On the Expressiveness of Polyadic and Synchronous Communication in Higher-Order Process Calculi

Ivan Lanese[1], Jorge A. Pérez[2], Davide Sangiorgi[1], and Alan Schmitt[3]

[1] Laboratory FOCUS (University of Bologna / INRIA)
[2] CITI - Department of Computer Science, FCT New University of Lisbon
[3] INRIA

Abstract. *Higher-order process calculi* are calculi in which processes can be communicated. We study the expressiveness of *strictly* higher-order process calculi, and focus on two issues well-understood for first-order calculi but not in the higher-order setting: *synchronous* vs. *asynchronous* communication and *polyadic* vs. *monadic* communication. First, and similarly to the first-order setting, synchronous process-passing is shown to be encodable into asynchronous process-passing. Then, the absence of name-passing is shown to induce a hierarchy of higher-order process calculi based on the arity of polyadic communication, thus revealing a striking point of contrast with respect to first-order calculi. Finally, the passing of *abstractions* (i.e., functions from processes to processes) is shown to be more expressive than process-passing alone.

1 Introduction

Higher-order process calculi are calculi in which processes can be communicated. In this paper, we study the expressive power of *strictly* higher-order process calculi, and concentrate on fundamental questions of expressiveness in process calculi at large: *asynchronous* vs. *synchronous* communication and *polyadic* vs. *monadic* communication. These are well-understood issues for first-order process calculi: several works (see, e.g., [1,2,3]) have studied the *asynchronous π-calculus* [4,5] and its relationship with the (synchronous) π-calculus. Also, the encoding of polyadic communication into monadic communication in the π-calculus [6] is simple and very robust [7,8]. However, analogous studies are lacking for calculi in the higher-order setting.

We approach these questions in the context of HOπ, a *strictly* higher-order process calculus (i.e., it has no name-passing features) [9]. HOπ is very expressive: it is Turing complete and several modelling idioms are expressible in it as derived constructs. Hence, answers to the questions we are interested in are far from obvious. We shall consider SHO and AHO, the synchronous and asynchronous variants of HOπ with polyadic communication (Section 2). SHO and AHO represent two *families* of higher-order process calculi: given $n \geq 0$, SHO^n (resp. AHO^n) denotes the synchronous (resp. asynchronous) higher-order process calculus with n-adic communication.

A fundamental consideration in strictly higher-order process calculi is that scope extrusions have a limited effect. In a process-passing setting, received processes can only be executed, forwarded, or discarded. Hence, an input context cannot gain access to the (private) names of the processes it receives; to the context, received processes are

much like a "black box". Although higher-order communications might lead to scope extrusion of the private names *contained* in the transmitted processes, such extrusions are of little significance: without name-passing, a receiving context can only use the names contained in a process in a restricted way, namely the way decreed by the sender process.[4] In a process-passing setting, sharing of (private) names is thus rather limited.

We begin by investigating the relationship between synchrony and asynchrony. Our first contribution is an *encodability* result: an encoding of SHO^n into AHO^n (Section 4). This reveals a similarity between first- and higher-order process calculi. Intuitively, a synchronous output is encoded by an asynchronous output that communicates both the communication object and its continuation. In Section 5 we move to examine the situation for polyadic communication. We consider variants of SHO with different arity in communications, and study their relative expressive power. Interestingly, in the case of polyadic communication, the absence of name-passing causes a loss in expressive power. Our second contribution is a *non-encodability* result: for every $n > 1$, SHO^n cannot be encoded into SHO^{n-1}. We thus obtain a *hierarchy* of higher-order process calculi of strictly increasing expressiveness. Hence, polyadic communication is a striking point of contrast between first- and higher-order process calculi. Finally, in Section 6 we consider the extension of SHO with *abstraction-passing*. An abstraction is an expression parametric on processes; the expressiveness of abstraction-passing is thus specific to the higher-order setting. We consider SHO^n_a, the extension of SHO^n with abstractions of order one (i.e., functions from processes to processes). We show that SHO^n can be encoded into SHO^1_a. Our final contribution uses this result to show that there is no encoding of SHO^n_a into SHO^m for $n, m > 0$.

Our notion of encoding exploits a refined account of internal actions: in SHO, the internal actions that result from synchronizations on restricted names are distinguished from those resulting from synchronizations on public names. Only the former are considered as internal actions; the latter are regarded as visible actions. While this distinction might appear as demanding in the light of recent proposals for "good encodings" (e.g., [10]), we find it useful to focus on *compositional* encodings that are *robust with respect to interferences*, that is, encodings that work in an arbitrary context of the target language (i.e., not necessarily a context in the image of the encoding). Further, the distinction is crucial in certain technical details of our proofs.

Extended discussions and full technical details can be found in [11, Chapter 6].

2 The Calculi

We define SHO^n and AHO^n, the two families of higher-order process calculi we shall be working with.

Definition 1 *Let x, y range over process variables, and $a, b, \ldots r, s, \ldots$ denote names. The language of* SHO *processes is given by the following syntax:*

$$P, Q, \ldots ::= a(\widetilde{x}).\, P \;\mid\; \bar{a}\langle \widetilde{Q} \rangle.\, P \;\mid\; P_1 \parallel P_2 \;\mid\; \nu r\, P \;\mid\; x \;\mid\; \mathbf{0}$$

[4] Here we refer to process-passing *without* passing of abstractions, i.e. functions from processes to processes. As we shall see, the situation is rather different with abstraction-passing.

