is that $\mathsf{HO}\pi$ without name-passing and recursion can serve as a core calculus for higher-order session concurrency.

We call this core calculus HO. We show that HO can encode HOπ more efficiently than π. In addition, in the higher-order session typed setting, HO offers more tractable bisimulation techniques than π (cf. Sect. 5.2).

Challenges and Contributions. We assess the expressivity of HOπ, HO, and π as delineated by session types. We introduce *type-preserving encodings*: type information is used to define encodings and to retain the semantics of session protocols. Indeed, not only we require well-typed source processes are encoded into well-typed target processes: we demand that session type constructs (input, output, branching, select) used to type the source process are preserved by the typing of the target process. This criterion is included in our notion of *precise encoding* (Definition 15), which extends encodability criteria for untyped processes with *full abstraction*. Full abstraction results are stated up to two behavioural equalities that characterise barbed congruence: *characteristic bisimilarity* ($\approx^{\mathsf{C}}$, defined in [14]) and *higher-order bisimilarity* ($\approx^{\mathsf{H}}$), introduced in this work. It turns out that $\approx^{\mathsf{H}}$ offers more direct reasoning than $\approx^{\mathsf{C}}$. Using precise encodings we establish strong correspondences between HOπ and its variants—see below.

One main contribution is an encoding of HOπ into HO (Sect. 7.1). Since HO lacks both name-passing and recursion, this encoding involves two *key challenges*:

a. In known (typed) encodings of name-passing into process-passing [36] only the output capability of names can be sent—a received name cannot be used in later inputs. This is far too limiting in HOπ, where session names may be passed around (*delegation*) and types describe interaction *structures*, rather than "loose" name capabilities.
b. Known encodings of recursion in untyped higher-order calculi do not carry over to session typed calculi such as HOπ, because linear abstractions cannot be copied/duplicated. Hence, the discipline of session types limits the possibilities for representing infinite behaviours—even simple forms, such as input-guarded replication.

Our encoding overcomes these two obstacles, as we discuss in Sect. 2.

Additional technical contributions include: (i) the encodability of HO into π (Sect. 7.2); (ii) extensions of our encodability results to richer settings (Sect. 8); (iii) a non encodability result showing that shared names strictly add expressive power to session calculi (Sect. 7.4). In essence, (i) extends known results for untyped processes [32] to the session typed setting. Concerning (ii), we develop extensions of our encodings to

- The extension of HOπ with *higher-order* abstractions (HOπ^+);
- The extension of HOπ with polyadic name passing and abstraction (HO$\widetilde{\pi}$);
- The super-calculus of HOπ^+ and HO$\widetilde{\pi}$ (HO$\widetilde{\pi}^+$), equivalent to the calculus in [22].

Figure 1 summarises our encodability results: they connect HOπ with existing higher-order process calculi [22], and highlight the status of HO as the core calculus for session concurrency. Finally, to our knowledge we are the first to prove the non encodability result (iii), exploiting session determinacy and typed equivalences.

Outline. Section 2 overviews key ideas of the precise encoding of HOπ into π. Section 3 presents HOπ and its subcalculi (HO and π); Sect. 4 summarises their session type system. Section 5 presents behavioural equalities for HOπ: we recall definitions of barbed congruence and characteristic bisimilarity [14], and introduce higher-order bisimilarity. We show that these three typed relations coincide (Theorem 2). Section 6 defines *precise encodings* by extending encodability criteria for untyped processes. Section 7 gives precise encodings of HOπ into HO and of HOπ into π (Theorems 3 and 4). Mutual encodings between π and HO are derivable; all these calculi are thus equally expressive. Via empirical and formal comparisons between these two precise encodings, in Sect. 7.3 we establish that HOπ and HO are more tightly related than HOπ and π (Theorem 5). Moreover, we prove the impossibility of encoding communication along shared names using linear names (Theorem 6). In Sect. 8 we show encodings of HOπ^+ and HO$\widetilde{\pi}$ into HOπ (Theorems 7 and 8). Section 9 collects concluding remarks and reviews related works. Omitted definitions and proofs are in [15].

2 Overview: Encoding Name Passing into Process Passing

A Precise Encoding of Name-Passing into Process-Passing. As mentioned above, our encoding of HOπ into HO (Sect. 7.1) should (a) enable the communication of arbitrary names, as required to represent delegation, and (b) address the fact that linearity of session types limits the possibilities for representing infinite behaviour. To encode name passing into HO we "pack" the name to be sent into an abstraction; upon reception, the receiver "unpacks" this object following a precise protocol on a fresh session:

$$[\![a!\langle b\rangle.P]\!] = a!\langle \lambda z.\ z?(x).(x\,b)\rangle.[\![P]\!]$$
$$[\![a?(x).Q]\!] = a?(y).(\nu\, s)(y\,s \mid \overline{s}!\langle \lambda x.\ [\![Q]\!]\rangle.\mathbf{0})$$

Above, a, b are names and s and $\overline{s}$ are linear session names (*endpoints*). Processes $a!\langle V\rangle.P$ and $a?(x).P$ denote output and input at a; abstractions and applications are denoted $\lambda x.P$ and $(\lambda x.P)a$. Processes $(\nu\, s)(P)$ and $\mathbf{0}$ represent hiding and inaction. Thus, following a communication on a, a (deterministic) reduction between s and $\overline{s}$ guarantees that b is properly unpacked by means of abstraction passing and appropriate applications. Notice that the above encoding requires three extra reduction steps to mimic a name communication step in HOπ. Also, an output action in the source process is translated into an output action in the encoded process (and similarly for input). This is key to ensure the preservation of session type operators mentioned above (cf. Definition 13).

As hinted at above, a challenge in encoding $\mu X.P$ is preserving linearity of session names. Intuitively, we encode the recursion body P as an abstraction $\lambda\tilde{x}.\,[\![P]\!]_\sigma$ in which each session name of P (included in set σ) is converted into a name variable in $\tilde{x}$. Since $\lambda\tilde{x}.\,[\![P]\!]_\sigma$ does not mention (linear) session names, we may embed it into a "duplicator" process which implements recursion using higher-order communication [40]. The encoding of the recursion variable

X invokes this duplicator in a by-need fashion: it receives $\lambda\tilde{x}.\llbracket P \rrbracket_\sigma$ and uses two copies of it: one copy allows us to obtain P through the application of the session names of P; the other allows us to invoke the duplicator when needed. Interestingly, for this encoding to work we require non-tail recursive session types; this exploits recent advances on the theory of duality for session types [2,3].

A Plausible Encoding That is Not Precise. Our notion of *precise encoding* (Definition 15) requires the translation of both process and types; it admits only process mappings that preserve session types *and* are fully abstract. Thus, our encodings not only exhibit strong behavioural correspondences, but also relate source and target processes with consistent communication structures described by session types. These requirements are demanding and make our developments far from trivial. In particular, requiring type preservation may rule out other plausible encoding strategies. To illustrate this point, consider the following alternative encoding of name-passing into HO:[1]

$$\begin{aligned}\llbracket a?(x).Q \rrbracket^u &= a!\langle \lambda x.\llbracket Q \rrbracket^u \rangle.\mathbf{0}\\ \llbracket a!\langle b \rangle.P \rrbracket^u &= a?(x).(x\,b \mid \llbracket P \rrbracket^u)\end{aligned}$$

Intuitively, the encoding of input takes the initiative by sending an abstraction containing the encoding of its continuation Q; the encoding of output applies this received value to name b. Hence, this mapping entails a "role inversion": outputs are translated into inputs, and inputs are translated into outputs. Although fairly reasonable, we will see that the encoding $\llbracket\cdot\rrbracket^u$ is *not type preserving*. Consequently, it is also not *precise*. Since individual prefixes (input, output, branching, select) represent actions in a structured communication sequence (i.e., a protocol abstracted by a session type), the encoding $\llbracket\cdot\rrbracket^u$ would simply alter the meaning of the session protocol in the source language.

3 Higher-Order Session π-Calculi

We introduce the *higher-order session π-calculus* (HOπ). We define syntax, operational semantics, and its sub-calculi (π and HO). A type system and behavioural equivalences are introduced in Sects. 4 and 5. Extensions of HOπ with higher-order abstractions and polyadicity (noted HOπ^+ and HO$\widetilde{\pi}$, respectively) are discussed in Sect. 8.

$$\begin{array}{llll} u,w & ::= & n \mid x,y,z & \qquad n ::= a,b \mid s,\overline{s} \qquad V,W ::= u \mid \boxed{\lambda x.P} \\ P,Q & ::= & u!\langle V\rangle.P \mid u?(x).P \mid u \triangleleft l.P \mid u \triangleright \{l_i : P_i\}_{i\in I} \mid \boxed{Vu} \mid P \mid Q \mid (\nu n)P \mid \mathbf{0} \mid X \mid \mu X.P \end{array}$$

Fig. 2. Syntax of HOπ. While HO lacks shaded constructs, π lacks boxed constructs.

[1] This encoding was suggested by a reviewer of a previous version of this paper.

3.1 HOπ: Syntax, Operational Semantics, and Subcalculi

Syntax. The syntax of HOπ is defined in Fig. 2. HOπ it is a subcalculus of the language studied in [22]. It is also a variant of the language that we investigated in [14], which includes higher-order value applications.

Names $a, b, c, \ldots$ (resp. $s, \overline{s}, \ldots$) range over shared (resp. session) names. Names $m, n, t, \ldots$ are session or shared names. Dual endpoints are $\overline{n}$ with $\overline{\overline{s}} = s$ and $\overline{a} = a$. Variables are denoted with $x, y, z, \ldots$, and recursive variables are denoted with $X, Y \ldots$. An abstraction $\lambda x.P$ is a process P with name parameter x. *Values* V, W include identifiers $u, v, \ldots$ and abstractions $\lambda x.P$ (first- and higher-order values, resp.).

Process terms $P, Q, \ldots$ include usual prefixes for sending/receiving values V. Processes $u \triangleleft l.P$ and $u \triangleright \{l_i : P_i\}_{i \in I}$ are the usual session processes for selecting and branching [11]. Process $V\,u$ is the application which substitutes name u on the abstraction V. Typing ensures that V is not a name. Recursion $\mu X.P$ binds the recursive variable X in P. Constructs for inaction $\mathbf{0}$, parallel composition $P_1 \mid P_2$, and name restriction $(\nu\, n)P$ are standard. Session name restriction $(\nu\, s)P$ simultaneously binds endpoints s and $\overline{s}$ in P. Functions $\mathtt{fv}(P)$ and $\mathtt{fn}(P)$ denote the sets of free variables and names. We assume V in $u!\langle V\rangle.P$ does not include free recursive variables X. If $\mathtt{fv}(P) = \emptyset$, we call P *closed*.

Operational Semantics. The *operational semantics* of HOπ is defined in terms of a reduction relation, denoted $\longrightarrow$ and given in Fig. 3 (top). We briefly describe the rules. Rule [App] defines name application. Rule [Pass] defines a shared interaction at n (with $\overline{n} = n$) or a session interaction. Rule [Sel] is the standard rule for labelled choice/selection. Other rules are standard π-calculus rules. Reduction is closed under *structural congruence*, noted $\equiv$ (cf. Fig. 3, bottom). We assume the expected extension of $\equiv$ to values V. We write $\longrightarrow^*$ for a multi-step reduction.

$$(\lambda x.P)\,u \longrightarrow P\{u/x\} \quad [\text{App}] \qquad n!\langle V\rangle.P \mid \overline{n}?(x).Q \longrightarrow P \mid Q\{V/x\} \quad [\text{Pass}]$$

$$n \triangleleft l_j.Q \mid \overline{n} \triangleright \{l_i : P_i\}_{i \in I} \longrightarrow Q \mid P_j \ \ (j \in I) \quad [\text{Sel}] \qquad P \longrightarrow P' \Rightarrow (\nu n)P \longrightarrow (\nu n)P' \quad [\text{Res}]$$

$$P \longrightarrow P' \ \Rightarrow\ P \mid Q \longrightarrow P' \mid Q \quad [\text{Par}] \qquad P \equiv Q \longrightarrow Q' \equiv P' \ \Rightarrow\ P \longrightarrow P' \quad [\text{Cong}]$$

$$P \mid \mathbf{0} \equiv P \qquad P_1 \mid P_2 \equiv P_2 \mid P_1 \qquad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3 \qquad (\nu n)\mathbf{0} \equiv \mathbf{0}$$

$$P \mid (\nu n)Q \equiv (\nu n)(P \mid Q)\ (n \notin \mathtt{fn}(P)) \qquad \mu X.P \equiv P\{\mu X.P/X\} \qquad P \equiv Q \text{ if } P \equiv_\alpha Q$$

Fig. 3. Operational semantics of HOπ.

Subcalculi. As motivated in the introduction, we define two *subcalculi* of HOπ:

- The *core higher-order session calculus*, denoted HO, lacks recursion and name passing; its formal syntax is obtained from Fig. 2 by excluding constructs in grey.

- The *session π-calculus*, denoted π, lacks higher-order communication but includes recursion; its formal syntax is obtained from Fig. 2 *by excluding constructs in* boxes.

Let $\mathsf{C} \in \{\mathsf{HO}\pi, \mathsf{HO}, \pi\}$. We write $\mathsf{C}^{-\mathsf{sh}}$ to denote the calculus C without shared names: we delete a, b from n. In Sect. 7 we shall demonstrate that $\mathsf{HO}\pi$, HO, and π have the same expressivity, and that C is strictly more expressive than $\mathsf{C}^{-\mathsf{sh}}$.

4 Session Types for $\mathsf{HO}\pi$

We define a session type system for $\mathsf{HO}\pi$ and state *type soundness* (Theorem 1), its main property. Our system distills the key features of [22,23] and so it is simpler.

The syntax of types of $\mathsf{HO}\pi$ follows. We write $\diamond$ to denote the process type.

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

Value type U includes first-order types C and higher-order types L. Types $C \to \diamond$ and $C \multimap \diamond$ denote *shared* and *linear* higher-order types, respectively. Session types, denoted by S, follow the standard binary session type syntax [11], with the extension that carried types U may be higher-order. Shared channel types are denoted $\langle S \rangle$ and $\langle L \rangle$. Types of HO exclude C from value types U; the types of π exclude L and $\langle L \rangle$. From each $\mathsf{C} \in \{\mathsf{HO}\pi, \mathsf{HO}, \pi\}$, $\mathsf{C}^{-\mathsf{sh}}$ excludes shared name types ($\langle S \rangle$ and $\langle L \rangle$), from name type C.

We write S dual S' if S is the *dual* of S'. Intuitively, session type duality is obtained by dualising ! by ?, ? by !, $\oplus$ by &, and &by $\oplus$, including the fixed point construction. We use the *co-inductive* definition of duality given in [2].

We consider shared, linear, and session *environments*, denoted Γ, Λ, and Δ, resp.:

$$\begin{array}{lll} \Gamma & ::= & \emptyset \mid \Gamma \cdot x : C \to \diamond \mid \Gamma \cdot u : \langle S \rangle \mid \Gamma \cdot u : \langle L \rangle \mid \Gamma \cdot X : \Delta \\ \Lambda & ::= & \emptyset \mid \Lambda \cdot x : C \multimap \diamond \\ \Delta & ::= & \emptyset \mid \Delta \cdot u : S \end{array}$$

Γ maps variables and shared names to value types, and recursive variables to session environments; it admits weakening, contraction, and exchange principles. Λ maps variables to linear higher-order types; Δ maps session names to session types. Both Λ and Δ are only subject to exchange. Domains of Γ, Λ and Δ are assumed pairwise distinct. $\Delta_1 \cdot \Delta_2$ is the disjoint union of Δ_1 and Δ_2. We focus on *balanced* session environments:

Definition 1 (Balanced). *We say that a session environment Δ is* balanced *if whenever $s : S_1, \overline{s} : S_2 \in \Delta$ then S_1* dual *S_2.*

$$\text{(Prom)}\ \frac{\Gamma;\emptyset;\emptyset \vdash V \triangleright C \multimap \diamond}{\Gamma;\emptyset;\emptyset \vdash V \triangleright C \to \diamond} \qquad \text{(EProm)}\ \frac{\Gamma;\Lambda \cdot x : C \multimap \diamond;\Delta \vdash P \triangleright \diamond}{\Gamma \cdot x : C \to \diamond;\Lambda;\Delta \vdash P \triangleright \diamond} \qquad \text{(Abs)}\ \frac{\Gamma;\Lambda;\Delta_1 \vdash P \triangleright \diamond \quad \Gamma;\emptyset;\Delta_2 \vdash x \triangleright C}{\Gamma \backslash x;\Lambda;\Delta_1 \backslash \Delta_2 \vdash \lambda x.P \triangleright C \multimap \diamond}$$

$$\text{(App)}\ \frac{U = C \multimap \diamond \vee C \to \diamond \quad \Gamma;\Lambda;\Delta_1 \vdash V \triangleright U \quad \Gamma;\emptyset;\Delta_2 \vdash u \triangleright C}{\Gamma;\Lambda;\Delta_1 \cdot \Delta_2 \vdash V\,u \triangleright \diamond} \qquad \text{(Send)}\ \frac{u : S \in \Delta_1 \cdot \Delta_2 \quad \Gamma;\Lambda_1;\Delta_1 \vdash P \triangleright \diamond \quad \Gamma;\Lambda_2;\Delta_2 \vdash V \triangleright U}{\Gamma;\Lambda_1 \cdot \Lambda_2;((\Delta_1 \cdot \Delta_2) \setminus u : S) \cdot u : !\langle U\rangle;S \vdash u!\langle V\rangle.P \triangleright \diamond}$$

$$\text{(Rcv)}\ \frac{\Gamma;\Lambda_1;\Delta_1 \cdot u : S \vdash P \triangleright \diamond \quad \Gamma;\Lambda_2;\Delta_2 \vdash x \triangleright U}{\Gamma \backslash x;\Lambda_1 \cdot \Lambda_2;\Delta_1 \backslash \Delta_2 \cdot u : ?(U);S \vdash u?(x).P \triangleright \diamond}$$

$$\text{(Req)}\ \frac{\Gamma;\emptyset;\emptyset \vdash u \triangleright U_1 \quad \Gamma;\Lambda;\Delta_1 \vdash P \triangleright \diamond \quad \Gamma;\emptyset;\Delta_2 \vdash V \triangleright U_2 \quad (U_1 = \langle S\rangle \wedge U_2 = S) \vee (U_1 = \langle L\rangle \wedge U_2 = L)}{\Gamma;\Lambda;\Delta_1 \cdot \Delta_2 \vdash u!\langle V\rangle.P \triangleright \diamond}$$

$$\text{(Acc)}\ \frac{\Gamma;\emptyset;\emptyset \vdash u \triangleright U_1 \quad \Gamma;\Lambda_1;\Delta_1 \vdash P \triangleright \diamond \quad \Gamma;\Lambda_2;\Delta_2 \vdash x \triangleright U_2 \quad (U_1 = \langle S\rangle \wedge U_2 = S) \vee (U_1 = \langle L\rangle \wedge U_2 = L)}{\Gamma \backslash x;\Lambda_1 \backslash \Lambda_2;\Delta_1 \backslash \Delta_2 \vdash u?(x).P \triangleright \diamond}$$

Fig. 4. Selected typing rules for HOπ.

Given the above intuitions for environments, the typing judgements for values V and processes P are self-explanatory. They are denoted $\Gamma;\Lambda;\Delta \vdash V \triangleright U$ and $\Gamma;\Lambda;\Delta \vdash P \triangleright \diamond$.

Figure 4 gives selected typing rules; see [15] for a full account. The shared type $C \to \diamond$ is derived using rule (Prom) only if the value has a linear type with an empty linear environment. Rule (EProm) allows us to freely use a shared type variable as linear. Abstraction values are typed with rule (Abs). Application typing is governed by rule (App): we expect the type C of an application name u to match the type of the application variable x (i.e., $C \multimap \diamond$ or $C \to \diamond$). In rule (Send), the type U of value V should appear as a prefix in the session type $!\langle U\rangle;S$ of u. Rule (Rcv) is its dual. Rules (Req) and (Acc) type interaction along shared names; the type of the sent/received object (S and L, resp.) should match the type of the sent/received subject ($\langle S\rangle$ and $\langle L\rangle$, resp.).

Definition 2. *We define the relation* $\longrightarrow$ *on session environments as:*

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

We state type soundness for HOπ; it implies type soundness for HO, π, and $\mathsf{C}^{-\mathsf{sh}}$.

Theorem 1 (Type Soundness). Suppose $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$ with Δ balanced. Then $P \longrightarrow P'$ implies $\Gamma;\emptyset;\Delta' \vdash P' \triangleright \diamond$ and $\Delta = \Delta'$ or $\Delta \longrightarrow \Delta'$ with Δ' balanced.

5 Behavioural Theory for HOπ

We first define reduction-closed, barbed congruence ($\cong$, Definition 7) as the reference equivalence relation for HOπ processes. We then define two characterisations

of $\cong$: *characteristic* and *higher-order bisimilarities* (denoted $\approx^{\mathtt{C}}$ and $\approx^{\mathtt{H}}$, cf. Definitions 8 and 9).

5.1 Reduction-Closed, Barbed Congruence ($\cong$)

We consider *typed relations* $\mathfrak{R}$ that relate closed terms whose session environments are balanced and *confluent*:

Definition 3 (Session Environment Confluence). *Let $\longrightarrow^*$ denote multi-step reduction as in Definition 2. We denote $\Delta_1 \rightleftharpoons \Delta_2$ if there exists Δ such that $\Delta_1 \longrightarrow^* \Delta$ and $\Delta_2 \longrightarrow^* \Delta$.*

Definition 4 (Typed Relation). *We say that $\Gamma;\emptyset;\Delta_1 \vdash P \triangleright \diamond \;\mathfrak{R}\; \Gamma;\emptyset;\Delta_2 \vdash Q \triangleright \diamond$ is a* typed relation *whenever P and Q are closed; Δ_1 and Δ_2 are balanced; and $\Delta_1 \rightleftharpoons \Delta_2$. We write $\Gamma;\Delta_1 \vdash P \;\mathfrak{R}\; \Delta_2 \vdash Q$ for the typed relation $\Gamma;\emptyset;\Delta_1 \vdash P \triangleright \diamond \;\mathfrak{R}\; \Gamma;\emptyset;\Delta_2 \vdash Q \triangleright \diamond$.*

As usual, a *barb* $\downarrow_n$ is an observable on an output prefix with subject n [20]. A *weak barb* $\Downarrow_n$ is a barb after zero or more reduction steps. Typed barbs $\downarrow_n$ (resp. $\Downarrow_n$) occur on typed processes $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$. When n is a session name we require that its dual endpoint $\overline{n}$ is not present in the session environment Δ:

Definition 5 (Barbs). *Let P be a closed process. We define:*

1. *$P \downarrow_n$ if $P \equiv (\nu\, \tilde{m})(n!\langle V\rangle.P_2 \mid P_3), n \notin \tilde{m}$.*

2. *$\Gamma;\Delta \vdash P \downarrow_n$ if $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$ with $P \downarrow_n$ and $\overline{n} \notin \mathsf{dom}(\Delta)$.*
   $\Gamma;\Delta \vdash P \Downarrow_n$ if $P \longrightarrow^* P'$ and $\Gamma;\Delta' \vdash P' \downarrow_n$.

To define a congruence relation, we introduce the family $\mathbb{C}$ of contexts:

Definition 6 (Context). *A context $\mathbb{C}$ is defined as:*

$$\begin{array}{rcl} \mathbb{C} & ::= & - \mid u!\langle V\rangle.\mathbb{C} \mid u?(x).\mathbb{C} \mid u!\langle \lambda x.\mathbb{C}\rangle.P \mid (\nu\, n)\mathbb{C} \mid (\lambda x.\mathbb{C})u \mid \mu X.\mathbb{C} \\ & \mid & \mathbb{C} \mid P \mid P \mid \mathbb{C} \mid u \triangleleft l.\mathbb{C} \mid u \triangleright \{l_1 : P_1, \cdots, l_i : \mathbb{C}, \cdots, l_n : P_n\} \end{array}$$

Notation $\mathbb{C}[P]$ replaces the hole $-$ in $\mathbb{C}$ with P.

We define reduction-closed, barbed congruence [12].

Definition 7 (Barbed Congruence). *Typed relation $\Gamma;\Delta_1 \vdash P \;\mathfrak{R}\; \Delta_2 \vdash Q$ is a* reduction-closed, barbed congruence *whenever:*

1. *If $P \longrightarrow P'$ then there exist Q', Δ_1', Δ_2' such that $Q \longrightarrow^* Q'$ and $\Gamma;\Delta_1' \vdash P' \;\mathfrak{R}\; \Delta_2' \vdash Q'$;*
2. *If $\Gamma;\Delta_1 \vdash P \downarrow_n$ then $\Gamma;\Delta_2 \vdash Q \Downarrow_n$;*
3. *For all $\mathbb{C}$, there exist Δ_1'', Δ_2'' such that $\Gamma;\Delta_1'' \vdash \mathbb{C}[P] \;\mathfrak{R}\; \Delta_2'' \vdash \mathbb{C}[Q]$;*
4. *The symmetric cases of 1 and 2.*

The largest such relation is denoted with $\cong$.

5.2 Two Equivalence Relations: $\approx^{\mathsf{C}}$ and $\approx^{\mathsf{H}}$

A Typed Labelled Transition System. In [14] we have characterised reduction-closed, barbed congruence for $\mathsf{HO}\pi$ via a typed relation called *characteristic bisimilarity*. Its definition uses a *typed* labelled transition system (LTS) informed by session types. Given a label ℓ (a visible action or τ), we write $\Gamma;\emptyset;\Delta \vdash P \overset{\ell}{\longmapsto} \Delta' \vdash P'$ to denote (strong) transitions. Weak transitions are as expected: we write $\Longmapsto$ for the reflexive, transitive closure of $\overset{\tau}{\longmapsto}$, $\overset{\ell}{\Longmapsto}$ for $\Longmapsto\overset{\ell}{\longmapsto}\Longmapsto$, and $\overset{\hat{\ell}}{\Longmapsto}$ for $\overset{\ell}{\Longmapsto}$ if $\ell \neq \tau$ and $\Longmapsto$ otherwise. Intuitively, the transitions of a typed process should be enabled by its associated typing:

$$\text{if } P \overset{\ell}{\longmapsto} P' \text{ and } (\Gamma,\Delta) \overset{\ell}{\longmapsto} (\Gamma,\Delta') \text{ then } \Gamma;\emptyset;\Delta \vdash P \overset{\ell}{\longmapsto} \Delta' \vdash P'.$$

As an example of how types enable transitions, consider the rule for input:

$$\frac{\overline{s} \notin \mathsf{dom}(\Delta) \qquad \Gamma;\Lambda';\Delta' \vdash V \triangleright U \qquad V = m \vee V \equiv \{U\}_{\mathsf{c}} \vee V \equiv \lambda x.\, t?(y).(y\, x) \text{ with } t \text{ fresh}}{(\Gamma;\Lambda;\Delta \cdot s : ?(U);S) \overset{s?\langle V\rangle}{\longmapsto} (\Gamma;\Lambda\cdot\Lambda';\Delta\cdot\Delta'\cdot s : S)}$$

This rule states that a session environment can input a value if such a value is typed with an input prefix and is either a name m, a *characteristic value* $\{U\}_{\mathsf{c}}$, or a *trigger value* (the abstraction $\lambda x.\, t?(y).(y\, x)$). A characteristic value is the simplest process that inhabits a type (here, the type U carried by the input prefix). The above rule is used to limit the input actions that can be observed from a session input prefix. For more details on the typed LTS and the characteristic process definition see [14]. Moreover, we define a *(first-order) trigger process*:

$$t \Leftarrow V : U \overset{\mathsf{def}}{=} t?(x).(\nu\, s)([\![?(U);\mathsf{end}]\!]^s \mid \overline{s}!\langle V\rangle.\mathbf{0}) \tag{1}$$

The trigger process $t \Leftarrow V : U$ is is defined as a process input prefixed on a fresh name t: it applies a value on the *characteristic process* $[\![?(U);\mathsf{end}]\!]^s$ (see [14] for details).

Characterising $\cong$. We define *characteristic* and *higher-order* bisimilarities. While higher-order bisimilarity is a new equality, characteristic bisimilarity was introduced in [14].

Definition 8 (Characteristic Bisimilarity). *A typed relation* $\mathfrak{R}$ *is called a* characteristic bisimulation *if for all* $\Gamma;\Delta_1 \vdash P_1\ \mathfrak{R}\ \Delta_2 \vdash Q_1$

1. *Whenever* $\Gamma;\Delta_1 \vdash P_1 \overset{(\nu\,\tilde{m}_1)n!\langle V_1:U\rangle}{\longmapsto} \Delta_1' \vdash P_2$, *there exist* Q_2, V_2, Δ_2' *such that* $\Gamma;\Delta_2 \vdash Q_1 \overset{(\nu\,\tilde{m}_2)n!\langle V_2:U\rangle}{\Longmapsto} \Delta_2' \vdash Q_2$ *and, for fresh* t,

$$\Gamma;\Delta_1'' \vdash (\nu\,\tilde{m}_1)(P_2 \mid t \Leftarrow V_1 : U_1)\ \mathfrak{R}\ \Delta_2'' \vdash (\nu\,\tilde{m}_2)(Q_2 \mid t \Leftarrow V_2 : U_2)$$

2. *For all* $\Gamma;\Delta_1 \vdash P_1 \overset{\ell}{\longmapsto} \Delta_1' \vdash P_2$ *such that* ℓ *is not an output, there exist* Q_2, Δ_2' *such that* $\Gamma;\Delta_2 \vdash Q_1 \overset{\hat{\ell}}{\Longmapsto} \Delta_2' \vdash Q_2$ *and* $\Gamma;\Delta_1' \vdash P_2 \mathrel{\mathfrak{R}} \Delta_2' \vdash Q_2$*; and*
3. *The symmetric cases of 1 and 2.*

The largest such bisimulation is called characteristic bisimilarity *and denoted by* $\approx^{\mathsf{C}}$.