$$\textsc{Inp}\ \frac{}{a(\widetilde{x}).\,P \xrightarrow{\ a(\widetilde{x})\ } P} \qquad \textsc{Out}\ \frac{}{\overline{a}\langle\widetilde{Q}\rangle.\,P \xrightarrow{\ \overline{a}\langle\widetilde{Q}\rangle\ } P} \qquad \textsc{Act1}\ \frac{P_1 \xrightarrow{\ \alpha\ } P_1' \quad \mathsf{cond}(\alpha, P_2)}{P_1 \parallel P_2 \xrightarrow{\ \alpha\ } P_1' \parallel P_2}$$

$$\textsc{Res}\ \frac{P \xrightarrow{\ \alpha\ } P' \quad r \notin \mathsf{n}(\alpha)}{\nu r\, P \xrightarrow{\ \alpha\ } \nu r\, P'} \qquad \textsc{Open}\ \frac{P \xrightarrow{\ (\nu \widetilde{s})\overline{a}\langle \widetilde{P}''\rangle\ } P' \quad r \neq a,\ r \in \mathsf{fn}(\widetilde{P}'') - \widetilde{s}}{\nu r\, P \xrightarrow{\ (\nu r \widetilde{s})\overline{a}\langle \widetilde{P}''\rangle\ } P'}$$

$$\textsc{Tau1}\ \frac{P_1 \xrightarrow{\ (\nu \widetilde{s})\overline{a}\langle \widetilde{P}\rangle\ } P_1' \quad P_2 \xrightarrow{\ a(\widetilde{x})\ } P_2' \quad \widetilde{s} \cap \mathsf{fn}(P_2) = \emptyset}{P_1 \parallel P_2 \xrightarrow{\ a\tau\ } \nu \widetilde{s}\,(P_1' \parallel P_2'\{\widetilde{P}/\widetilde{x}\})} \qquad \textsc{IntRes}\ \frac{P \xrightarrow{\ a\tau\ } P'}{\nu a\, P \xrightarrow{\ \tau\ } \nu a\, P}$$

Fig. 1. The LTS of SHO. Symmetric rules ACT2 and TAU2 are omitted.

Using standard notations and properties for tuples of syntactic elements, polyadicity in process-passing is interpreted as expected: an output prefixed process $\overline{a}\langle\widetilde{Q}\rangle.\,P$ sends the tuple of processes $\widetilde{Q}$ on name (or channel) a and then continues as P; an input prefixed process $a(\widetilde{x}).\,P$ can receive a tuple $\widetilde{Q}$ on name a and continue as $P\{\widetilde{Q}/\widetilde{x}\}$. In both cases, a is said to be the *subject* of the action. We write $\mid \widetilde{x} \mid$ for the length of tuple $\widetilde{x}$; the length of the tuples that are passed around determines the actual *arity* in polyadic communication. In interactions, we assume inputs and outputs have the same arity; we shall rely on notions of *types* and *well-typed processes* as in [9]. Parallel composition allows processes to interact, and $\nu r\, P$ makes r private (or restricted) to the process P. Notions of bound and free names and variables ($\mathsf{bn}(\cdot)$, $\mathsf{fn}(\cdot)$, $\mathsf{bv}(\cdot)$, and $\mathsf{fv}(\cdot)$, resp.) are defined in the usual way: an input $a(\widetilde{x}).\,P$ binds the free occurrences of variables in $\widetilde{x}$ in P; similarly, $\nu r\, P$ binds the free occurrences of name r in P. We abbreviate $a(\widetilde{x}).\,P$ as $a.\,P$ when none of the variables in $\widetilde{x}$ is in $\mathsf{fv}(P)$; $\overline{a}\langle\widetilde{\mathbf{0}}\rangle.\,P$ as $\overline{a}.\,P$; $\overline{a}\langle Q\rangle.\,\mathbf{0}$ as $\overline{a}\langle Q\rangle$; and $\nu a\,\nu b\,P$ as $\nu a\,b\,P$. Notation $\prod^k P$ stands for k copies of process P in parallel.

The semantics for SHO is given by the Labelled Transition System (LTS) in Figure 1; we use $\mathsf{cond}(\alpha, P)$ to abbreviate $\mathsf{bv}(\alpha) \cap \mathsf{fv}(P) = \emptyset \wedge \mathsf{bn}(\alpha) \cap \mathsf{fn}(P) = \emptyset$. As anticipated, we distinguish between *internal* and *public* synchronizations. The former are given by synchronizations on *restricted* names, are the only source of internal behavior, and are denoted as $\xrightarrow{\ \tau\ }$. The latter are given by synchronization on *public* names: a synchronization on the public name a leads to the visible action $\xrightarrow{\ a\tau\ }$. We thus have four kinds of transitions: in addition to internal and public synchronizations, there are input transitions $P \xrightarrow{\ a(\widetilde{x})\ } P'$, and output transitions $P \xrightarrow{\ (\nu \widetilde{s})\overline{a}\langle\widetilde{Q}\rangle\ } P'$ (with extrusion of the tuple of names $\widetilde{s}$), which have the expected meaning. We use α to range over actions. The *signature* of α, $\mathsf{sig}(\alpha)$, is defined as $\mathsf{sig}(a(\widetilde{x})) = a\,\mathsf{in}$, $\mathsf{sig}((\nu \widetilde{s})\overline{a}\langle\widetilde{Q}\rangle) = a\,\mathsf{out}$, $\mathsf{sig}(a\tau) = a\tau$, $\mathsf{sig}(\tau) = \tau$, and is undefined otherwise. Notions of bound/free names and variables extend to actions as expected. We use $\vec{\alpha}$ to denote a sequence of actions $\alpha_1, \ldots, \alpha_n$. Weak transitions are defined in the usual way. We write $\Rightarrow$ for the reflexive, transitive closure of $\xrightarrow{\ \tau\ }$. Given an action $\alpha \neq \tau$, notation $\stackrel{\alpha}{\Longrightarrow}$ stands for $\Rightarrow\xrightarrow{\ \alpha\ }\Rightarrow$ and $\stackrel{\tau}{\Longrightarrow}$ stands for $\Rightarrow$. Given a sequence $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$, we define $\stackrel{\vec{\alpha}}{\Longrightarrow}$ as $\stackrel{\alpha_1}{\Longrightarrow} \cdots \stackrel{\alpha_n}{\Longrightarrow}$.

By varying the arity in polyadic communication, Definition 1 actually gives a *family* of higher-order process calculi. We have the following notational convention:

Convention 2 *For each $n > 0$, SHO^n corresponds to the calculus obtained from the syntax given in Definition 1 in which polyadic communication has arity at most n.*

Definition 3 AHO *corresponds to the fragment of* SHO *where output actions have no continuations. All the definitions extend to* AHO *processes as expected;* AHO^n *is thus the asynchronous calculus with n-adic communication.*

The following definition is standard.