Interestingly, for reasoning about HOπ processes we can also exploit the simpler *higher-order bisimilarity*. We replace triggers as in (1) with *higher-order triggers*:

$$t \hookleftarrow V \stackrel{\text{def}}{=} t?(x).(\nu\, s)(x\, s \mid \overline{s}!\langle V\rangle.\mathbf{0}) \tag{2}$$

We may then define:

Definition 9 (Higher-Order Bisimilarity). *Higher-order bisimilarity, denoted by* $\approx^{\mathsf{H}}$*, is defined by replacing Clause (1) in Definition 8 with the following clause:*

Whenever $\Gamma;\Delta_1 \vdash P_1 \overset{(\nu\,\tilde{m}_1)n!\langle V_1\rangle}{\longmapsto} \Delta_1' \vdash P_2$ *then there exist* Q_2, V_2, Δ_2' *such that* $\Gamma;\Delta_2 \vdash Q_1 \overset{(\nu\,\tilde{m}_2)n!\langle V_2\rangle}{\Longmapsto} \Delta_2' \vdash Q_2$ *and, for fresh* t,
$\Gamma;\Delta_1'' \vdash (\nu\,\tilde{m}_1)(P_2 t \hookleftarrow V_1) \mathrel{\mathfrak{R}} \Delta_2'' \vdash (\nu\,\tilde{m}_2)(Q_2 \mid t \hookleftarrow V_2)$

We state the following important result, which attests the significance of $\approx^{\mathsf{H}}$:

Theorem 2. *Typed relations* $\cong$, $\approx^{\mathsf{H}}$, *and* $\approx^{\mathsf{C}}$ *coincide for* HOπ *processes.*

Proof. Coincidence of $\cong$ and $\approx^{\mathsf{C}}$ was established in [14]. Coincidence of $\approx^{\mathsf{H}}$ with $\cong$ and $\approx^{\mathsf{C}}$ is a new result: see [15] for details. □

Remark 1 (Comparison between $\approx^{\mathsf{H}}$ *and* $\approx^{\mathsf{C}}$*).* The key difference between $\approx^{\mathsf{H}}$ and $\approx^{\mathsf{C}}$ is in the trigger process considered. Because of the application in (2), $\approx^{\mathsf{H}}$ cannot be used to reason about processes in π. In contrast, $\approx^{\mathsf{C}}$ is more general: it can uniformly input characteristic, first- or higher-order values. This convenience comes at a price: the definition of (1) requires information on the type of V; in contrast, the higher-order trigger (2) is more generic and simple, as it works independently of the given type.

An up-to technique. Processes that do not use shared names are deterministic. The following up-to technique, based on determinacy properties, will be useful in proofs (Sect. 7). Recall that $\Gamma;\Delta \vdash P \overset{\tau}{\longmapsto} \Delta' \vdash P'$ denotes an internal (typed) transition.

Notation 1 (Deterministic Transitions). *We distinguish two kinds of* τ*-transitions:* session transitions*, noted* $\Gamma;\Delta \vdash P \overset{\tau_s}{\longmapsto} \Delta' \vdash P'$*, and* β-transitions*, noted* $\Gamma;\Delta \vdash P \overset{\tau_\beta}{\longmapsto} \Delta' \vdash P'$*. Intuitively,* $\overset{\tau_s}{\longmapsto}$ *results from a session communication (i.e., synchronization between two dual endpoints), while* $\overset{\tau_\beta}{\longmapsto}$ *results from an application. We write* $\Gamma;\Delta \vdash P \overset{\tau_d}{\longmapsto} \Delta' \vdash P'$ *to denote a session transition or a* β*-transition. See [15] for definitions of* $\overset{\tau_\beta}{\longmapsto}$ *and* $\overset{\tau_s}{\longmapsto}$.

We have the following determinacy property; see [15] for details.

Lemma 1 (τ-Inertness). *(1) Let $\Gamma;\Delta \vdash P \overset{\tau_d}{\longmapsto} \Delta' \vdash P'$ be a deterministic transition, with balanced Δ. Then $\Gamma;\Delta \vdash P \cong \Delta' \vdash P'$ with $\Delta \longrightarrow^* \Delta'$ balanced. (2) Let P be an $\mathsf{HO}\pi^{-\mathsf{sh}}$ process. Assume $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$. Then $P \longrightarrow^* P'$ implies $\Gamma;\Delta \vdash P \cong \Delta' \vdash P'$ with $\Delta \longrightarrow^* \Delta'$.*

We use Lemma 1 to prove Theorem 6, the negative result stated in Sect. 7.4. This property also enables us to define the following up-to technique, useful in full abstraction proofs. We write $\overset{\tau_d}{\Longmapsto}$ to denote a (possibly empty) sequence of deterministic steps $\overset{\tau_d}{\longmapsto}$.

Lemma 2 (Up-to Deterministic Transition). *Let $\Gamma;\Delta_1 \vdash P_1 \ \mathfrak{R} \ \Delta_2 \vdash Q_1$ such that if whenever:*

1. *$\forall (\nu\, \tilde{m}_1) n!\langle V_1\rangle$ such that $\Gamma;\Delta_1 \vdash P_1 \overset{(\nu\, \tilde{m}_1) n!\langle V_1\rangle}{\longmapsto} \Delta_3 \vdash P_3$ implies that $\exists Q_2, V_2$ such that $\Gamma;\Delta_2 \vdash Q_1 \overset{(\nu\, \tilde{m}_2) n!\langle V_2\rangle}{\Longmapsto} \Delta_2' \vdash Q_2$ and $\Gamma;\Delta_3 \vdash P_3 \overset{\tau_d}{\Longmapsto} \Delta_1' \vdash P_2$ and for fresh t: $\Gamma;\Delta_1'' \vdash (\nu\, \tilde{m}_1)(P_2 \mid t \hookleftarrow V_1) \ \mathfrak{R} \ \Delta_2'' \vdash (\nu\, \tilde{m}_2)(Q_2 \mid t \hookleftarrow V_2)$.*
2. *$\forall \ell \neq (\nu\, \tilde{m}) n!\langle V\rangle$ such that $\Gamma;\Delta_1 \vdash P_1 \overset{\ell}{\longmapsto} \Delta_3 \vdash P_3$ implies that $\exists Q_2$ such that $\Gamma;\Delta_1 \vdash Q_1 \overset{\hat{\ell}}{\Longmapsto} \Delta_2' \vdash Q_2$ and $\Gamma;\Delta_3 \vdash P_3 \overset{\tau_d}{\Longmapsto} \Delta_1' \vdash P_2$ and $\Gamma;\Delta_1' \vdash P_2 \ \mathfrak{R} \ \Delta_2' \vdash Q_2$.*
3. *The symmetric cases of 1 and 2.*

Then $\mathfrak{R} \subseteq \approx^H$.

6 Criteria for Typed Encodings

We define the formal notion of *encoding* by extending to a typed setting existing criteria for untyped processes (as in, e.g., [8,10,16,24,26,27,30,41]). We first define a typed calculus parametrised by a syntax, operational semantics, and typing. Based on this definition, in Sects. 7 and 8 we define concrete instances of (higher-order) typed calculi.

Definition 10 (Typed Calculus). *A* typed calculus $\mathcal{L}$ *is a tuple* $\langle \mathsf{C}, \mathcal{T}, \overset{\tau}{\longmapsto}, \approx, \vdash\rangle$ *where* C *and* $\mathcal{T}$ *are sets of processes and types, respectively; also,* $\overset{\tau}{\longmapsto}$, $\approx$, *and* $\vdash$ *denote a transition system, a typed equivalence, and a typing system for* C, *respectively.*

As we explain later, we write $\overset{\tau}{\longmapsto}$ to denote an operational semantics defined in terms of τ-transitions (to characterise reductions). Our notion of encoding considers mappings on both processes and types; these are denoted $[\![\cdot]\!]$ and $(\cdot)$, respectively:

Definition 11 (Typed Encoding). *Consider typed calculi* $\mathcal{L}_1 = \langle \mathsf{C}_1, \mathcal{T}_1, \overset{\tau}{\longmapsto}_1, \approx_1, \vdash_1\rangle$ *and* $\mathcal{L}_2 = \langle \mathsf{C}_2, \mathcal{T}_2, \overset{\tau}{\longmapsto}_2, \approx_2, \vdash_2\rangle$. *Given mappings* $[\![\cdot]\!] : \mathsf{C}_1 \to \mathsf{C}_2$ *and* $(\cdot) : \mathcal{T}_1 \to \mathcal{T}_2$, *we write* $\langle [\![\cdot]\!], (\cdot)\rangle : \mathcal{L}_1 \to \mathcal{L}_2$ *to denote the* typed encoding *of* $\mathcal{L}_1$ *into* $\mathcal{L}_2$.

Mapping $(\!|\cdot|\!)$ extends to typing environments, e.g., $(\!|\Delta \cdot u : S|\!) = (\!|\Delta|\!) \cdot u : (\!|S|\!)$. We introduce syntactic criteria for typed encodings. Let σ denote a substitution of names for names (a renaming, as usual). Given environments Δ and Γ, we write $\sigma(\Delta)$ and $\sigma(\Gamma)$ to denote the effect of applying σ on the domains of Δ and Γ (clearly, $\sigma(\Gamma)$ concerns only shared names in Γ: process and recursive variables in Γ are not affected by σ).

Definition 12 (Syntax Preservation). *We say that typed encoding* $\langle [\![\cdot]\!], (\!|\cdot|\!)\rangle : \mathcal{L}_1 \to \mathcal{L}_2$ *is* syntax preserving *if it is:*

1. Homomorphic wrt parallel, *if* $(\!|\Gamma|\!); \emptyset; (\!|\Delta_1 \cdot \Delta_2|\!) \vdash_2 [\![P_1 \mid P_2]\!] \triangleright \diamond$ *then* $(\!|\Gamma|\!); \emptyset; (\!|\Delta_1|\!) \cdot (\!|\Delta_2|\!) \vdash_2 [\![P_1]\!] \mid [\![P_2]\!] \triangleright \diamond$.
2. Compositional wrt restriction, *if* $(\!|\Gamma|\!); \emptyset; (\!|\Delta|\!) \vdash_2 [\![(\nu\, n)P]\!] \triangleright \diamond$ *then* $(\!|\Gamma|\!); \emptyset; (\!|\Delta|\!) \vdash_2 (\nu\, n)[\![P]\!] \triangleright \diamond$.
3. Name invariant, *if* $(\!|\sigma(\Gamma)|\!); \emptyset; (\!|\sigma(\Delta)|\!) \vdash_2 [\![\sigma(P)]\!] \triangleright \diamond$ *then* $\sigma((\!|\Gamma|\!)); \emptyset; \sigma((\!|\Delta|\!)) \vdash_2 \sigma([\![P]\!]) \triangleright \diamond$, *for any injective renaming of names* σ.

Homomorphism wrt parallel (used in, e.g., [26,27]) expresses that encodings should preserve the distributed topology of source processes. This criterion is appropriate for both encodability and non encodability results; in our setting, it is induced by rules for typed composition. Compositionality wrt restriction is also supported by typing and is useful in our encodability results (Sect. 7). The name invariance criterion follows [10,16].

We now state *type preservation*, a static criterion on the mapping $(\!|\cdot|\!) : \mathcal{T}_1 \to \mathcal{T}_2$: it ensures that type operators are preserved. The source and target languages that we consider here share five (session) type operators: input, output, recursion (binary operators); selection and branching (n-ary operators). Type preservation enables us to focus on mappings $(\!|\cdot|\!)$ that always translate a type operator into itself. This is key to retain the meaning of structured protocols: as session types operators abstract communication behaviour, type preserving encodings help us maintain behaviour across translations.

Definition 13 (Type Preservation). *The typed encoding* $\langle [\![\cdot]\!], (\!|\cdot|\!)\rangle : \mathcal{L}_1 \to \mathcal{L}_2$ *is* type preserving *if for every k-ary type operator* op *in* $\mathcal{T}_1$ *it holds that*

$$(\!|\mathsf{op}(T_1, \cdots, T_k)|\!) = \mathsf{op}((\!|T_1|\!), \cdots, (\!|T_k|\!))$$

Example 1. Following the discussion in Sect. 2, let $(\!|\cdot|\!)_u$ be a mapping on session types such that $(\!|!\langle U\rangle; S|\!)_u = ?((\!|U|\!)_u); (\!|S|\!)_u$ and $(\!|?(U); S|\!)_u = !\langle(\!|U|\!)_u\rangle; (\!|S|\!)_u$ (other type operators are translated homomorphically). That is, $(\!|\cdot|\!)_u$ translates the output type operator into an input type operator (and viceversa). Therefore, $(\!|\cdot|\!)_u$ does not satisfy type preservation.

Next we define semantic criteria for typed encodings:

Definition 14 (Semantic Preservation). *Consider two typed calculi* $\mathcal{L}_1$ *and* $\mathcal{L}_2$, *defined as* $\mathcal{L}_1 = \langle \mathsf{C}_1, \mathcal{T}_1, \overset{\tau}{\longmapsto}_1, \approx_1, \vdash_1\rangle$ *and* $\mathcal{L}_2 = \langle \mathsf{C}_2, \mathcal{T}_2, \overset{\tau}{\longmapsto}_2, \approx_2, \vdash_2\rangle$. *We say that the encoding* $\langle [\![\cdot]\!], (\!|\cdot|\!)\rangle : \mathcal{L}_1 \to \mathcal{L}_2$ *is* semantic preserving *if it satisfies the properties below.*

1. Type Soundness: *if* $\Gamma;\emptyset;\Delta \vdash_1 P \triangleright \diamond$ *then* $(\Gamma);\emptyset;(\Delta) \vdash_2 [\![P]\!] \triangleright \diamond$, *for any* P *in* C_1.
2. Barb Preserving: *if* $\Gamma;\Delta \vdash_1 P \downarrow_n$ *then* $(\Gamma);(\Delta) \vdash_2 [\![P]\!] \Downarrow_n$.
3. Operational Correspondence: *If* $\Gamma;\emptyset;\Delta \vdash_1 P \triangleright \diamond$ *then*
   (a) *Completeness: If* $\Gamma;\Delta \vdash_1 P \overset{\tau}{\longmapsto}_1 \Delta' \vdash_1 P'$ *then* $\exists Q, \Delta''$ *s.t.*
   (i) $(\Gamma);(\Delta) \vdash_2 [\![P]\!] \Longrightarrow_2 (\Delta'') \vdash_2 Q$
   and (ii) $(\Gamma);(\Delta'') \vdash_2 Q \approx_2 (\Delta') \vdash_2 [\![P']\!]$.
   (b) *Soundness: If* $(\Gamma);(\Delta) \vdash_2 [\![P]\!] \Longrightarrow_2 (\Delta') \vdash_2 Q$ *then* $\exists P', \Delta''$ *s.t.*
   (i) $\Gamma;\Delta \vdash_1 P \overset{\tau}{\longmapsto}_1 \Delta'' \vdash_1 P'$
   and (ii) $(\Gamma);(\Delta'') \vdash_2 [\![P']\!] \approx_2 (\Delta') \vdash_2 Q$.
4. Full Abstraction: $\Gamma;\Delta \vdash_1 P \approx_1 \Delta' \vdash_1 Q$ *if and only if* $(\Gamma);(\Delta) \vdash_2 [\![P]\!] \approx_2 (\Delta') \vdash_2 [\![Q]\!]$.

Together with type preservation (Definition 13), type soundness is a distinguishing criterion in our notion of encoding. Barb preservation, related to success sensitiveness in [10], is convenient in our developments as all considered calculi have the same notion of barb. Operational correspondence, standardly divided into completeness and soundness, is also based on [10]; it relies on τ-transitions (reductions). Completeness ensures that a step of the source process is mimicked by a step of its associated encoding; soundness is its converse. Above, operational correspondence is stated in generic terms. It is worth stressing that the operational correspondence statements for our encodings are tailored to the specifics of each encoding, and so they are actually stronger than the criteria given above (see Propositions 3, 6, 10, 13 and [15] for details). Finally, following [27,32,45], we consider full abstraction as an encodability criterion: this leads to stronger encodability results.

We introduce *precise* and *minimal* encodings. While we state strong positive encodability results in terms of *precise* encodings, to prove the non-encodability result in Sect. 7.4, we appeal to the weaker *minimal* encodings.

Definition 15 (Typed Encodings: Precise and Minimal). *We say that the typed encoding* $\langle [\![\cdot]\!], (\cdot)\rangle : \mathcal{L}_1 \to \mathcal{L}_2$ *is* precise, *if it is syntax, type, and semantic preserving (Definitions 12, 13, 14). We say that the encoding is* minimal, *if it is syntax preserving (Definition 12), barb preserving (Definition 14-2), and operationally complete (Definition 14-3(a)).*

The following property will come in handy in Sect. 8:

Proposition 1. *Let* $\langle [\![\cdot]\!]^1, (\cdot)^1\rangle : \mathcal{L}_1 \to \mathcal{L}_2$ *and* $\langle [\![\cdot]\!]^2, (\cdot)^2\rangle : \mathcal{L}_2 \to \mathcal{L}_3$ *be two precise encodings. Then their composition, denoted* $\langle [\![\cdot]\!]^2 \circ [\![\cdot]\!]^1, (\cdot)^2 \circ (\cdot)^1\rangle : \mathcal{L}_1 \to \mathcal{L}_3$, *is precise.*

7 Expressiveness Results

We first present two precise encodings: (1) higher-order communication with recursion and name-passing ($\mathsf{HO}\pi$) into higher-order communication without

name-passing nor recursion (HO) (Sect. 7.1); and (2) HOπ into the first-order calculus with name-passing with recursion (π) (Sect. 7.2). We then compare these encodings (Sect. 7.3). Moreover, in Sect. 7.4 we state our impossibility result for shared/linear names. We consider the typed calculi (cf. Definition 10):

$$\mathcal{L}_{\mathsf{HO}\pi} = \langle \mathsf{HO}\pi, \mathcal{T}_1, \overset{\tau}{\longmapsto}, \approx^{\mathsf{H}}, \vdash \rangle \quad \mathcal{L}_{\mathsf{HO}} = \langle \mathsf{HO}, \mathcal{T}_2, \overset{\tau}{\longmapsto}, \approx^{\mathsf{H}}, \vdash \rangle \quad \mathcal{L}_{\pi} = \langle \pi, \mathcal{T}_3, \overset{\tau}{\longmapsto}, \approx^{\mathsf{C}}, \vdash \rangle$$

where: $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ are sets of types of HOπ, HO, and π, respectively. The typing $\vdash$ is defined in Sect. 4. The LTSs follow the intuitions given in Sect. 5.2. Moreover, $\approx^{\mathsf{H}}$ is as in Definition 9, and $\approx^{\mathsf{C}}$ is as in Definitions 8.

7.1 From HOπ to HO

HO is expressive enough to precisely encode HOπ. As discussed above, the main challenges are to encode (1) name passing and (2) recursion, for which we only use abstraction passing. As explained in Sect. 2, for (1), we pass an abstraction which enables to use the name upon application. For (2), we copy a process upon reception; passing around linear abstractions is delicate because they cannot be copied. To handle linearity, we define the following auxiliary mapping $[\![\cdot]\!]_\sigma$ from processes with free names to processes without free names (but with free variables instead):

Definition 16 (Auxiliary Mapping). *Let $|\cdot| : 2^{\mathcal{N}} \longrightarrow \mathcal{V}^\omega$ denote a map of sequences of lexicographically ordered names to sequences of variables, defined inductively as: $|\epsilon| = \epsilon$ and $|n \cdot \tilde{m}| = x_n \cdot |\tilde{m}|$. Also, let σ be a set of session names. Figure 5 defines an auxiliary mapping $[\![\cdot]\!]_\sigma : \mathsf{HO} \to \mathsf{HO}$.*

Let P be an HOπ process with $\mathsf{fn}(P) = \{n_1, \cdots, n_k\}$. Intuitively, our encoding $[\![\cdot]\!]^1_f$ exploits the abstraction $\lambda x_1, \cdots, x_k.\, [\![[\![P]\!]^1_f]\!]_\emptyset$, where $|n_j| = x_j$, for all $j \in \{1, \ldots, k\}$:

Definition 17 (Typed Encoding of HOπ into HO). *Let f be a map from process variables to sequences of name variables. The typed encoding $\langle [\![\cdot]\!]^1_f, (\cdot)^1 \rangle : \mathcal{L}_{\mathsf{HO}\pi} \to \mathcal{L}_{\mathsf{HO}}$ is given in Fig. 6. Mapping $(\cdot)^1$ on types homomorphically extends to environments Δ and Γ, with $(\Gamma \cdot X : \Delta_1)^1 = (\Gamma)^1 \cdot z_X : (S_1, \ldots, S_m, S^*) \to \diamond$ where S^* is defined as $\mu \mathsf{t}.?((S_1, \ldots, S_m, \mathsf{t}) \to \diamond)$; end provided that $\Delta_1 = \{n_i : S_i\}_{1 \le i \le m}$.*

$$[\![\mathbf{0}]\!]_\sigma \overset{\text{def}}{=} \mathbf{0} \qquad [\![n!\langle \lambda x.\, Q \rangle.P]\!]_\sigma \overset{\text{def}}{=} u!\langle \lambda x.\, [\![Q]\!]_\sigma \rangle.[\![P]\!]_\sigma \qquad [\![(\nu n)P]\!]_\sigma \overset{\text{def}}{=} (\nu n)[\![P]\!]_{\sigma \cdot n}$$

$$[\![P \mid Q]\!]_\sigma \overset{\text{def}}{=} [\![P]\!]_\sigma \mid [\![Q]\!]_\sigma \qquad [\![x n]\!]_\sigma \overset{\text{def}}{=} x u \qquad [\![(\lambda x.Q) n]\!]_\sigma \overset{\text{def}}{=} (\lambda x.[\![Q]\!]_\sigma) u$$

$$[\![n?(x).P]\!]_\sigma \overset{\text{def}}{=} u?(x).[\![P]\!]_\sigma \quad [\![n \triangleleft l.P]\!]_\sigma \overset{\text{def}}{=} u \triangleleft l.[\![P]\!]_\sigma \quad [\![n \triangleright \{l_i : P_i\}_{i \in I}]\!]_\sigma \overset{\text{def}}{=} u \triangleright \{l_i : [\![P_i]\!]_\sigma\}_{i \in I}$$

In all cases: $u = n$ if $n \in \sigma$; otherwise $u = x_n$.

Fig. 5. Auxiliary mapping used to encode HOπ into HO (Definition 16).

Types:

$$\lfloor S \rfloor^1 \stackrel{\text{def}}{=} (?(\langle\!\langle S \rangle\!\rangle^1 \multimap \diamond); \mathsf{end}) \multimap \diamond \qquad \lfloor \langle S \rangle \rfloor^1 \stackrel{\text{def}}{=} (?(\langle\langle\!\langle S \rangle\!\rangle^1 \rangle \to \diamond); \mathsf{end}) \multimap \diamond$$

$$\lfloor \langle L \rangle \rfloor^1 \stackrel{\text{def}}{=} (?(\langle\langle\!\langle L \rangle\!\rangle^1 \rangle \to \diamond); \mathsf{end}) \multimap \diamond \qquad \lfloor C \multimap \diamond \rfloor^1 \stackrel{\text{def}}{=} \langle\!\langle C \rangle\!\rangle^1 \multimap \diamond \qquad \lfloor C \to \diamond \rfloor^1 \stackrel{\text{def}}{=} \langle\!\langle C \rangle\!\rangle^1 \to \diamond$$

$$\langle\!\langle \langle S \rangle \rangle\!\rangle^1 \stackrel{\text{def}}{=} \langle\langle\!\langle S \rangle\!\rangle^1 \rangle \qquad \langle\!\langle \langle L \rangle \rangle\!\rangle^1 \stackrel{\text{def}}{=} \langle\langle\!\langle L \rangle\!\rangle^1 \rangle$$

$$\langle\!\langle !\langle U \rangle; S \rangle\!\rangle^1 \stackrel{\text{def}}{=} !\langle \lfloor U \rfloor^1 \rangle; \langle\!\langle S \rangle\!\rangle^1 \qquad \langle\!\langle ?(U); S \rangle\!\rangle^1 \stackrel{\text{def}}{=} ?(\lfloor U \rfloor^1); \langle\!\langle S \rangle\!\rangle^1$$

$$\langle\!\langle \oplus\{l_i : S_i\}_{i \in I} \rangle\!\rangle^1 \stackrel{\text{def}}{=} \oplus\{l_i : \langle\!\langle S_i \rangle\!\rangle^1\}_{i \in I} \qquad \langle\!\langle \&\{l_i : S_i\}_{i \in I} \rangle\!\rangle^1 \stackrel{\text{def}}{=} \&\{l_i : \langle\!\langle S_i \rangle\!\rangle^1\}_{i \in I}$$

$$\langle\!\langle \mathsf{t} \rangle\!\rangle^1 \stackrel{\text{def}}{=} \mathsf{t} \qquad \langle\!\langle \mu \mathsf{t}.S \rangle\!\rangle^1 \stackrel{\text{def}}{=} \mu \mathsf{t}.\langle\!\langle S \rangle\!\rangle^1 \qquad \langle\!\langle \mathsf{end} \rangle\!\rangle^1 \stackrel{\text{def}}{=} \mathsf{end}$$

Terms:

$$[\![u!\langle w \rangle.P]\!]_f^1 \stackrel{\text{def}}{=} u!\langle \lambda z.\, z?(x).(x\, w) \rangle.[\![P]\!]_f^1 \qquad [\![u?(x{:}C).Q]\!]_f^1 \stackrel{\text{def}}{=} u?(y).(\nu\, s)(y\, s \mid \overline{s}!\langle \lambda x.[\![Q]\!]_f^1 \rangle.\mathbf{0})$$

$$[\![u!\langle \lambda x.Q \rangle.P]\!]_f^1 \stackrel{\text{def}}{=} u!\langle \lambda x.[\![Q]\!]_f^1 \rangle.[\![P]\!]_f^1 \qquad [\![u?(x{:}L).P]\!]_f^1 \stackrel{\text{def}}{=} u?(x).[\![P]\!]_f^1$$

$$[\![s \triangleleft l.P]\!]_f^1 \stackrel{\text{def}}{=} s \triangleleft l.[\![P]\!]_f^1 \qquad [\![s \triangleright \{l_i{:}P_i\}_{i \in I}]\!]_f^1 \stackrel{\text{def}}{=} s \triangleright \{l_i : [\![P_i]\!]_f^1\}_{i \in I}$$

$$[\![\mathbf{0}]\!]_f^1 \stackrel{\text{def}}{=} \mathbf{0} \qquad [\![(\nu n)P]\!]_f^1 \stackrel{\text{def}}{=} (\nu n)[\![P]\!]_f^1$$

$$[\![x\, u]\!]_f^1 \stackrel{\text{def}}{=} x\, u \qquad [\![(\lambda x.Q)\, u]\!]_f^1 \stackrel{\text{def}}{=} (\lambda x.[\![Q]\!]_f^1)\, u$$

$$[\![P \mid Q]\!]_f^1 \stackrel{\text{def}}{=} [\![P]\!]_f^1 \mid [\![Q]\!]_f^1$$

$$[\![\mu X.P]\!]_f^1 \stackrel{\text{def}}{=} (\nu\, s)(\overline{s}!\langle \lambda(\|\tilde{n}\|, y).\, y?(z_X).\llbracket [\![P]\!]_{f,\{X \to \tilde{n}\}}^1 \rrbracket_{\emptyset} \rangle.\mathbf{0} \mid s?(z_X).[\![P]\!]_{f,\{X \to \tilde{n}\}}^1) \quad (\tilde{n} = \mathsf{fn}(P))$$

$$[\![X]\!]_f^1 \stackrel{\text{def}}{=} (\nu\, s)(z_X\, (\tilde{n}, s) \mid \overline{s}!\langle \lambda(\|\tilde{n}\|, y).\, z_X\, (\|\tilde{n}\|, y) \rangle.\mathbf{0}) \quad (\tilde{n} = f(X))$$

Above $\mathsf{fn}(P)$ denotes a lexicographically ordered sequence of free names in P. The input bound variable x is annotated by a type to distinguish first- and higher-order cases.

Fig. 6. Encoding of HOπ into HO (Definition 17).

Note that Δ in $X : \Delta$ is mapped to a non-tail recursive session type with variable z_X. Non-tail recursive session types were studied in [2,3]; to our knowledge, this is the first application in the context of higher-order session types. For simplicity, we use polyadic name abstractions. A precise encoding of polyadicity into HO is given in Sect. 8.

Key elements in Fig. 6 are encodings of *name passing* ($[\![u!\langle w \rangle.P]\!]_f^1$ and $[\![u?(x).P]\!]_f^1$) and *recursion* ($[\![\mu X.P]\!]_f^1$ and $[\![X]\!]_f^1$). As motivated in Sect. 2, a name w is passed as an input-guarded abstraction; on the receiver side, the encoding i) receives the abstraction; ii) applies to it a fresh endpoint s; iii) uses the dual endpoint $\overline{s}$ to send the continuation P as an abstraction. Thus, name substitution is achieved via name application. As for recursion, to encode $\mu X.P$ we first record a mapping from recursive variable X to process variable z_X. Then, using $\llbracket \cdot \rrbracket_\sigma$ in Definition 16, we encode the recursion body P as a name abstraction in which free names of P are converted into name variables. (Notice that P is first encoded into HO and then transformed using mapping $\llbracket \cdot \rrbracket_\sigma$.) Subsequently, this higher-order value is embedded in an input-guarded "duplicator" process. We encode X in such a way that it simulates recursion unfolding by invoking the duplicator in a by-need fashion. That is, upon reception, the HO abstraction encoding P is duplicated: one copy is used to reconstitute the original recursion

body P (through the application of $\mathtt{fn}(P)$); another copy is used to re-invoke the duplicator when needed. We illustrate the encoding by means of an example.