Definition 4 (Barbs) *Given a process P and a name a, we write (i) $P \downarrow_a$ —a* strong input barb*— if P can perform an input action with subject a; and (ii) $P \downarrow_{\overline{a}}$ —a* strong output barb*— if P can perform an output action with subject a. Given $\mu \in \{a, \overline{a}\}$, we define a* weak *barb $P \Downarrow_\mu$ if, for some P', $P \Rightarrow P' \downarrow_\mu$.*

3 The Notion of Encoding

Our definition of encoding is inspired by the notion of "good encoding" in [10]. We say that a *language* $\mathcal{L}$ is given by: (i) an algebra of *processes* $\mathcal{P}$, with an associated function $\mathsf{fn}(\cdot)$; (ii) a labeled transition relation $\longrightarrow$ on $\mathcal{P}$, i.e., a structure $(\mathcal{P}, \mathcal{A}, \longrightarrow)$ for some set $\mathcal{A}$ of *actions* (or *labels*) with an associated function $\mathsf{sig}(\cdot)$; and (iii) a weak behavioral equivalence $\approx$ such that: if $P \approx Q$ and $P \stackrel{\alpha}{\Longrightarrow} P'$ then $Q \stackrel{\alpha'}{\Longrightarrow} Q'$, $P' \approx Q'$, and $\mathsf{sig}(\alpha) = \mathsf{sig}(\alpha')$. Given languages $\mathcal{L}_\mathsf{s} = (\mathcal{P}_\mathsf{s}, \longrightarrow_\mathsf{s}, \approx_\mathsf{s})$ and $\mathcal{L}_\mathsf{t} = (\mathcal{P}_\mathsf{t}, \longrightarrow_\mathsf{t}, \approx_\mathsf{t})$, a *translation* of $\mathcal{L}_\mathsf{s}$ into $\mathcal{L}_\mathsf{t}$ is a function $[\![\cdot]\!] : \mathcal{P}_\mathsf{s} \to \mathcal{P}_\mathsf{t}$. We shall call *encoding* any translation that satisfies the following syntactic and semantic conditions.

Definition 5 (Syntactic Conditions) *Let $[\![\cdot]\!] : \mathcal{P}_\mathsf{s} \to \mathcal{P}_\mathsf{t}$ be a translation of $\mathcal{L}_\mathsf{s}$ into $\mathcal{L}_\mathsf{t}$. We say that $[\![\cdot]\!]$ is:*

1. compositional *if for every k-ary operator $\mathtt{op}$ of $\mathcal{L}_\mathsf{s}$ and for all $S_1, \ldots, S_k$ with $\mathsf{fn}(S_1, \ldots, S_k) = N$, there exists a k-ary context $C^N_\mathtt{op} \in \mathcal{P}_\mathsf{t}$ that depends on N and $\mathtt{op}$ such that $[\![\mathtt{op}(S_1, \ldots, S_k)]\!] = C^N_\mathtt{op}[[\![S_1]\!], \ldots, [\![S_k]\!]]$;*
2. name invariant *if $[\![\sigma(P)]\!] = \sigma([\![P]\!])$, for any injective renaming of names σ.*

Definition 6 (Semantic Conditions) *Let $[\![\cdot]\!] : \mathcal{P}_\mathsf{s} \to \mathcal{P}_\mathsf{t}$ be a translation of $\mathcal{L}_\mathsf{s}$ into $\mathcal{L}_\mathsf{t}$. We say that $[\![\cdot]\!]$ is:*

1. complete *if for every $S, S' \in \mathcal{P}_\mathsf{s}$ and $\alpha \in \mathcal{A}_\mathsf{s}$ such that $S \stackrel{\alpha}{\Longrightarrow}_\mathsf{s} S'$, it holds that $[\![S]\!] \stackrel{\beta}{\Longrightarrow}_\mathsf{t} \approx_\mathsf{t} [\![S']\!]$, where $\beta \in \mathcal{A}_\mathsf{t}$ and $\mathsf{sig}(\alpha) = \mathsf{sig}(\beta)$;*
2. sound *if for every $S \in \mathcal{P}_\mathsf{s}$, $T \in \mathcal{P}_\mathsf{t}$, $\beta \in \mathcal{A}_\mathsf{t}$ such that $[\![S]\!] \stackrel{\beta}{\Longrightarrow}_\mathsf{t} T$ there exists an $S' \in \mathcal{P}_\mathsf{s}$ and an $\alpha \in \mathcal{A}_\mathsf{s}$ such that $S \stackrel{\alpha}{\Longrightarrow}_\mathsf{s} S'$, $T \Rightarrow \approx_\mathsf{t} [\![S']\!]$, and $\mathsf{sig}(\alpha) = \mathsf{sig}(\beta)$;*
3. adequate *if for every $S, S' \in \mathcal{P}_\mathsf{s}$, if $S \approx_\mathsf{s} S'$ then $[\![S]\!] \approx_\mathsf{t} [\![S']\!]$;*
4. diverge-reflecting *if for every $S \in \mathcal{P}_\mathsf{s}$, $[\![S]\!]$ diverges only if S diverges.*

Adequacy is crucial to obtain *composability* of encodings (see Prop. 7 below). We stress that we always use *weak* behavioral equivalences. Some properties of our notion of encoding are given in the following proposition, whose proof we omit for space reasons.