Example 2 (The Encoding $[\![\cdot]\!]^1_f$ At Work). Let $P = \mu X.a!\langle m\rangle.X$ be an $\mathsf{HO}\pi$ process. Its encoding into HO is given next; notice that $f = \emptyset$ and $f' = X \to x_a x_m$.

$$[\![P]\!]^1_f = (\nu\, s_1)(s_1?(x).[\![a!\langle m\rangle.X]\!]^1_{f'} \mid \overline{s_1}!\langle\lambda(x_a, x_m, z).\, z?(x).[\![\![a!\langle m\rangle.X]\!]^1_{f'}]\!]_\emptyset\rangle.\mathbf{0})$$

$$[\![a!\langle m\rangle.X]\!]^1_{f'} = a!\langle\lambda z.\, z?(x).(x\, m)\rangle.(\nu\, s_2)(x\,(a, m, s_2) \mid \overline{s_2}!\langle\lambda(x_a, x_m, z).\, x\,(x_a, x_m, z)\rangle.\mathbf{0})$$

$$[\![\![a!\langle m\rangle.X]\!]^1_{f'}]\!]_\emptyset = x_a!\langle\lambda z.\, z?(x).(x\, x_m)\rangle.(\nu\, s_2)(x\,(x_a, x_m, s_2) \mid \overline{s_2}!\langle\lambda(x_a, x_m, z).\, x\,(x_a, x_m, z)\rangle.\mathbf{0})$$

That is, by writing V to denote the process

$$\lambda(x_a, x_m, z).\, z?(x).x_a!\langle\lambda z.\, z?(x).(x\, x_m)\rangle.(\nu\, s_2)(x\,(x_a, x_m, s_2) \mid \overline{s_2}!\langle\lambda(x_a, x_m, z).\, x\,(x_a, x_m, z)\rangle.\mathbf{0})$$

we would have

$$[\![P]\!]^1_f = (\nu\, s_1)(s_1?(x).a!\langle\lambda z.\, z?(x).(x\, m)\rangle.(\nu\, s_2)(x\,(a, m, s_2) \mid \overline{s_2}!\langle\lambda(x_a, x_m, z).\, x\,(x_a, x_m, z)\rangle.\mathbf{0}) \mid \overline{s_1}!\langle V\rangle.\mathbf{0})$$

Next we illustrate the behaviour of $[\![P]\!]^1_f$; below ℓ stands for $a!\langle\lambda z.\, z?(x).(x\, m)\rangle$.

$$\begin{aligned}
[\![P]\!]^1_f &\equiv (\nu\, s_1)(\overline{s_1}!\langle V\rangle.\mathbf{0} \mid s_1?(x).a!\langle\lambda z.\, z?(x).(x\, m)\rangle.(\nu\, s_2)(\overline{s_2}!\langle\lambda(x_a, x_m, z).\\
&\qquad x\,(x_a, x_m, z)\rangle.\mathbf{0}) \mid x\,(a, m, s_2))\\
&\overset{\tau}{\longmapsto} a!\langle\lambda z.\, z?(x).(x\, m)\rangle.(\nu\, s_2)(\overline{s_2}!\langle V\rangle.\mathbf{0} \mid s_2?(x).a!\langle\lambda z.\, z?(x).(x\, m)\rangle.\\
&\qquad (\nu\, s_3)(\overline{s_3}!\langle\lambda(x_a, x_m, z).\, x\,(x_a, x_m, z)\rangle.\mathbf{0}) \mid x\,(a, m, s_3))\\
&\equiv_\alpha a!\langle\lambda z.\, z?(x).(x\, m)\rangle.(\nu\, s_1)(\overline{s_1}!\langle V\rangle.\mathbf{0} \mid s_1?(x).a!\langle\lambda z.\, z?(x).(x\, m)\rangle.\\
&\qquad (\nu\, s_2)(\overline{s_2}!\langle\lambda(x_a, x_m, z).\, x\,(x_a, x_m, z)\rangle.\mathbf{0}) \mid x\,(a, m, s_2))\\
&\equiv a!\langle\lambda z.\, z?(x).(x\, m)\rangle.[\![\mu X.a!\langle m\rangle.X]\!]^1_f \overset{\ell}{\longmapsto} [\![\mu X.a!\langle m\rangle.X]\!]^1_f.
\end{aligned}$$

We now describe the properties of the encoding. Directly from Fig. 6 we may state:

Proposition 2 ($\mathsf{HO}\pi$ into HO: Type Preservation). *The encoding from $\mathcal{L}_{\mathsf{HO}\pi}$ into $\mathcal{L}_{\mathsf{HO}}$ (cf. Definition 17) is type preserving.*

Now, we state operational correspondence with respect to reductions; the full statement (and proof) can be found in [15].

Proposition 3 ($\mathsf{HO}\pi$ into HO: Operational Correspondence - Excerpt). *Let P be an $\mathsf{HO}\pi$ process such that $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$.*

1. *Completeness: Suppose $\Gamma;\Delta \vdash P \overset{\tau}{\longmapsto} \Delta' \vdash P'$. Then we have:*
   (a) If $P' \equiv (\nu\,\tilde{m})(P_1 \mid P_2\{^m/_x\})$ then $\exists R$ s.t.

$$(\Gamma)^1;(\Delta)^1 \vdash [\![P]\!]^1_f \overset{\tau}{\longmapsto} (\Delta)^1 \vdash (\nu\,\tilde{m})([\![P_1]\!]^1_f \mid R), \text{ and}$$

$$(\Gamma)^1;(\Delta)^1 \vdash (\nu\,\tilde{m})([\![P_1]\!]^1_f \mid R) \overset{\tau_\beta}{\longmapsto}\overset{\tau_s}{\longmapsto}\overset{\tau_\beta}{\longmapsto} (\Delta)^1 \vdash (\nu\,\tilde{m})([\![P_1]\!]^1_f \mid [\![P_2]\!]^1_f\{^m/_x\}).$$

(b) *If* $P' \equiv (\nu\,\tilde{m})(P_1 \mid P_2\{^{\lambda y.\,Q}/_x\})$ *then*
$(\Gamma)^1; (\Delta)^1 \vdash [\![P]\!]^1_f \overset{\tau}{\longmapsto} (\Delta_1)^1 \vdash (\nu\,\tilde{m})([\![P_1]\!]^1_f \mid [\![P_2]\!]^1_f\{^{\lambda y.\,[\![Q]\!]^1_\emptyset}/_x\})$.

(c) *If* $P' \not\equiv (\nu\,\tilde{m})(P_1 \mid P_2\{^m/_x\}) \wedge P' \not\equiv (\nu\,\tilde{m})(P_1 \mid P_2\{^{\lambda y.\,Q}/_x\})$ *then*
$(\Gamma)^1; (\Delta)^1 \vdash [\![P]\!]^1_f \overset{\tau}{\longmapsto} (\Delta'_1)^1 \vdash [\![P']\!]^1_f$.

2. *Soundness: Suppose* $(\Gamma)^1; (\Delta)^1 \vdash [\![P]\!]^1_f \overset{\tau}{\longmapsto} (\Delta')^1 \vdash Q$. *Then* $\Delta' = \Delta$ *and either*

(a) $\exists P'$ *s.t.* $\Gamma; \Delta \vdash P \overset{\tau}{\longmapsto} \Delta \vdash P'$, *and* $Q = [\![P']\!]^1_f$.

(b) $\exists P_1, P_2, x, m, Q'$ *s.t.* $\Gamma; \Delta \vdash P \overset{\tau}{\longmapsto} \Delta \vdash (\nu\,\tilde{m})(P_1 \mid P_2\{^m/_x\})$, *and*
$(\Gamma)^1; (\Delta)^1 \vdash Q \overset{\tau_\beta}{\longmapsto}\overset{\tau_s}{\longmapsto}\overset{\tau_\beta}{\longmapsto} (\Delta)^1 \vdash [\![P_1]\!]^1_f \mid [\![P_2\{^m/_x\}]\!]^1_f$

Observe how we can explicitly distinguish the role of finite, deterministic reductions ($\overset{\tau_s}{\longmapsto}$ and $\overset{\tau_\beta}{\longmapsto}$, defined in Notation 1) in both soundness and completeness statements.

The typed operational correspondence given above is an important component in the proof of *full abstraction*, which we state next.

Proposition 4 (HOπ into HO: Full Abstraction). *Let* P_1, Q_1 *be* HOπ *processes.*
$\Gamma; \Delta_1 \vdash P_1 \approx^H \Delta_2 \vdash Q_1$ *if and only if* $(\Gamma)^1; (\Delta_1)^1 \vdash [\![P_1]\!]^1_f \approx^H (\Delta_2)^1 \vdash [\![Q_1]\!]^1_f$.

We may state the main result of this section. See [15] for details.

Theorem 3 (Precise Encoding of HOπ into HO). *The encoding from* $\mathcal{L}_{\mathsf{HO}\pi}$ *into* $\mathcal{L}_{\mathsf{HO}}$ *(cf. Definition 17) is precise.*

7.2 From HOπ to π

We now discuss the precise encodability of HOπ into π; the non trivial issue is encoding higher-order communication, which is present in HOπ but not in π. We closely follow Sangiorgi's encoding [33,36], which represents the exchange of a process with the exchange of a fresh *trigger name*. Trigger names may then be used to activate copies of the process, which becomes a persistent resource represented by an input-guarded replication. We cast this strategy in the setting of session-typed communications. In the presence of session names (which are linear and cannot be replicated), our approach uses replicated names as triggers for shared resources and non-replicated names for linear resources (cf. $[\![u!\langle\lambda x.\,Q\rangle.P]\!]^2$).

Definition 18 (Typed Encoding of HOπ into π). *The typed encoding* $\langle [\![\cdot]\!]^2, (\cdot)^2 \rangle : \mathcal{L}_{\mathsf{HO}\pi} \to \mathcal{L}_\pi$ *is defined in Fig. 7.*

Observe how $[\![(\lambda x.\,P)\,u]\!]^2$ naturally induces a name substitution. We describe key properties of this encoding. First, type preservation and operational correspondence:

Proposition 5 (HOπ into π: Type Preservation). *The encoding from* $\mathcal{L}_{\mathsf{HO}\pi}$ *into* $\mathcal{L}_\pi$ *(cf. Definition 18) is type preserving.*

Types:

$$(\!|!\langle S \multimap \diamond\rangle; S_1|\!)^2 \stackrel{\text{def}}{=} !\langle\langle?((\!|S|\!)^2); \text{end}\rangle\rangle; (\!|S_1|\!)^2 \qquad (\!|?(S \multimap \diamond); S_1|\!)^2 \stackrel{\text{def}}{=} ?(\langle?((\!|S|\!)^2); \text{end}\rangle); (\!|S_1|\!)^2$$

Terms:

$$[\![u!\langle\lambda x.Q\rangle.P]\!]^2 \stackrel{\text{def}}{=} \begin{cases} (\nu a)(u!\langle a\rangle.([\![P]\!]^2 \mid *\, a?(y).y?(x).[\![Q]\!]^2)) & (s \notin \text{fn}(Q)) \\ (\nu a)(u!\langle a\rangle.([\![P]\!]^2 \mid a?(y).y?(x).[\![Q]\!]^2)) & (\text{otherwise}) \end{cases}$$

$$[\![u?(x).P]\!]^2 \stackrel{\text{def}}{=} u?(x).[\![P]\!]^2$$

$$[\![x\,u]\!]^2 \stackrel{\text{def}}{=} (\nu s)(x!\langle s\rangle.\overline{s}!\langle u\rangle.\mathbf{0})$$

$$[\![(\lambda x.P)\,u]\!]^2 \stackrel{\text{def}}{=} (\nu s)(s?(x).[\![P]\!]^2 \mid \overline{s}!\langle u\rangle.\mathbf{0})$$

Notice: $*\,P$ means $\mu X.(P \mid X)$. Elided mappings are homomorphic.

Fig. 7. Encoding of HOπ into π (Definition 18).

Proposition 6 (HOπ into π: Operational Correspondence - Excerpt). *Let P be an HOπ process such that $\Gamma; \emptyset; \Delta \vdash P \triangleright \diamond$.*

1. *Completeness: Suppose $\Gamma; \Delta \vdash P \overset{\ell}{\longmapsto} \Delta' \vdash P'$. Then either:*
   (a) *If $\ell = \tau$ then one of the following holds:*
   - $(\Gamma)^2; (\Delta)^2 \vdash [\![P]\!]^2 \overset{\tau}{\longmapsto} (\Delta')^2 \vdash (\nu\, \tilde{m})([\![\,]\!]_1^P 2 \mid (\nu\, a)([\![\,]\!]_2^P 2\{^a/_x\} \mid *\, a?(y).y?(x).[\![Q]\!]^2$ *for some P_1, P_2, Q;*
   - $(\Gamma)^2; (\Delta)^2 \vdash [\![P]\!]^2 \overset{\tau}{\longmapsto} (\Delta')^2 \vdash (\nu\, \tilde{m})([\![\,]\!]_1^P 2 \mid (\nu\, s)([\![\,]\!]_2^P 2\{^{\overline{s}}/_x\} \mid s?(y).y?(x).[\![Q]\!]^2$, *for some P_1, P_2, Q;*
   - $(\Gamma)^2; (\Delta)^2 \vdash [\![P]\!]^2 \overset{\tau}{\longmapsto} (\Delta')^2 \vdash [\![P']\!]^2$

   (b) *If $\ell = \tau_\beta$ then $(\Gamma)^2; (\Delta)^2 \vdash [\![P]\!]^2 \overset{\tau_s}{\longmapsto} (\Delta')^2 \vdash [\![P']\!]^2$.*
2. *Suppose $(\Gamma)^2; (\Delta)^2 \vdash [\![P]\!]^2 \overset{\tau}{\longmapsto} (\Delta')^2 \vdash R$.*
   Then $\exists P'$ such that $P \overset{\tau}{\longmapsto} P'$ and $(\Gamma)^2; (\Delta')^2 \vdash [\![P']\!]^2 \approx^H (\Delta')^2 \vdash R$.

Exploiting the above properties (type preservation, typed operational correspondence), we can show that our typed encoding is fully abstract and precise.

Proposition 7 (HOπ to π: Full Abstraction). *Let P_1, Q_1 be HOπ processes. $\Gamma; \Delta_1 \vdash P_1 \approx^H \Delta_2 \vdash Q_1$ if and only if $(\Gamma)^2; (\Delta_1)^2 \vdash [\![P_1]\!]^2 \approx^C (\Delta_2)^2 \vdash [\![Q_1]\!]^2$.*

Theorem 4 (Precise Encoding of HOπ into π). *The encoding from $\mathcal{L}_{\text{HO}\pi}$ into $\mathcal{L}_\pi$ (cf. Definition 18) is precise.*

7.3 Comparing Precise Encodings

The precise encodings in Sects. 7.1 and 7.2 confirm that HO and π constitute two important sources of expressiveness in HOπ. This naturally begs the question: which of the two sub-calculi is more tightly related to HOπ? We argue, both empirically and formally, that when compared to π, HO is more economical and satisfies tighter correspondences.

Empirical Comparison: Reduction Steps. We first contrast the way in which (a) the encoding from HOπ to HO (Sect. 7.1) translates processes with name passing; (b) the encoding from HOπ to π (Sect. 7.2) translates processes with abstraction passing. Consider the HOπ processes:

$$P_1 = s!\langle a\rangle.\mathbf{0} \mid \overline{s}?(x).(x!\langle s_1\rangle.\mathbf{0} \mid \ \ldots \ \mid x!\langle s_n\rangle.\mathbf{0}) \qquad P_2 = s!\langle \lambda x.\, P\rangle.\mathbf{0} \mid \overline{s}?(x).(x\, s_1 \mid \ \ldots \ \mid x\, s_n)$$

P_1 features *pure* name passing (no abstraction-passing), whereas P_2 involves *pure* abstraction passing (no name passing). In both cases, the intended communication on s leads to n usages of the communication object (name a in P_1, abstraction $\lambda x.\, P$ in P_2). Consider now the reduction steps from P_1 and P_2:

$$P_1 \overset{\tau}{\longmapsto} a!\langle s_1\rangle.\mathbf{0} \mid \ \ldots \ \mid a!\langle s_n\rangle.\mathbf{0}$$
$$P_2 \overset{\tau}{\longmapsto} (\lambda x.\, P)\, s_1 \mid \ \ldots \ \mid (\lambda x.\, P)\, s_n \quad \underbrace{\overset{\tau_\beta}{\longmapsto}\overset{\tau_\beta}{\longmapsto} \cdots \overset{\tau_\beta}{\longmapsto}}_{n} \quad P\{s_1/x\} \mid \ \ldots \ \mid P\{s_1/x\}$$

By considering the encoding of P_1 into HO we obtain:

$$\begin{aligned}
[\![P_1]\!]^1_f = \quad & s!\langle \lambda z.\, z?(y).y\, a\rangle.\mathbf{0} \mid \\
& \overline{s}?(x).(\nu\, t)(x\, t \mid \overline{t}!\langle \lambda x.\, (x!\langle \lambda z.\, z?(y).y\, s_1\rangle.\mathbf{0} \mid \ \ldots \ \mid x!\langle \lambda z.\, z?(y).y\, s_n\rangle.\mathbf{0})\rangle.\mathbf{0}) \\
\overset{\tau_s}{\longmapsto}\overset{\tau_\beta}{\longmapsto} \ & (\nu\, t)(t?(y).y\, a \mid \overline{t}!\langle \lambda x.\, (x!\langle \lambda z.\, z?(y).y\, s_1\rangle.\mathbf{0} \mid \ \ldots \ \mid x!\langle \lambda z.\, z?(y).y\, s_n\rangle.\mathbf{0})\rangle.\mathbf{0}) \\
\overset{\tau_s}{\longmapsto}\overset{\tau_\beta}{\longmapsto} \ & a!\langle \lambda z.\, z?(y).y\, s_1\rangle.\mathbf{0} \mid \ \ldots \ \mid a!\langle \lambda z.\, z?(y).y\, s_n\rangle.\mathbf{0}
\end{aligned}$$

Now, we encode P_2 into π:

$$\begin{aligned}
[\![P_2]\!]^2 \quad = \quad & (\nu\, b)(s!\langle b\rangle.\mathbf{0} \mid \ * \ b?(y).y?(x).P) \mid \\
& \overline{s}?(x).((\nu\, s)(x!\langle s\rangle.\overline{s}!\langle s_1\rangle.\mathbf{0}) \mid \ \ldots \ \mid (\nu\, s)(x!\langle s\rangle.\overline{s}!\langle s_n\rangle.\mathbf{0})) \\
\overset{\tau_s}{\longmapsto}\overset{\tau_s}{\longmapsto}\overset{\tau_s}{\longmapsto} \ & (\nu\, b)(*\, b?(y).y?(x).P \mid P\{s_1/x\} \mid \ \ldots \ \mid (\nu\, s)(b!\langle s\rangle.\overline{s}!\langle s_n\rangle.\mathbf{0})) \\
\Longmapsto_{2*(n-1)} \ & (\nu\, b)(*\, b?(y).y?(x).P \mid P\{s_1/x\} \mid \ \ldots \ \mid P\{s_n/x\})
\end{aligned}$$

Clearly, encoding P_1 into HO is more economical than encoding P_2 into π. Not only moving to a pure higher-order setting requires less reduction steps than in the first-order concurrency of π; in the presence of shared names, moving to a first-order setting brings the need of setting up and handling replicated processes which will eventually lead to garbage (stuck) processes (cf. $*\, b?(y).y?(x).P$ above). In contrast, the mechanism present in HO works efficiently regardless of the linear or shared properties of the name that is "packed" into the abstraction. The use of β-transitions guarantees local synchronizations, which are arguably more economical than point-to-point, session synchronizations.

It is useful to move our comparison to a purely linear setting. Consider processes:

$$Q_1 = s'!\langle s\rangle.\mathbf{0} \mid \overline{s'}?(x).x!\langle a\rangle.\mathbf{0} \overset{\tau}{\longmapsto} s!\langle a\rangle.\mathbf{0} \qquad Q_2 = s!\langle \lambda x.\, P\rangle.\mathbf{0} \mid \overline{s}?(x).x\, a \overset{\tau}{\longmapsto}\overset{\tau}{\longmapsto} P\{a/x\}$$

Q_1 is a π process; Q_2 is an HO processs. If we consider their encodings into HO and π, respectively, we obtain:

$$\begin{array}{rcl}
[\![Q_1]\!]^1_f & = & s'!\langle\lambda z.\, z?(y).y\, s\rangle.\mathbf{0} \mid \overline{s'}?(x).(\nu\, t)(x\, t \mid \bar{t}!\langle\lambda x.\, x!\langle\lambda z.\, z?(y).y\, a\rangle.\mathbf{0}\rangle.\mathbf{0}) \\
& \xrightarrow{\tau_s}\xrightarrow{\tau_\beta} & (\nu\, t)(t?(y).y\, s \mid \bar{t}!\langle\lambda x.\, x!\langle\lambda z.\, z?(y).y\, a\rangle.\mathbf{0}\rangle.\mathbf{0}) \\
& \xrightarrow{\tau_s} & \lambda x.\, x!\langle\lambda z.\, z?(y).y\, a\rangle.\mathbf{0}\; s \xrightarrow{\tau_\beta} s!\langle\lambda z.\, z?(y).y\, a\rangle.\mathbf{0} \\
[\![Q_2]\!]^2 & = & (\nu\, t)(s!\langle t\rangle.\mathbf{0} \mid \bar{t}?(y).y?(x).P) \mid \overline{s}?(x).(\nu\, s)(x!\langle s\rangle.\overline{s}!\langle a\rangle.\mathbf{0}) \\
& \xrightarrow{\tau_s}\xrightarrow{\tau_s} & (\nu\, s)(s?(x).P \mid \overline{s}!\langle a\rangle.\mathbf{0}) \xrightarrow{\tau_s} P\{a/x\}
\end{array}$$

In this case, the encoding $[\![\cdot]\!]^2$ is more efficient, as it induces less reduction steps. Therefore, considering a fragment of HOπ without shared communications (linearity only) has consequences in terms of reduction steps. Notice that we prove that linear communications do not suffice to encode shared communications (Sect. 7.4).

Formal Comparison: Labelled Transition Correspondence. We now formally establish differences between $[\![\cdot]\!]^1_f$ and $[\![\cdot]\!]^2$. To this end, we introduce an extra encodability criterion: a form of operational correspondence for *visible actions*. Below we write ℓ_1, ℓ_2 to denote actions different from τ and $\overset{\ell}{\longmapsto}$ to denote an LTS. As actions from different calculi may be different, we also consider a mapping on action labels, denoted $\{\cdot\}$:

Definition 19 (Labelled Correspondence / Tight Encodings). *Consider typed calculi $\mathcal{L}_1$ and $\mathcal{L}_2$, defined as $\mathcal{L}_1 = \langle \mathsf{C}_1, \mathcal{T}_1, \overset{\ell_1}{\longmapsto}_1, \approx_1, \vdash_1\rangle$ and $\mathcal{L}_2 = \langle \mathsf{C}_2, \mathcal{T}_2, \overset{\ell_2}{\longmapsto}_2, \approx_2, \vdash_2\rangle$. The encoding $\langle [\![\cdot]\!], (\cdot)\rangle : \mathcal{L}_1 \to \mathcal{L}_2$ satisfies* labelled operational correspondence *if it satisfies:*

1. *If $\Gamma; \Delta \vdash_1 P \overset{\ell_1}{\longmapsto}_1 \Delta' \vdash_1 P'$ then $\exists Q, \Delta'', \ell_2$ s.t. (i) $(\Gamma);(\Delta) \vdash_2 [\![P]\!] \overset{\ell_2}{\Longmapsto}_2 (\Delta'') \vdash_2 Q$; (ii) $\ell_2 = \{\ell_1\}$; (iii) $(\Gamma);(\Delta'') \vdash_2 Q \approx_2 (\Delta') \vdash_2 [\![P']\!]$.*
2. *If $(\Gamma);(\Delta) \vdash_2 [\![P]\!] \overset{\ell_2}{\Longmapsto}_2 (\Delta') \vdash_2 Q$ then $\exists P', \Delta'', \ell_1$ s.t. (i) $\Gamma; \Delta \vdash_1 P \overset{\ell_1}{\longmapsto}_1 \Delta'' \vdash_1 P'$; (ii) $\ell_2 = \{\ell_1\}$; (iii) $(\Gamma);(\Delta'') \vdash_2 [\![P']\!] \approx_2 (\Delta') \vdash_2 Q$.*

A tight encoding *is a typed encoding which is precise (Definition 15) and that also satisfies labelled operational correspondence as above.*

We may formally state that HOπ and HO are more closely related than HOπ and π:

Theorem 5 (HO Tightly Encodes HOπ). *While the encoding of* HOπ *into* HO *(Definition 17) is tight, the encoding of* HOπ *into* π *(Definition 18) is not tight.*

To substantiate the above claim, we show that the encoding $[\![\cdot]\!]^1_f$ enjoys labelled operational correspondence, whereas $[\![\cdot]\!]^2$ does not. Consider the following mapping:

$$\{(\nu\, m\tilde{n}_1)n!\langle m\rangle\}^1 \stackrel{\text{def}}{=} (\nu\, m\tilde{n}_1)n!\langle\lambda z.\ z?(x).x\, m\rangle \qquad \{n?\langle m\rangle\}^1 \stackrel{\text{def}}{=} n?\langle\lambda z.\ z?(x).x\, m\rangle$$
$$\{(\nu\, \tilde{m})n!\langle\lambda x.\, P\rangle\}^1 \stackrel{\text{def}}{=} (\nu\, \tilde{m})n!\langle\lambda x.\, [\![P]\!]^1_\emptyset\rangle \qquad \{n?\langle\lambda x.\, P\rangle\}^1 \stackrel{\text{def}}{=} n?\langle\lambda x.\, [\![P]\!]^1_\emptyset\rangle$$
$$\{n \oplus l\}^1 \stackrel{\text{def}}{=} n \oplus l \qquad \{n\&l\}^1 \stackrel{\text{def}}{=} n\&l$$

Then the following result, a complement of Proposition 3, holds:

Proposition 8 (Labelled Transition Correspondence, HOπ into HO). *Let P be an HOπ process. If $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$ then:*

1. *Suppose $\Gamma;\Delta \vdash P \overset{\ell_1}{\longmapsto} \Delta' \vdash P'$. Then we have:*
   (a) *If $\ell_1 \in \{(\nu\, \tilde{m})n!\langle m\rangle, (\nu\, \tilde{m})n!\langle\lambda x.\, Q\rangle, s \oplus l, s\&l\}$ then $\exists \ell_2$ s.t.*
   $(\Gamma)^1;(\Delta)^1 \vdash [\![P]\!]^1_f \overset{\ell_2}{\longmapsto} (\Delta')^1 \vdash [\![P']\!]^1_f$ *and* $\ell_2 = \{\ell_1\}^1$.
   (b) *If $\ell_1 = n?\langle\lambda y.\, Q\rangle$ and $P' = P_0\{^{\lambda y.\, Q}/_x\}$ then $\exists \ell_2$ s.t.*
   $(\Gamma)^1;(\Delta)^1 \vdash [\![P]\!]^1_f \overset{\ell_2}{\longmapsto} (\Delta')^1 \vdash [\![P_0]\!]^1_f\{^{\lambda y.\, [\![Q]\!]^1_\emptyset}/_x\}$ *and* $\ell_2 = \{\ell_1\}^1$.
   (c) *If $\ell_1 = n?\langle m\rangle$ and $P' = P_0\{^m/_x\}$ then $\exists \ell_2, R$ s.t. $(\Gamma)^1;(\Delta)^1 \vdash [\![P]\!]^1_f \overset{\ell_2}{\longmapsto} (\Delta')^1 \vdash R$,*
   with $\ell_2 = \{\ell_1\}^1$, and $(\Gamma)^1;(\Delta')^1 \vdash R \overset{\tau_\beta}{\longmapsto}\overset{\tau_s}{\longmapsto}\overset{\tau_\beta}{\longmapsto} (\Delta')^1 \vdash [\![P_0]\!]^1_f\{^m/_x\}$.
2. *Suppose $(\Gamma)^1;(\Delta)^1 \vdash [\![P]\!]^1_f \overset{\ell_2}{\longmapsto} (\Delta')^1 \vdash Q$. Then we have:*
   (a) *If $\ell_2 \in \{(\nu\, \tilde{m})n!\langle\lambda z.\ z?(x).(x\, m)\rangle, (\nu\, \tilde{m})n!\langle\lambda x.R\rangle, s \oplus l, s\&l\}$ then $\exists \ell_1, P'$ s.t.*
   $\Gamma;\Delta \vdash P \overset{\ell_1}{\longmapsto} \Delta' \vdash P'$, $\ell_1 = \{\ell_2\}^1$, *and* $Q = [\![P']\!]^1_f$.
   (b) *If $\ell_2 = n?\langle\lambda y.R\rangle$ then either:*
   (i) *$\exists \ell_1, x, P', P''$ s.t.*
   $\Gamma;\Delta \vdash P \overset{\ell_1}{\longmapsto} \Delta' \vdash P'\{^{\lambda y.\, P''}/_x\}$, $\ell_1 = \{\ell_2\}^1$, $[\![P'']\!]^1_\emptyset = R$, *and* $Q = [\![P']\!]^1_f$.
   (ii) *$R \equiv y?(x).(x\, m)$ and $\exists \ell_1, z, P'$ s.t. $\Gamma;\Delta \vdash P \overset{\ell_1}{\longmapsto} \Delta' \vdash P'\{^m/_z\}$,*
   $\ell_1 = \{\ell_2\}^1$, *and* $(\Gamma)^1;(\Delta')^1 \vdash Q \overset{\tau_\beta}{\longmapsto}\overset{\tau_s}{\longmapsto}\overset{\tau_\beta}{\longmapsto} (\Delta'')^1 \vdash [\![P'\{^m/_z\}]\!]^1_f$

The analog of Proposition 8 does not hold for the encoding of HOπ into π. Consider the HOπ process:

$$\Gamma;\emptyset;\Delta \vdash s!\langle\lambda x.\, P\rangle.\mathbf{0} \triangleright \diamond \overset{s!\langle\lambda x.\, P\rangle}{\longmapsto} \emptyset \vdash \mathbf{0} \not\longmapsto$$

with $\lambda x.\, P$ being a linear value. We translate it into a π process:

$$(\Gamma)^2;\emptyset;(\Delta)^2 \vdash (\nu\, a)(s!\langle a\rangle.\mathbf{0} \mid a?(y).y?(x).P) \triangleright \diamond \overset{s!\langle a\rangle}{\longmapsto} \Delta' \vdash a?(y).y?(x).P \triangleright \diamond \overset{a?\langle V\rangle}{\longmapsto} \ldots$$

The resulting processes have a mismatch both in the typing environment ($\Delta' \neq (\emptyset)^2$) and in the actions that they can subsequently observe: the first process cannot perform any action, while the second process can perform actions of the encoding of $\lambda x.\, P$.

7.4 A Negative Result

As most session calculi, $\mathsf{HO}\pi$ includes communication on both shared and linear names. The former enables non determinism, unrestricted behaviour; the latter allows to represent deterministic, linear communication structures. The expressiveness of shared names is also illustrated by our encoding from $\mathsf{HO}\pi$ into π (Fig. 7). This result begs the question: can we represent shared name interaction using session name interaction? It turns out that shared names add expressiveness to $\mathsf{HO}\pi$: we prove the non existence of a minimal encoding (cf. Definition 15) of shared name interaction into linear interaction.

Theorem 6. *There is no minimal encoding from* π *to* $\mathsf{HO}\pi^{-\mathsf{sh}}$. *Hence, for any* $\mathsf{C}_1, \mathsf{C}_2 \in \{\mathsf{HO}\pi, \mathsf{HO}, \pi\}$, *there is no minimal encoding from* $\mathcal{L}_{\mathsf{C}_1}$ *into* $\mathcal{L}_{\mathsf{C}_2^{-\mathsf{sh}}}$.