Proposition 7 *Let $[\![\cdot]\!]$ be an encoding of $\mathcal{L}_\mathsf{s}$ into $\mathcal{L}_\mathsf{t}$. Then $[\![\cdot]\!]$ satisfies:*

Barb preservation *For every $S \in \mathcal{P}_\mathsf{s}$ it holds that $S \Downarrow_{\overline{a}}$ (resp. $S \Downarrow_a$) if and only if $[\![S]\!] \Downarrow_{\overline{a}}$ (resp. $[\![S]\!] \Downarrow_a$).*

Preservation of free names *Let a be a name. If $a \in \mathsf{fn}(P)$ then $a \in \mathsf{fn}([\![P]\!])$.*

Composability *If $\mathcal{C}[\![\cdot]\!]$ is an encoding of $\mathcal{L}_1$ into $\mathcal{L}_2$, and $\mathcal{D}[\![\cdot]\!]$ is an encoding of $\mathcal{L}_2$ into $\mathcal{L}_3$ then their composition $(\mathcal{D} \cdot \mathcal{C})[\![\cdot]\!]$ is an encoding of $\mathcal{L}_1$ into $\mathcal{L}_3$.*

4 An Encodability Result for Synchronous Communication

Here we study the relationship between synchronous and asynchronous communication. While it is easy to define an encoding of SHO^n into AHO^{n+1} (i.e., by sending the communication object and the continuation of the output action in a single synchronization, the continuation being an additional parameter), an encoding of asynchronous process-passing into synchronous communication of the *same arity* is much more challenging. We now describe such an encoding. Intuitively, the idea is to send a *single process* consisting of a guarded choice between a communication object and the continuation of the synchronous output. For the monadic case the encoding is as follows:

$$[\![\overline{a}\langle P\rangle.\, S]\!] = \nu k\, l\, (\overline{a}\langle k.\, ([\![P]\!] \parallel \overline{k}) + l.\, ([\![S]\!] \parallel \overline{k})\rangle \parallel \overline{l}) \quad [\![a(x).\, R]\!] = a(x).\, (x \parallel [\![R]\!])$$

where "$+$" stands for the encoding of disjoint choice proposed for HOCORE [12]; k, l are names not in $\mathsf{fn}(P, S)$; and $[\![\cdot]\!]$ is an homomorphism for the other operators in SHO^1. The encoding exploits the fact that the continuation should be executed exactly once, while the communication object can be executed zero or more times. In fact, there is only one copy of $\overline{l}$, the trigger that executes the encoding of the continuation. Notice that $\overline{l}$ releases both the encoding of the continuation and a trigger for executing the encoding of the communication object (denoted $\overline{k}$); such an execution will only occur when the choice sent by the encoding of output appears at the top level. This way, it is easy to see that a trigger $\overline{k}$ is always available. This idea can be generalized as follows:

Definition 8 (Synchronous to Asynchronous) *For each $n > 0$, the encoding of SHO^n into AHO^n is defined as follows:*

$$[\![\overline{a}\langle P_1, \ldots, P_n\rangle.\, S]\!] = \nu k\, l\, (\overline{a}\langle [\![P_1]\!], \ldots, [\![P_{n-1}]\!], T_{k,l}[\, [\![P_n]\!], [\![S]\!]\,]\rangle \parallel \overline{l})$$
$$[\![a(x_1, \ldots, x_n).\, R]\!] = a(x_1, \ldots, x_n).\, (x_n \parallel [\![R]\!])$$

with $T_{k,l}[M_1, M_2] \stackrel{def}{=} k.\, (M_1 \parallel \overline{k}) + l.\, (M_2 \parallel \overline{k})$, $\{k, l\} \cap \mathsf{fn}(P_1, \ldots, P_n, S) = \emptyset$, and where $[\![\cdot]\!]$ is an homomorphism for the other operators in SHO^n.

Correctness of the encoding (i.e. proofs that the encoding satisfies the conditions in Section 3) is presented in [11]. The encoding provides compelling evidence on the expressive power of (asynchronous) process-passing. The observation that the encoding of synchronous into asynchronous communication is a particular case of that of polyadic into monadic communication leaves open the possibility that an encoding as in the π-calculus might exist in a process-passing setting. In the next section we prove that this is *not* the case.

5 Separation Results for Polyadic Communication

Here we present the separation results for SHO. Section 5.1 introduces the notion of *disjoint forms*, which are useful to capture a number of *stability conditions*, i.e., invariant properties of higher-order processes with respect to their sets of private names. Stability conditions are essential in defining the hierarchy of SHO calculi based on polyadic communication, which is reported in Section 5.2.

5.1 Disjoint Forms

The *disjoint forms* for SHO processes are intended to capture the invariant structure of processes along communications, focusing on the private names shared among the participants. Their definition exploits *contexts*, that is, processes with a hole. We shall consider *multi-hole contexts*, that is, contexts with more than one hole. More precisely, a multi-hole context is n-ary if at most n different holes $[\cdot]_1, \ldots, [\cdot]_n$, occur in it. (A process is a 0-ary multi-hole context.) We will assume that any hole $[\cdot]_i$ can occur more than once in the context expression. Notions of free and bound names for contexts are as expected and denoted $\mathsf{bn}(\cdot)$ and $\mathsf{fn}(\cdot)$, respectively.

Definition 9 *The syntax of (guarded, multihole) contexts is defined as:*

$$C, C', \ldots ::= a(x).\, D \;\mid\; \bar{a}\langle D\rangle.\, D \;\mid\; C \parallel C \;\mid\; \nu r\, C \;\mid\; P$$
$$D, D', \ldots ::= [\cdot]_i \;\mid\; C \;\mid\; D \parallel D \;\mid\; \nu r\, D$$

Definition 10 (Disjoint Form) *Let $T \equiv \nu\widetilde{n}(P \parallel C[\widetilde{R}])$ be a* SHO^m *process where*

1. *$\widetilde{n}$ is a set of names such that $\widetilde{n} \subseteq \mathsf{fn}(P, \widetilde{R})$ and $\widetilde{n} \cap \mathsf{fn}(C) = \emptyset$;*
2. *C is a k-ary (guarded, multihole) context;*
3. *$\widetilde{R}$ contains k closed processes.*

*We then say that T is in k-*adic disjoint form with respect to *$\widetilde{n}$, $\widetilde{R}$,* and *P.*

A disjoint form captures the fact that processes $\widetilde{R}$ and context C do not share private names, i.e., that their sets of names are *disjoint*. A disjoint form can arise as the result of the communication between two processes that do not share private names; processes $\widetilde{R}$ would be then components of some process P_0 that evolved into P by communicating $\widetilde{R}$ to C. The above definition decrees an arbitrary (but fixed) arity for the context. We shall say that processes in such a form are in *n-adic disjoint form*, or NDF. By restricting the arity of the context, this general definition can be instantiated:

Definition 11 (Monadic and Zero-adic Disjoint Forms) *Let T be a process in disjoint form with respect to some $\widetilde{n}$, $\widetilde{R}$, and P. If $\mid \widetilde{R} \mid = 1$ then T is said to be in* monadic disjoint form *(or* MDF*) with respect to $\widetilde{n}$, R, and P. If $\mid \widetilde{R} \mid = 0$ then T is said to be in* zero-adic disjoint form *(or* ZDF*) with respect to $\widetilde{n}$ and P.*

Proposition 12 (Encodings preserve ZDFs) *Let $[\![\cdot]\!]$ be an encoding. If T is in ZDF with respect to some $\widetilde{n}$ and P then $[\![T]\!]$ is in ZDF with respect to $\widetilde{n}$ and $[\![P]\!]$.*

Properties of Disjoint Forms I: Stability Conditions. Stability conditions are central to capture the following insight: without name-passing, the set of names private to a process remains invariant along computations. Hence, two processes that interact respecting the stability conditions and do not share any private name will never be able to establish a private link. The distinction on internal actions is essential to define stability conditions for internal synchronizations (Lemma 13) and output actions (Lemma 14).

Lemma 13 *Let $T \equiv \nu\widetilde{n}\,(P \parallel C[\widetilde{R}])$ be a process in NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and P. If $T \xrightarrow{\tau} T'$ then: $T' \equiv \nu\widetilde{n}\,(P' \parallel C'[\widetilde{R}])$; $\mathsf{fn}(P', \widetilde{R}) \subseteq \mathsf{fn}(P, \widetilde{R})$ and $\mathsf{fn}(C') \subseteq \mathsf{fn}(C)$; T' is in NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and P'.*

The following results state that there is a stability condition for output actions, and the way in which a ZDF evolves after a *public* synchronization.

Lemma 14 *Let $T \equiv \nu\widetilde{n}\,(P \parallel C[\widetilde{R}])$ be a process in NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and P. If $T \xrightarrow{(\nu\widetilde{s})\overline{a}\langle Q\rangle} T'$ then: there exist P', C', $\widetilde{n}'$ such that $T' \equiv \nu\widetilde{n}'\,(P' \parallel C'[\widetilde{R}])$; $\mathsf{fn}(P', \widetilde{R}) \subseteq \mathsf{fn}(P, \widetilde{R})$, $\mathsf{fn}(C') \subseteq \mathsf{fn}(C)$ and $\widetilde{n}' \subseteq \widetilde{n}$ hold; T' is in NDF with respect to $\widetilde{n}'$, $\widetilde{R}$, and P'.*

Lemma 15 *Let T be a SHO^n process in ZDF with respect to $\widetilde{n}$ and P. Suppose $T \xrightarrow{a\tau} T'$ where $\xrightarrow{a\tau}$ is a public n-adic synchronization with $P \xrightarrow{(\nu\widetilde{n})\overline{a}\langle\widetilde{R}\rangle} P'$ as a premise. Then T' is in n-adic disjoint form with respect to $\widetilde{n}$, $\widetilde{R}$, and P'.*

Properties of Disjoint Forms II: Origin of Actions. We now give some properties regarding the order and origin of internal and output actions of processes in DFs.

Definition 16 *Let $T = \nu\widetilde{n}\,(A \parallel C[\widetilde{R}])$ be a process in NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and A. Suppose $T \xrightarrow{\alpha} T'$ for some action α.*

- *Let α be an output action. We say that α* originates in *A if $A \xrightarrow{\alpha'} A'$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$, and that α* originates in *C if $C[\widetilde{R}] \xrightarrow{\alpha'} C'[\widetilde{R}]$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$. In both cases, $\alpha = (\nu\widetilde{s})\alpha'$ for some $\widetilde{s}$.*
- *Let $\alpha = \tau$. We say that α* originates in *A if, for some $a \in \widetilde{n}$, $A \xrightarrow{a\tau} A'$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$, or if $A \xrightarrow{\tau} A'$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$. We say that α* originates in *C if $C[\widetilde{R}] \xrightarrow{\tau} C'[\widetilde{R}]$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$.*

The lemma below formalizes when two actions of a disjoint form can be *swapped*.

Lemma 17 (Swapping Lemma) *Let $T = \nu\widetilde{n}\,(A \parallel C[\widetilde{R}])$ be a process in NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and A. Consider two actions α and β that can be either output actions or internal synchronizations. Suppose that α originates in A, β originates in C, and that there exists a T' such that $T \xrightarrow{\alpha}\xrightarrow{\beta} T'$. Then $T \xrightarrow{\beta}\xrightarrow{\alpha} T'$ also holds, i.e., action β can be performed before α without affecting the final behavior.*

The converse of the Swapping Lemma does not hold: since an action β originated in C can enable an action α originated in A, these cannot be swapped. We now generalize the Swapping Lemma to a sequence of internal synchronizations and output actions.

Lemma 18 (Commuting Lemma) *Let $T = \nu\widetilde{n}\,(A \parallel C[\widetilde{R}])$ be a NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and A. Suppose $T \stackrel{\vec{\alpha}}{\Longrightarrow} T'$, where $\vec{\alpha}$ is a sequence of output actions and internal synchronizations. Let $\vec{\alpha}_C$ (resp. $\vec{\alpha}_A$) be its subsequence without actions originated in A (resp. C) or in its derivatives. Then, there exists a process T_1 such that*

1. $T \stackrel{\vec{\alpha}_C}{\Longrightarrow} T_1 \stackrel{\vec{\alpha}_A}{\Longrightarrow} T'$.
2. $T_1 \equiv \nu\widetilde{n}'\,(A \parallel \prod^{m_1} R_1 \parallel \cdots \parallel \prod^{m_k} R_k \parallel C'[\widetilde{R}])$, *for some* $m_1, \ldots, m_k \geq 0$.