By Definitions 17 and 18 and Propositions 3 and 4, we have:

Corollary 1. *Let* $\mathsf{C}_1, \mathsf{C}_2 \in \{\mathsf{HO}\pi, \mathsf{HO}, \pi\}$. *There is a precise encoding of* $\mathcal{L}_{\mathsf{C}_1^{-\mathsf{sh}}}$ *in* $\mathcal{L}_{\mathsf{C}_2^{-\mathsf{sh}}}$.

8 Extensions: Higher-Order Abstractions and Polyadicity

Here we extend $\mathsf{HO}\pi$ in two directions: (i) $\mathsf{HO}\pi^+$ extends $\mathsf{HO}\pi$ with higher-order applications/abstractions; (ii) $\mathsf{HO}\widetilde{\pi}$ extends $\mathsf{HO}\pi$ with polyadicity. In both cases, we detail the required modifications in syntax and types. Using encoding composability (Proposition 1), the two extensions may be combined into $\mathsf{HO}\widetilde{\pi}^+$: the polyadic extension of $\mathsf{HO}\pi^+$.

$\mathsf{HO}\pi$ with Higher-Order Abstractions ($\mathsf{HO}\pi^+$) and with Polyadicity ($\mathsf{HO}\widetilde{\pi}$). We first introduce $\mathsf{HO}\pi^+$, the extension of $\mathsf{HO}\pi$ with higher-order abstractions and applications. This is the calculus that we studied in [14]. The syntax of $\mathsf{HO}\pi^+$ is obtained from Fig. 2 by extending $V\,u$ to $V\,W$, where W is a higher-order value. As for the reduction semantics, we keep the rules in Fig. 3, except for [App] which is replaced by

$$(\lambda x.\, P)\, V \longrightarrow P\{^{V}/_{x}\}$$

The syntax of types is modified as follows: $L ::= U{\to}\diamond \mid U{\multimap}\diamond$. These types can be easily accommodated in the type system in Sect. 4: we replace C by U in [Abs] and C by U' in [App]. Subject reduction (Theorem 1) holds for $\mathsf{HO}\pi^+$ (cf. [14])

The calculus $\mathsf{HO}\widetilde{\pi}$ extends $\mathsf{HO}\pi$ with polyadic name passing $\tilde{n}$ and $\lambda\tilde{x}.\,Q$ in the syntax of values V. The operational semantics is kept unchanged, with the expected use of the simultaneous substitution $\{\tilde{V}/\tilde{x}\}$. The type syntax is extended to:

$$L ::= \tilde{C}{\to}\diamond \mid \tilde{C}{\multimap}\diamond \qquad S ::= !\langle\tilde{U}\rangle; S \mid ?(\tilde{U}); S \mid \cdots$$

As in [22,23], the type system for $\mathsf{HO}\widetilde{\pi}$ disallows a shared name that directly carries polyadic shared names.

By combining $\mathsf{HO}\pi^+$ and $\mathsf{HO}\widetilde{\pi}$ into a single calculus we obtain $\mathsf{HO}\widetilde{\pi}^+$: the extension of $\mathsf{HO}\pi$ allows *both* higher-order abstractions/aplications and polyadicity.

Precise Encodings of $\mathsf{HO}\pi^+$ *and* $\mathsf{HO}\widetilde{\pi}$ *into* $\mathsf{HO}\pi$. We give encodings of $\mathsf{HO}\pi^+$ into $\mathsf{HO}\pi$ and into $\mathsf{HO}\widetilde{\pi}$, and show that they are precise. We use encoding composition (Proposition 1) to encode $\mathsf{HO}\widetilde{\pi}^+$ into HO and π. We consider the following typed calculi (cf. Definition 10):

- $\mathcal{L}_{\mathsf{HO}\pi^+} = \langle \mathsf{HO}\pi^+, \mathcal{T}_4, \overset{\ell}{\longmapsto}, \approx^{\mathrm{H}}, \vdash \rangle$, where $\mathcal{T}_4$ is a set of types of $\mathsf{HO}\pi^+$; the typing $\vdash$ is defined in Sect. 4 with extended rules [Abs] and [App].
- $\mathcal{L}_{\mathsf{HO}\widetilde{\pi}} = \langle \mathsf{HO}\widetilde{\pi}, \mathcal{T}_5, \overset{\ell}{\longmapsto}, \approx^{\mathrm{H}}, \vdash \rangle$, where $\mathcal{T}_5$ is the set of types of $\mathsf{HO}\widetilde{\pi}$; the typing $\vdash$ is defined in in Sect. 4 with polyadic types.

First, the typed encoding $\langle [\![\cdot]\!]^3, (\cdot)^3 \rangle : \mathsf{HO}\pi^+ \to \mathsf{HO}\pi$ is defined in Fig. 8. It satisfies the following properties:

Proposition 9 ($\mathsf{HO}\pi^+$ into $\mathsf{HO}\pi$: Type Preservation). *The encoding from $\mathcal{L}_{\mathsf{HO}\pi^+}$ into $\mathcal{L}_{\mathsf{HO}\pi}$ (cf. Fig. 8) is type preserving.*

Proposition 10 (Operational Correspondence: From $\mathsf{HO}\pi^+$ to $\mathsf{HO}\pi$-Excerpt). *Let P be an $\mathsf{HO}\pi^+$ process such that $\Gamma; \emptyset; \Delta \vdash P$.*

1. *Completeness: $\Gamma; \Delta \vdash P \overset{\ell}{\longmapsto} \Delta' \vdash P'$ implies*
   (a) *If $\ell = \tau_\beta$ then $(\Gamma)^3; (\Delta)^3 \vdash [\![P]\!]^3 \overset{\tau}{\longmapsto} \Delta'' \vdash R$ and $(\Gamma)^3; (\Delta')^3 \vdash [\![P']\!]^3 \approx^H \Delta'' \vdash R$, for some R;*
   (b) *If $\ell = \tau$ and $\ell \neq \tau_\beta$ then $(\Gamma)^3; (\Delta)^3 \vdash [\![P]\!]^3 \overset{\tau}{\longmapsto} (\Delta')^3 \vdash [\![P']\!]^3$.*
2. *Soundness: $(\Gamma)^3; (\Delta)^3 \vdash [\![P]\!]^3 \overset{\tau}{\longmapsto} (\Delta'')^3 \vdash Q$ implies either*
   (a) *$\Gamma; \Delta \vdash P \overset{\tau}{\longmapsto} \Delta' \vdash P'$ with $Q \equiv [\![P']\!]^3$*
   (b) *$\Gamma; \Delta \vdash P \overset{\tau_\beta}{\longmapsto} \Delta' \vdash P'$ and $(\Gamma)^3; (\Delta'')^3 \vdash Q \overset{\tau_\beta}{\longmapsto} (\Delta'')^3 \vdash [\![P']\!]^3$.*

Types :

$$(\!|L{\to}\diamond|\!)^3 \overset{\text{def}}{=} ?((\!|L|\!)^3); \mathsf{end}{\to}\diamond \qquad (\!|!\langle L{\to}\diamond\rangle; S|\!)^3 \overset{\text{def}}{=} !\langle(\!|L{\to}\diamond|\!)^3\rangle; (\!|S|\!)^3$$

$$(\!|L{\multimap}\diamond|\!)^3 \overset{\text{def}}{=} ?((\!|L|\!)^3); \mathsf{end}{\multimap}\diamond \qquad (\!|?(L{\to}\diamond); S|\!)^3 \overset{\text{def}}{=} ?((\!|L{\to}\diamond|\!)^3); (\!|S|\!)^3$$

Terms :

$$\{\!\{x\}\!\}^3 \overset{\text{def}}{=} x$$

$$\{\!\{\lambda x : L.P\}\!\}^3 \overset{\text{def}}{=} \lambda z.z?(x).[\![P]\!]^3$$

$$[\![(x : L)\, V]\!]^3 \overset{\text{def}}{=} (\nu\, s)(x\, s \mid \overline{s}!\langle\{\!\{V\}\!\}^3\rangle.\mathbf{0})$$

$$[\![u!\langle\lambda x : L.Q\rangle.P]\!]^3 \overset{\text{def}}{=} u!\langle\{\!\{\lambda x.Q\}\!\}^3\rangle.[\![P]\!]^3$$

$$[\![(\lambda x : L.P)\, V]\!]^3 \overset{\text{def}}{=} (\nu\, s)(s?(x).[\![P]\!]^3 \mid \overline{s}!\langle\{\!\{V\}\!\}^3\rangle.\mathbf{0})$$

Mappings for elided processes and types are homomorphic.

Fig. 8. Encoding of $\mathsf{HO}\pi^+$ into $\mathsf{HO}\pi$.

Proposition 11 (Full Abstraction. From HOπ⁺ to HOπ).
Let P, Q be HOπ⁺ processes with $\Gamma; \emptyset; \Delta_1 \vdash P \triangleright \diamond$ and $\Gamma; \emptyset; \Delta_2 \vdash Q \triangleright \diamond$. Then $\Gamma; \Delta_1 \vdash P \approx^H \Delta_2 \vdash Q$ if and only if $(\Gamma)^3; (\Delta_1)^3 \vdash [\![P]\!]^3 \approx^H (\Delta_2)^3 \vdash [\![Q]\!]^3$

Using the above propositions, Theorems 3 and 4, and Proposition 1, we derive the following:

Theorem 7 (Encoding HOπ⁺ into HOπ). *The encoding from $\mathcal{L}_{\mathsf{HO}\pi^+}$ into $\mathcal{L}_{\mathsf{HO}\pi}$ (cf. Fig. 8) is precise. Hence, the encodings from $\mathcal{L}_{\mathsf{HO}\pi^+}$ to $\mathcal{L}_{\mathsf{HO}}$ and $\mathcal{L}_{\pi}$ are also precise.*

Second, we define the typed encoding $\langle [\![\cdot]\!]^4, (\cdot)^4 \rangle : \mathsf{HO}\widetilde{\pi} \to \mathsf{HO}\pi$ in Fig. 9. For simplicity, we give the dyadic case (tuples of length 2); the general case is as expected. The encoding of $\mathsf{HO}\widetilde{\pi}$ satisfies the following properties:

Proposition 12 ($\mathsf{HO}\widetilde{\pi}$ into HOπ: Type Preservation). *The encoding from $\mathcal{L}_{\mathsf{HO}\widetilde{\pi}}$ into $\mathcal{L}_{\mathsf{HO}\pi}$ (cf. Fig. 9) is type preserving.*

Proposition 13 (Operational Correspondence: From $\mathsf{HO}\widetilde{\pi}$ to HOπ-Excerpt). *Let $\Gamma; \emptyset; \Delta \vdash P$.*

1. *Completeness: $\Gamma; \Delta \vdash P \overset{\ell}{\longmapsto} \Delta' \vdash P'$ implies*
   (a) *If $\ell = \tau_\beta$ then $(\Gamma)^4; (\Delta)^4 \vdash [\![P]\!]^4 \overset{\tau_\beta}{\longmapsto} \overset{\tau_s}{\longmapsto} \overset{\tau_s}{\longmapsto} (\Delta')^4 \vdash [\![P']\!]^4$*
   (b) *If $\ell = \tau$ then $(\Gamma)^4; (\Delta)^4 \vdash [\![P]\!]^4 \overset{\tau}{\longmapsto} \overset{\tau}{\longmapsto} \overset{\tau}{\longmapsto} (\Delta')^4 \vdash [\![P']\!]^4$*
2. *Soundness: $(\Gamma)^4; (\Delta)^4 \vdash [\![P]\!]^4 \overset{\ell}{\longmapsto} (\Delta_1)^4 \vdash P_1$ implies*
   (a) *If $\ell = \tau_\beta$ then $\Gamma; \Delta \vdash P \overset{\tau_\beta}{\longmapsto} \Delta' \vdash P'$ and $(\Gamma)^4; (\Delta_1)^4 \vdash P_1 \overset{\tau_s}{\longmapsto} \overset{\tau_s}{\longmapsto} (\Delta')^4 \vdash (P')^4$*
   (b) *If $\ell = \tau$ then $\Gamma; \Delta \vdash P \overset{\tau}{\longmapsto} \Delta' \vdash P'$ and $(\Gamma)^4; (\Delta_1)^4 \vdash P_1 \overset{\tau}{\longmapsto} \overset{\tau}{\longmapsto} \overset{\tau}{\longmapsto} (\Delta')^4 \vdash (P')^4$*

Types :

$$
\begin{array}{rcl}
(\!|!\langle S_1, S_2\rangle; S|\!)^4 & \stackrel{\text{def}}{=} & !\langle (\!|S_1|\!)^4\rangle; !\langle (\!|S_2|\!)^4\rangle; (\!|S|\!)^4 \\
(\!|!\langle L\rangle; S|\!)^4 & \stackrel{\text{def}}{=} & !\langle (\!|L|\!)^4\rangle; (\!|S|\!)^4 \\
(\!|(C_2, C_2)\to\diamond|\!)^4 & \stackrel{\text{def}}{=} & (?((\!|C_1|\!)^4); ?((\!|C_2|\!)^4); \mathsf{end})\to\diamond \\
(\!|(C_1, C_2)\multimap\diamond|\!)^4 & \stackrel{\text{def}}{=} & (?((\!|C_1|\!)^4); ?((\!|C_2|\!)^4); \mathsf{end})\multimap\diamond
\end{array}
$$

Terms :

$$
\begin{array}{rcl}
[\![u!\langle u_1, u_2\rangle.P]\!]^4 & \stackrel{\text{def}}{=} & u!\langle u_1\rangle.u!\langle u_2\rangle.[\![P]\!]^4 \\
[\![u!\langle\lambda(x_1, x_2).Q\rangle.P]\!]^4 & \stackrel{\text{def}}{=} & u!\langle\lambda z.z?(x_1).z?(x_2).[\![Q]\!]^4\rangle.[\![P]\!]^4 \\
[\![x(u_1, u_2)]\!]^4 & \stackrel{\text{def}}{=} & (\nu\, s)(x\, s \mid \overline{s}!\langle u_1\rangle.\overline{s}!\langle u_2\rangle.\mathbf{0}) \\
[\![(\lambda(x_1, x_2).P)(u_1, u_2)]\!]^4 & \stackrel{\text{def}}{=} & (\nu\, s)(s?(x_1).s?(x_2).[\![P]\!]^4 \mid \overline{s}!\langle u_1\rangle.\overline{s}!\langle u_2\rangle.\mathbf{0})
\end{array}
$$

The input cases are defined as the output cases by replacing ! by ?. Elided mappings for processes and types are homomorphic.

Fig. 9. Encoding of $\mathsf{HO}\widetilde{\pi}$ (dyadic case) into HOπ.

Proposition 14 (Full Abstraction: From $\mathsf{HO}\widetilde{\pi}$ to $\mathsf{HO}\pi$). *Let P, Q be $\mathsf{HO}\widetilde{\pi}$ processes with $\Gamma; \emptyset; \Delta_1 \vdash P \triangleright \diamond$ and $\Gamma; \emptyset; \Delta_2 \vdash Q \triangleright \diamond$. Then we have:*
$\Gamma; \Delta_1 \vdash P \approx^{H} \Delta_2 \vdash Q$ if and only if $(\Gamma)^4; (\Delta_1)^4 \vdash [\![P]\!]^4 \approx^{H} (\Delta_2)^4 \vdash [\![Q]\!]^4$.