5.2 A Hierarchy of Synchronous Higher-Order Process Calculi

Here we introduce a hierarchy of synchronous higher-order process calculi. The hierarchy is defined in terms of the impossibility of encoding SHO^n into SHO^{n-1}. We first present the result that sets the basic case of the hierarchy, namely that biadic process-passing cannot be encoded into monadic process-passing (Theorem 19). The proof exploits the notion of MDF and its associated stability conditions. We then state the general result, i.e., the impossibility of encoding SHO^{n+1} into SHO^n (Theorem 20).

Theorem 19 *There is no encoding of* SHO^2 *into* SHO^1.

Proof (Sketch). Assume, towards a contradiction, that an encoding $[\![\cdot]\!] : \mathrm{SHO}^2 \to \mathrm{SHO}^1$ does indeed exist. In what follows, we use i, j to range over $\{1, 2\}$, assuming that $i \neq j$. Assume processes $S_1 = \overline{s_1}$ and $S_2 = \overline{s_2}$. Consider the SHO^2 process $P = E^{(2)} \parallel F^{(2)}$, where $E^{(2)}$ and $F^{(2)}$ are defined as follows:

$$E^{(2)} = \overline{a}\langle S_1,\, S_2\rangle.\,\mathbf{0}$$
$$F^{(2)} = \nu b\,(a(x_1, x_2).\,(\overline{b}\langle\overline{b_1}.\,x_1\rangle.\,\mathbf{0} \parallel \overline{b}\langle\overline{b_2}.\,x_2\rangle.\,\mathbf{0} \parallel b(y_1).\,b(y_2).\,y_1))$$

where both $b_1, b_2 \notin \mathsf{fn}(E^{(2)})$ (with $b_1 \neq b_2$) and $s_1, s_2 \notin \mathsf{fn}(F^{(2)})$ (with $s_1 \neq s_2$) hold. P can perform only the following computations:

$$P \xrightarrow{a\tau} P_0 \xrightarrow{\tau}\xrightarrow{\tau} P_1 \xrightarrow{\overline{b_1}} P_2 \xrightarrow{\overline{s_1}} \mathbf{0} \tag{1}$$
$$P \xrightarrow{a\tau} P_0 \xrightarrow{\tau}\xrightarrow{\tau} P_1' \xrightarrow{\overline{b_2}} P_2' \xrightarrow{\overline{s_2}} \mathbf{0}\,. \tag{2}$$

In P_0 there is an internal choice on b, which has direct influence on: (i) the output action on b_i and (ii) the output action on s_i. Notice that each of these actions enables the following one, and that an output on b_i precludes the possibility of actions on b_j and s_j. The behavior of $[\![P]\!]$ —the encoding of P— can thus be described as follows:

$$[\![P]\!] \stackrel{a\tau}{\Longrightarrow}\approx [\![P_0]\!] \Rightarrow\approx [\![P_1]\!] \stackrel{\overline{b_1}}{\Longrightarrow}\approx [\![P_2]\!] \stackrel{\overline{s_1}}{\Longrightarrow}\approx \mathbf{0} \quad \text{and} \tag{3}$$
$$[\![P]\!] \stackrel{a\tau}{\Longrightarrow}\approx [\![P_0]\!] \Rightarrow\approx [\![P_1']\!] \stackrel{\overline{b_2}}{\Longrightarrow}\approx [\![P_2']\!] \stackrel{\overline{s_2}}{\Longrightarrow}\approx \mathbf{0}\,. \tag{4}$$

Actually, outputs may have parameters, but this does not change our results. The first (weak) transition, namely $[\![P]\!] \stackrel{a\tau}{\Longrightarrow}\approx [\![P_0]\!]$, is the same in both possibilities. For SHO[1] processes T, T', and T_0, it holds

$$[\![P]\!] \Rightarrow T \stackrel{a\tau}{\longrightarrow} T' \Rightarrow T_0 \approx [\![P_0]\!]\,. \tag{5}$$

By examining the disjoint forms in the processes in (5) and using the stability conditions (Prop. 12, Lemma 15, Lemma 13) one can show that T_0 is in MDF with respect to a set of names $\widetilde{l}$, and some processes R and A_0. Indeed, for some context C_0 (with private name b), we have that $T_0 = \nu\widetilde{l}\,(A_0 \parallel C_0[R])$. Notice that (5) ensures that $T_0 \approx [\![P_0]\!]$. Hence, by definition of $\approx$, T_0 should be able to match each action from $[\![P_0]\!]$ by performing either the sequence of actions given in (3) or the one in (4). Crucially, both (3) and (4) involve only internal synchronizations and output actions. Therefore, by Lemmas 13 and 14, every derivative of T_0 intended to mimic the behavior of $[\![P_0]\!]$ (and its derivatives) is in MDF with respect to R, some l_i and some A_i.

By analyzing the bisimilarity game between T_0 and $[\![P_0]\!]$, it is possible to infer the following behavior starting in T_0:

$$T_0 \Rightarrow T_1 \stackrel{\overline{b_1}}{\Longrightarrow} T_2 \stackrel{\overline{s_1}}{\Longrightarrow}\approx \mathbf{0} \quad \text{and} \tag{6}$$

$$T_0 \Rightarrow T_1' \stackrel{\overline{b_2}}{\Longrightarrow} T_2' \stackrel{\overline{s_2}}{\Longrightarrow}\approx \mathbf{0}. \tag{7}$$

where, by definition of $\approx$, $[\![P_i]\!] \approx T_i$ for $i \in \{0,1,2\}$ and $[\![P_j']\!] \approx T_j'$ for $j \in \{1,2\}$. Call C_2 and C_2' the derivatives of C_0 in T_2 and T_2', respectively. It is worth noticing that by conditions on names, output actions on s_1 and s_2 cannot originate in C_2 and C_2'.

The behavior of T_0 described in (6) and (7) can be equivalently described as $T_0 \stackrel{\alpha_1}{\Longrightarrow} \mathbf{0}$ and $T_0 \stackrel{\alpha_2}{\Longrightarrow} \mathbf{0}$, where α_1 contains outputs on b_1 and s_1, and α_2 contains outputs on b_2 and s_2, respectively. Using the Commuting Lemma (Lemma 18) on T_0, we know there exist processes T_1^*, and T_2^* such that

1. $T_1^* \equiv \nu\widetilde{n}_1\,(A_0 \parallel \prod^m R \parallel C_1^*[R])$ and $T_2^* \equiv \nu\widetilde{n}_2\,(A_0 \parallel \prod^{m'} R \parallel C_2^*[R])$, for some $m, m' \geq 0$. Recall that T_1^* and T_2^* are the results of performing every output action and internal synchronization originated in C_0. Since the encoding does not introduce divergence, we have that $C_1^*[R] \not\rightarrow$ and $C_2^*[R] \not\rightarrow$.
2. T_1^* (resp. T_2^*) can only perform an output action on s_1 (resp. s_2) and internal actions. Hence, we have that $T_1^* \Downarrow_{\overline{s_1}}$, $T_1^* \not\Downarrow_{\overline{s_2}}$ and $T_2^* \Downarrow_{\overline{s_2}}$, $T_2^* \not\Downarrow_{\overline{s_1}}$ should hold.

Item (1) allows to observe that the only difference between T_1^* and T_2^* is in the number of copies of R (the sets of restricted names are also different, but these do not involve the names we are interested in). This number has direct influence on performing an output action on s_1 or on s_2; as such, it has influence on the bisimulation game between $[\![P_2]\!]$ and T_2, and that between $[\![P_2']\!]$ and T_2'. More precisely, we have both $T_0 \stackrel{\overline{b_1}}{\Longrightarrow} T_1^*$ (with $T_1^* \Downarrow_{\overline{s_1}}$) and $T_0 \stackrel{\overline{b_2}}{\Longrightarrow} T_2^*$ (with $T_2^* \Downarrow_{\overline{s_2}}$). By assuming $m > m'$, we obtain that T_1^* corresponds to the composition of T_2^* and a number of copies of R. Hence, $T_1^* \Downarrow_{\overline{s_2}}$ and $T_0 \stackrel{\overline{b_1}}{\Longrightarrow} T^*$ with $T^* \downarrow_{\overline{s_2}}$. By operational correspondence, we have $P_0 \stackrel{\overline{b_1}}{\Longrightarrow} P'$ such that $T^* \Rightarrow T'$ with $T' \approx [\![P']\!]$. Notice that since the strong barb on s_2

in T^* cannot disappear (there is no reception on s_2), it is still in T'. Thus P' has a weak barb on s_2, which is impossible. □

The scheme used in the proof of Theorem 19 can be generalized for calculi with arbitrary polyadicity. Therefore we have the following.

Theorem 20 *For every $n > 1$, there is no encoding of* SHO^n *into* SHO^{n-1}.

Remark 21 (A hierarchy for *asynchronous* calculi) *Theorem 20 holds for calculi in* AHO *as well. The main structure of the proof is the same, but one needs to adapt the different pieces.*

6 The Expressive Power of Abstraction-Passing

In this section we show that abstraction-passing, i.e., the communication of parameterizable processes, is strictly more expressive than process-passing. We consider SHO^n_a, the extension of SHO^n with the communication of abstractions of order one, i.e., functions from processes to processes. The language of SHO^n_a processes is obtained by extending the syntax of SHO^n processes (Definition 1) in the following way:

$$P, Q, \ldots ::= \cdots \;\mid\; (x)P \;\mid\; P_1 \lfloor P_2 \rfloor$$

That is, we consider *abstractions* $(x)P$ and *applications* $P_1 \lfloor P_2 \rfloor$, that allow to apply an abstraction P_1 to an argument P_2. As usual, $(x_1) \ldots (x_n)P$ is abbreviated as $(x_1, \ldots, x_n)P$. The LTS of SHO^n_a extends that of SHO^n with the rule:

$$\textsc{App}\ \frac{}{(x)P \lfloor Q \rfloor \xrightarrow{\tau} P\{^{Q}/_{x}\}}\ .$$

Moreover, for SHO^n_a we rely on types and well-typed processes as in [9]; roughly speaking, the type system ensures consistent uses of application w.r.t. abstractions.

We now show that abstraction-passing increases the expressive power of pure process-passing in SHO. The result is based on the encoding below.

Definition 22 (Monadic abstraction-passing can encode polyadic communication) *The encoding $[\![\cdot]\!] : \mathrm{SHO}^2 \to \mathrm{SHO}^1_a$ is defined as:*

$$\begin{aligned}
[\![\overline{a}\langle P_1, P_2\rangle.\, R]\!] &= a(z).\, ([\![R]\!] \parallel \nu m\, n\, c\, (\overline{n} \parallel z\lfloor n.\, (\overline{c} \parallel \overline{m}) + m.\, ([\![P_1]\!] \parallel \overline{m})\rfloor \\
&\qquad\qquad \parallel c.\, z\lfloor [\![P_2]\!]\rfloor)) \\
[\![a(x_1, x_2).\, Q]\!] &= \nu b\, (\overline{a}\langle (y)\overline{b}\langle y\rangle\rangle \parallel b(x_1).\, (x_1 \parallel b(x_2).\, [\![Q]\!]))
\end{aligned}$$

where $[\![\cdot]\!]$ is an homomorphism for the other operators in SHO^2.

The encoding is correct, except that it does not preserve signatures (as inputs are translated into outputs and viceversa); a correct encoding can be written by resorting to abstractions with abstractions as parameters. This encoding leads also to the separation result below. The result is remarkable since it formalizes the fact that the expressive power of abstraction-passing is beyond any arity of polyadic communication.