Using the above propositions, Theorems 3 and 4, and Proposition 1, we derive the following:

Theorem 8 (Encoding of $\mathsf{HO}\widetilde{\pi}$ into $\mathsf{HO}\pi$). *The encoding from $\mathcal{L}_{\mathsf{HO}\widetilde{\pi}}$ into $\mathcal{L}_{\mathsf{HO}\pi}$ (cf. Fig. 9) is precise. Hence, the encodings from $\mathcal{L}_{\mathsf{HO}\widetilde{\pi}}$ to $\mathcal{L}_{\mathsf{HO}}$ and $\mathcal{L}_{\pi}$ are also precise.*

By combining Theorems 7 and 8, we can extend preciseness to the super-calculus $\mathsf{HO}\widetilde{\pi}^{+}$.

9 Concluding Remarks and Related Work

We have thoroughly studied the expressivity of the higher-order π-calculus with sessions, here denoted $\mathsf{HO}\pi$. Unlike most previous works, we have carried out our study in the setting of *session types*. Types not only delineate and enable encodings; they inform the techniques required to reason about such encodings. Our results cover a wide spectrum of features intrinsic to higher-order concurrency: pure process-passing (first- and higher-order abstractions), name-passing, polyadicity, linear/shared communication (cf. Fig. 1). Remarkably, the discipline embodied by session types turns out to be fundamental to show that all these languages are equally expressive, up to strong typed bisimilarities. Indeed, although our encodings may be used in an untyped setting, session type information is critical to establish key properties for preciseness, in particular full abstraction.

Related Work. There is a vast literature on expressiveness for process calculi; we refer to [28] and [29, Sect. 2.3] for surveys. Our study casts known results [32] into a session typed setting, and offers new encodability results. Our work stresses the view of "encodings as protocols", namely session protocols which enforce linear and shared disciplines for names, a distinction little explored in previous works. This distinction enables us to obtain refined operational correspondence results (cf. Propositions. 3, 6, 10, 13). We showed that HO suffices to encode the first-order session calculus [11], here denoted π. To our knowledge, this is a new result; its significance is stressed by the demanding encodability criteria considered, in particular full abstraction up to typed bisimilarities ($\approx^{\mathsf{H}}/\approx^{\mathsf{C}}$, cf. Propositions 4 and 7). This encoding is relevant in a broader setting, as known encodings of name-passing into higher-order calculi [4,19,36,42,44] require limitations in source/target languages, do not consider types, and/or fail to satisfy strong encodability criteria (see below). We also showed that HO can encode $\mathsf{HO}\pi$ and its extension with higher-order applications ($\mathsf{HO}\pi^{+}$). Thus, all these calculi are equally expressive with fully abstract encodings (up to $\approx^{\mathsf{H}}/\approx^{\mathsf{C}}$). These appear to be the first results of this kind.

Early works on (relative) expressiveness appealed to different notions of encoding. Later on, proposals of abstract frameworks which formalise the notion

of encoding and state associated syntactic/semantic criteria were put forward; recent proposals include [8,10,30,31,41]. Our formulation of precise encoding (Definition 15) builds upon existing proposals (e.g., [10,16,26]) to account for the session types associated to $\mathsf{HO}\pi$.

Early expressiveness studies for higher-order calculi are [32,39]; recent works include [4,16,17,42,43]. Due to the close relationship between higher-order process calculi and functional calculi, encodings of (variants of) the λ-calculus into the π-calculus (see, e.g., [1,7,33,37,46]) are also related. Sangiorgi's encoding of the higher-order π-calculus into the π-calculus [32] is fully abstract with respect to reduction-closed, barbed congruence. We have shown in Sect. 7.2 that the analogue of Sangiorgi's encoding for the session typed setting also satisfies full abstraction (up to $\approx^{\mathsf{H}}/\approx^{\mathsf{C}}$, cf. Proposition 6). A basic form of input/output types is used in [35], where the encoding in [32] is casted in the asynchronous setting, with output and applications coalesced in a single construct. Building upon [35], a simply typed encoding for synchronous processes is given in [36]; the reverse encoding (i.e., first-order communication into higher-order processes) is also studied for an asynchronous, localised π-calculus (only the output capability of names can be sent around). The work [34] studies hierarchies for calculi with *internal* first-order mobility and with higher-order mobility without name-passing (similarly as the subcalculus HO). The hierarchies are defined according to the order of types needed in typing. Via fully abstract encodings, it is shown that that name- and process-passing calculi with equal order of types have the same expressiveness.

Other related works are [4,17,19,42]. The paper [4] gives a fully abstract encoding of the π-calculus into Homer, a higher-order calculus with explicit locations, local names, and nested locations. The paper [19] presents a *reflective* calculus with a "quoting" operator: names are quoted processes and represent the code of a process; name-passing is then a way of passing the code of a process. This reflective calculus can encode both first- and higher-order π-calculus. Building upon [40], the work [42] studies the (non)encodability of the untyped π-calculus into a higher-order π-calculus with a powerful name relabelling operator, which is essential to encode name-passing. The paper [44] defines an encoding of the (untyped) π-calculus without relabeling. This encoding is quite different from the one in Sect. 7.1: in [44] names are encoded using polyadic name abstractions (called *pipes*); guarded replication enables infinite behaviours. While our encoding satisfies full abstraction, the encoding in [44] does not: only divergence-reflection and operational correspondence (soundness and completeness) properties are established. Soundness is stated up-to *pipe-bisimilarity*, an equivalence tailored to the encoding strategy; the authors of [44] describe this result as "weak".

A core higher-order calculus is studied in [17]: it lacks restriction, name passing, output prefix, and replication/recursion. Still, this subcalculus of HO is Turing equivalent. The work [16] extends this core calculus with restriction, output prefix, and polyadicity; it shows that synchronous communication can encode asynchronous communication, and that process passing polyadicity induces an expressiveness hierarchy. The paper [43] complements [16] by

studying the expressivity of second-order process abstractions. Polyadicity is shown to induce an expressiveness hierarchy; also, by adapting the encoding in [32], process abstractions are encoded into name abstractions. In contrast, here we give a fully abstract encoding of $\mathsf{HO}\widetilde{\pi}^+$ into HO that preserves session types; this improves [16,43] by enforcing linearity disciplines on process behaviour. The focus of [16,42–44] is on untyped, higher-order processes; they do not address communication disciplined by (session) type systems.

Within session types, the works [5,6] encode binary sessions into a linearly typed π-calculus. While [6] gives an encoding of π into a linear calculus (an extension of [1]), the work [5] gives operational correspondence (without full abstraction) for the first- and higher-order π-calculi into [13]. By the result of [6], $\mathsf{HO}\pi^+$ is encodable into the linearly typed π-calculi. The syntax of $\mathsf{HO}\pi$ is a subset of that in [22,23]. The work [22] develops a higher-order session calculus with process abstractions and applications; it admits the type $U = U_1 \to U_2 \ldots U_n \to \diamond$ and its linear type U^1 which corresponds to $\tilde{U}{\to}\diamond$ and $\tilde{U}{\multimap}\diamond$ in a super-calculus of $\mathsf{HO}\pi^+$ and $\mathsf{HO}\widetilde{\pi}$. Our results show that the calculus in [22] is not only expressed but also reasoned in HO via precise encodings (with a limited form of arrow types: $C{\to}\diamond$ and $C{\multimap}\diamond$). The recent work [25] studies two encodings: from PCF with an effect system into a session-typed π-calculus, and its reverse. The reverse encoding is used to implement session channel passing in Concurrent Haskell. In future work we plan to use the core calculi studied in this paper to implement higher-order communication efficiently into Concurrent Haskell without losing its expressiveness.

Acknowledgments. We have benefited from feedback from the users of the Moca mailing list, in particular Greg Meredith and Xu Xian. We are grateful to the anonymous reviewers for their useful remarks and suggestions. This work has been partially sponsored by the Doctoral Prize Fellowship, EPSRC EP/K011715/1, EPSRC EP/K034413/1, and EPSRC EP/L00058X/1, EU project FP7-612985 UpScale, and EU COST Action IC1201 BETTY. Pérez is also affiliated to the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS), Universidade Nova de Lisboa, Portugal.

References

1. Berger, M., Honda, K., Yoshida, N.: Sequentiality and the π-calculus. In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, pp. 29–45. Springer, Heidelberg (2001)
2. Bernardi, G., Dardha, O., Gay, S.J., Kouzapas, D.: On duality relations for session types. In: Maffei, M., Tuosto, E. (eds.) TGC 2014. LNCS, vol. 8902, pp. 51–66. Springer, Heidelberg (2014)
3. Bono, V., Padovani, L.: Typing copyless message passing. LMCS **8**(1), 1–50 (2012)
4. Bundgaard, M., Hildebrandt, T.T., Godskesen, J.C.: A cps encoding of name-passing in higher-order mobile embedded resources. Theor. Comput. Sci. **356**(3), 422–439 (2006)
5. Dardha, O., Giachino, E., Sangiorgi, D.: Session types revisited. In: Proceedings of PPDP 2012, pp. 139–150. ACM (2012)

6. Demangeon, R., Honda, K.: Full abstraction in a subtyped pi-calculus with linear types. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 280–296. Springer, Heidelberg (2011)
7. Fu, Y.: Variations on mobile processes. Theor. Comput. Sci. **221**(1–2), 327–368 (1999)
8. Fu, Y., Lu, H.: On the expressiveness of interaction. Theor. Comput. Sci. **411** (11–13), 1387–1451 (2010)
9. Gay, S.J., Vasconcelos, V.T.: Linear type theory for asynchronous session types. J. Funct. Program. **20**(1), 19–50 (2010)
10. Gorla, D.: Towards a unified approach to encodability and separation results for process calculi. Inf. Comput. **208**(9), 1031–1053 (2010)
11. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, p. 122. Springer, Heidelberg (1998)
12. Honda, K., Yoshida, N.: On reduction-based process semantics. TCS **151**(2), 437–486 (1995)
13. Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the Pi-Calculus. TOPLAS **21**(5), 914–947 (1999)
14. Kouzapas, D., Pérez, J.A., Yoshida, N.: Characteristic bisimulation for higher-order session processes. In: CONCUR 2015. LIPIcs, vol. 42, pp. 398–411, Dagstuhl, Germany (2015)
15. Kouzapas, D., Pérez, J.A., Yoshida, N.: Full version of this paper. Technical report, Imperial College / University of Groningen (2015). http://arxiv.org/abs/1502.02585
16. Lanese, I., Pérez, J.A., Sangiorgi, D., Schmitt, A.: On the expressiveness of polyadic and synchronous communication in higher-order process calculi. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 442–453. Springer, Heidelberg (2010)
17. Lanese, I., Pérez, J.A., Sangiorgi, D., Schmitt, A.: On the expressiveness and decidability of higher-order process calculi. Inf. Comput. **209**(2), 198–226 (2011)
18. League, C., Shao, Z., Trifonov, V.: Type-preserving compilation of Featherweight Java. ACM Trans. Program. Lang. Syst. **24**(2), 112–152 (2002)
19. Meredith, L.G., Radestock, M.: A reflective higher-order calculus. Electr. Notes Theor. Comput. Sci. **141**(5), 49–67 (2005)
20. Milner, R., Sangiorgi, D.: Barbed bisimulation. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 685–695. Springer, Heidelberg (1992)
21. Morrisett, J.G., Walker, D., Crary, K., Glew, N.: From system F to typed assembly language. ACM Trans. Program. Lang. Syst. **21**(3), 527–568 (1999)
22. Mostrous, D., Yoshida, N.: Two session typing systems for higher-order mobile processes. In: Della Rocca, S.R. (ed.) TLCA 2007. LNCS, vol. 4583, pp. 321–335. Springer, Heidelberg (2007)
23. Mostrous, D., Yoshida, N.: Session typing and asynchronous subtying for higher-order π-Calculus. Inf. Comput. **241**, 227–263 (2015)
24. Nestmann, U.: What is a "good" encoding of guarded choice? Inf. Comput. **156** (1–2), 287–319 (2000)
25. Orchard, D., Yoshida, N.: Effects as sessions, sessions as effects. In: POPL. ACM (2016)
26. Palamidessi, C.: Comparing the expressive power of the synchronous and asynchronous pi-calculi. MSCS **13**(5), 685–719 (2003)

27. Palamidessi, C., Saraswat, V.A., Valencia, F.D., Victor, B.: On the expressiveness of linearity vs persistence in the asychronous pi-calculus. Proc. LICS **2006**, 59–68 (2006)
28. Parrow, J.: Expressiveness of process algebras. Electr. Notes Theor. Comput. Sci. **209**, 173–186 (2008)
29. Pérez, J.A.: Higher-Order Concurrency: Expressiveness and Decidability Results. PhD thesis, University of Bologna (2010)
30. Peters, K., Nestmann, U., Goltz, U.: On distributability in process calculi. In: Felleisen, M., Gardner, P. (eds.) ESOP 2013. LNCS, vol. 7792, pp. 310–329. Springer, Heidelberg (2013)
31. Peters, K., van Glabbeek, R.J.: Analysing and comparing encodability criteria. In: Proceedings of EXPRESS/SOS. EPTCS, vol. 190, pp. 46–60 (2015)
32. Sangiorgi, D.: Expressing Mobility in Process Algebras: First-Order and Higher Order Paradigms. PhD thesis, University of Edinburgh (1992)
33. Sangiorgi, D.: The lazy lambda calculus in a concurrency scenario. In: 7th LICS Conference, pp. 102–109. IEEE Computer Society Press (1992)
34. Sangiorgi, D.: π-calculus, internal mobility and agent-passing calculi. TCS **167**(2), 235–274 (1996)
35. Sangiorgi, D.: Asynchronous process calculi: the first- and higher-order paradigms. Theor. Comput. Sci. **253**(2), 311–350 (2001)
36. Sangiorgi, D., Walker, D.: The π-Calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
37. Sangiorgi, D., Xu, X.: Trees from functions as processes. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 78–92. Springer, Heidelberg (2014)
38. Shao, Z., Appel, A.W.: A type-based compiler for standard ML. In: Proceedings of PLDI 1995, pp. 116–129. ACM (1995)
39. Thomsen, B.: Calculi for Higher Order Communicating Systems. PhD thesis, Department of Computer Science, Imperial College (1990)
40. Thomsen, B.: Plain CHOCS: a second generation calculus for higher order processes. Acta Informatica **30**(1), 1–59 (1993)
41. van Glabbeek, R.J.: Musings on encodings and expressiveness. In Proceedings of EXPRESS/SOS. EPTCS, vol. 89, pp. 81–98 (2012)
42. Xu, X.: Distinguishing and relating higher-order and first-order processes by expressiveness. Acta Informatica **49**(7–8), 445–484 (2012)
43. Xu, X., Yin, Q., Long, H.: On the expressiveness of parameterization in process-passing. In: Tuosto, E., Ouyang, C. (eds.) WS-FM 2013. LNCS, vol. 8379, pp. 147–167. Springer, Heidelberg (2014)
44. Xu, X., Yin, Q., Long, H.: On the computation power of name parameterization in higher-order processes. In: Proceedings of ICE. EPTCS, vol. 189, pp. 114–127 (2015)
45. Yoshida, N.: Graph types for monadic mobile processes. In: Chandru, V., Vinay, V. (eds.) FSTTCS 1996. LNCS, vol. 1180, pp. 371–386. Springer, Heidelberg (1996)
46. Yoshida, N., Berger, M., Honda, K.: Strong normalisation in the pi -calculus. Inf. Comput. **191**(2), 145–202 (2004)