Theorem 23 *For every $n, m > 1$, there is no encoding of SHO^n_a into SHO^m.*

Proof. Suppose, for the sake of contradiction, there is an encoding $\mathcal{A}[\![\cdot]\!] : \mathrm{SHO}^n_a \rightarrow \mathrm{SHO}^m$. Thanks to Def. 22, we have an encoding $\mathcal{B}[\![\cdot]\!] : \mathrm{SHO}^{m+1} \rightarrow \mathrm{SHO}^n_a$.[5] Now, the composition of two encodings is an encoding (Prop. 7), and so $(\mathcal{A} \cdot \mathcal{B})[\![\cdot]\!]$ is an encoding of SHO^{m+1} into SHO^m. However, by Theorem 20 such an encoding does not exist, and we reach a contradiction. □

7 Concluding Remarks

Summary. In first-order process calculi (a)synchronous and polyadic communication are well-understood mechanisms. In this paper, we have studied the expressiveness of these mechanisms in the context of *strictly higher-order* process calculi. Our results strengthen and complement expressiveness studies for higher-order process calculi in [12,13,11,9,14]. We have studied two families of higher-order process calculi: the first one, called AHO^n, is the asynchronous higher-order process calculus with n-adic communication; the second, called SHO^n, is the synchronous variant of AHO^n. Our first contribution was an *encodability* result of SHO^n into AHO^n. We then moved to analyze polyadic communication, and showed that in this case the absence of name-passing does entail a loss in expressiveness; this is represented by the impossibility of encoding SHO^n into SHO^{n-1}. This *non-encodability* result induces a *hierarchy* of higher-order process calculi based on the arity allowed in process-passing communications. This hierarchy holds for AHO as well. Finally, we showed an encoding of SHO^n into SHO^1 extended with abstraction-passing, and used it in our final contribution: the non-existence of an encoding of abstraction-passing into process-passing of any arity.

Related Work. Sangiorgi [9] proposed a hierarchy of HOπ fragments, based on the degree of the abstractions allowed (the level of arrow nesting in the type of the abstraction). This hierarchy is shown to match the expressiveness of a hierarchy of first-order calculi with only internal mobility. The argument that the hierarchy is strict is however intensional, counting the causal dependencies among names. In contrast, the hierarchy we consider here is given by the size of the tuples that can be passed around in polyadic communications. Also related are [12,13,11], in which expressiveness/decidability issues of HOCORE—roughly, the fragment of HOπ without restriction— are addressed.

Other works have used the distinction between internal and public synchronizations that we have used in the LTS for SHO. In [15], labels of internal actions are annotated with the name on which the synchronization occurs so as to define *located* semantics for the π-calculus; such semantics are then used to study concurrent semantics using a standard LTS. In the higher-order setting, [16] defines a variant of CHOCS in which synchronizations on so-called *activation channels* (i.e., the fresh channels used in the encoding of CHOCS into the π-calculus to trigger a copy of a process) are distinguished from other synchronizations. An LTS based on such a distinction is shown to be finitely branching; its induced bisimilarity is shown to coincide with bisimulation in CHOCS.

[5] The fact that the encoding does not preserve signatures can be overcome with a direct proof.

Future Work. It would be interesting to explore whether the hierarchy in Section 5 can be presented without resorting to the distinction on internal actions. This would require to formalize the concept of encoding robust with respect to interferences. Also, the result in Section 6 gives the base case of a hierarchy based on abstraction-passing. Here we have considered abstractions of order one; we plan to generalize such a result to abstractions of arbitrary order so as to define a hierarchy based on abstraction-passing.

Acknowledgments. We are grateful to J. Aranda and F. Valencia for discussions on the topics of this paper, and to the anonymous reviewers for their suggestions. This research was carried out while the second author was a PhD student at University of Bologna. This research has been partially supported by INRIA Equipe Associée BACON, by Project FP7- 231620 HATS, and by FCT / MCTES (CMU-PT/NGN44-2009-12).

References

1. Palamidessi, C.: Comparing the expressive power of the synchronous and asynchronous pi-calculi. Mathematical Structures in Computer Science **13**(5) (2003) 685–719
2. Nestmann, U.: What is a "good" encoding of guarded choice? Inf. Comput. **156**(1-2) (2000) 287–319 A preliminary version appeared in EXPRESS'97.
3. Cacciagrano, D., Corradini, F., Palamidessi, C.: Separation of synchronous and asynchronous communication via testing. Theor. Comput. Sci. **386**(3) (2007) 218–235
4. Boudol, G.: Asynchrony and the π-calculus (note). Technical report, Rapport de Recherche 1702, INRIA, Sophia-Antipolis (1992)
5. Honda, K., Tokoro, M.: An object calculus for asynchronous communication. In: Proc. of ECOOP. Volume 512 of Lecture Notes in Computer Science., Springer (1991) 133–147
6. Milner, R.: The Polyadic pi-Calculus: A Tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh (1991)
7. Quaglia, P., Walker, D.: Types and full abstraction for polyadic pi-calculus. Inf. Comput. **200**(2) (2005) 215–246
8. Yoshida, N.: Graph types for monadic mobile processes. In: Proc. of FSTTCS. Volume 1180 of Lecture Notes in Computer Science., Springer (1996) 371–386
9. Sangiorgi, D.: π-calculus, internal mobility and agent-passing calculi. Theor. Comput. Sci. **167**(2) (1996) 235–274
10. Gorla, D.: Towards a unified approach to encodability and separation results for process calculi. In: Proc. of CONCUR. Volume 5201 of Lecture Notes in Computer Science., Springer (2008) 492–507
11. Pérez, J.A.: Higher-Order Concurrency: Expressiveness and Decidability Results. PhD thesis, University of Bologna (2010) Draft in `www.japerez.phipages.com/`.
12. Lanese, I., Pérez, J.A., Sangiorgi, D., Schmitt, A.: On the expressiveness and decidability of higher-order process calculi. In: Proc. of LICS'08, IEEE Computer Society (2008) 145–155
13. Di Giusto, C., Pérez, J.A., Zavattaro, G.: On the expressiveness of forwarding in higher-order communication. In: Proc. of ICTAC. Volume 5684 of Lecture Notes in Computer Science., Springer (2009) 155–169
14. Sangiorgi, D.: Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. PhD thesis CST–99–93, University of Edinburgh, Dept. of Comp. Sci. (1992)
15. Lanese, I.: Concurrent and located synchronizations in *pi*-calculus. In: Proc. of SOFSEM. Volume 4362 of Lecture Notes in Computer Science., Springer (2007) 388–399
16. Amadio, R.M.: On the reduction of Chocs bisimulation to π-calculus bisimulation. In: Proc. of CONCUR. Volume 715 of Lecture Notes in Computer Science., Springer (1993) 112–126