

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

I. Lanese¹, J. A. Pérez¹, D. Sangiorgi¹, and A. Schmitt²

¹ Lab. FOCUS (Univ. of Bologna / INRIA)

² INRIA Grenoble - Rhône Alpes

Abstract. *Higher-order process calculi* are process calculi in which processes (more generally, values containing processes) can be communicated. We study the expressiveness of *strictly* higher-order process calculi, and focus on well-understood issues for first-order calculi but not in the higher-order setting, namely *synchronous* vs. *asynchronous* communication and *polyadic* vs. *monadic* communication. First, synchronous process passing is shown to be encodable into asynchronous process passing. Then we show that the absence of name passing leads to a hierarchy of higher-order process calculi based on the arity allowed in 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 (more generally, values containing 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. For first-order process calculi such as the π -calculus these are well-understood issues: several works (see, e.g., [1,2,3,4]) have studied the theory of the *asynchronous π -calculus* [5,6] and its relationship with the (synchronous) π -calculus. Also, the encoding of polyadic communication into monadic communication [7] is simple and very robust [8,9]. However, analogous studies are lacking for calculi in the higher-order setting.

We approach these questions in the context of $\text{HO}\pi$, a *strictly* higher-order process calculus (i.e., it has no name-passing features) [10]. $\text{HO}\pi$ is very expressive: it is Turing complete and several modelling idioms (disjoint choice, input-guarded replication, lists) 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 $\text{HO}\pi$ with polyadic communication (Section 2). SHO and AHO actually 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 of the process.³ The sharing of (private) names in a process-passing discipline is then limited: names can be *sent* as part of processes but they cannot be freely *used* by a recipient.

We begin by investigating the relationship between synchrony and asynchrony in process-passing calculi. Our first main result is an *encodability* result: an encoding of SHO^n into AHO^n (Section 4). This reveals a similarity between first-order and higher-order process calculi. Intuitively, a synchronous output is encoded by an asynchronous output that communicates both the communication object and its continuation. Based on this positive result, in Section 5 we move to examine the situation for polyadic communication in process-passing calculi. 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 main 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-order and higher-order process calculi.

In the final part of the paper (Section 6) we consider the expressiveness of the extension of SHO with *abstraction-passing*. An abstraction is an expression parametric on processes. The expressiveness of abstraction passing is then an issue specific to the higher-order setting. We consider SHO_a^n , 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_a^1 . In our last main contribution we use this result to show that there is no encoding of SHO_a^n into SHO^n .

The notion of encoding we use in this paper exploits a refined account of internal actions that is enforced by the labeled transition system (LTS) of SHO. In fact, the LTS distinguishes the internal actions that result from synchronizations on restricted names from those that result from synchronizations on public names. We consider only the former as internal actions; the latter are considered as visible actions. While the distinction between internal and public synchronizations might appear as rather demanding in the light of proposals for “good” encodings in the literature (e.g., [11]), we find it useful to concentrate in *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 on the two forms of internal transition is crucial in certain technical details of our proofs.

For the sake of space, in this presentation we omit most proofs and technical details. Full details and extended discussions can be found in [12], Chapter 6.

³ In this part of the discussion we understand process-passing that does *not* consider abstraction-passing, i.e., the communication of functions from processes to processes. As we shall see, the situation is rather different with abstraction-passing.

2 The Calculi

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

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

$$P, Q, \dots ::= a(\tilde{x}).P \mid \bar{a}(\tilde{Q}).P \mid P_1 \parallel P_2 \mid \nu r P \mid x \mid \mathbf{0}$$

Assuming standard notation and properties for tuples of syntactic elements, polyadicity in process passing is interpreted as expected: an output message $\bar{a}(\tilde{Q})$ sends the tuple of processes \tilde{Q} on name (or channel) a and then continues as P ; an input prefixed process $a(\tilde{x}).P$ can receive a tuple \tilde{Q} on name a and continue as $P\{\tilde{Q}/\tilde{x}\}$. In both cases, a is said to be the *subject* of the action. We sometimes write $|\tilde{x}|$ for the length of tuple \tilde{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 [10]. 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 ($\text{bn}(\cdot)$, $\text{fn}(\cdot)$, $\text{bv}(\cdot)$, and $\text{fv}(\cdot)$, resp.) are defined in the usual way: an input $a(\tilde{x}).P$ binds the free occurrences of variables in \tilde{x} in P ; similarly, $\nu r P$ binds the free occurrences of name r in P . We abbreviate $a(\tilde{x}).P$ as $a.P$ when none of the variables in \tilde{x} is in $\text{fv}(P)$; $\bar{a}(\tilde{\mathbf{0}}).P$ as $\bar{a}.P$; and $\bar{a}(Q).\mathbf{0}$ as $\bar{a}(Q)$. Notation $\prod^k P$ stands for k copies of process P in parallel.

The semantics for SHO is given in terms of the Labelled Transition System (LTS) given in Figure 1. It is defined in the usual way, with only one noteworthy difference: as hinted at in the introduction, 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 $\xrightarrow{a\tau}$. The distinction between internal and public synchronizations does not have behavioral consequences; it only represents a more refined standpoint of internal behavior. We thus have four kinds of transitions: in addition to internal and public synchronizations, there are input transitions $P \xrightarrow{a(\tilde{x})} P'$, and output transitions $P \xrightarrow{(\nu \tilde{y})\bar{a}(\tilde{Q})} P'$ (with extrusion of the tuple of names \tilde{y}), which have the expected meaning. We use α to range over actions. The subject of action α , denoted as $\text{sub}(\alpha)$, is defined as $\text{sub}(a(\tilde{x})) = a$, $\text{sub}(\bar{a}(\tilde{Q})) = a$, $\text{sub}(a\tau) = a$, and is undefined otherwise. Notions of bound and free names and variables extend to actions as expected. We sometimes use $\vec{\alpha}$ to denote a sequence of actions $\alpha_1, \dots, \alpha_n$. Weak transitions are defined in the usual way. We write \Rightarrow for the reflexive, transitive closure of $\xrightarrow{\tau}$. Given an action α , notation $\xRightarrow{\alpha}$ stands for $\Rightarrow \xrightarrow{\alpha} \Rightarrow$. Given a sequence $\vec{\alpha} = \alpha_1, \dots, \alpha_n$, we define $\xRightarrow{\vec{\alpha}}$ as $\xRightarrow{\alpha_1} \dots \xRightarrow{\alpha_n}$.

By varying the arity in polyadic communication, Definition 3 actually gives a *family* of higher-order process calculi. We have the following notational convention:

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

$$\begin{array}{c}
\text{INP } a(\tilde{x}).P \xrightarrow{a(\tilde{x})} P \qquad \text{OUT } \bar{a}(\tilde{Q}) \xrightarrow{\bar{a}(\tilde{Q})} \mathbf{0} \\
\text{ACT1 } \frac{P_1 \xrightarrow{\alpha} P'_1 \quad \text{bv}(\alpha) \cap \text{fv}(P_2) = \emptyset}{P_1 \parallel P_2 \xrightarrow{\alpha} P'_1 \parallel P_2} \\
\text{TAU1 } \frac{P_1 \xrightarrow{(\nu \tilde{y})\bar{a}(\tilde{P})} P'_1 \quad P_2 \xrightarrow{a(\tilde{x})} P'_2 \quad \tilde{y} \cap \text{fn}(P_2) = \emptyset}{P_1 \parallel P_2 \xrightarrow{a\tau} \nu \tilde{y} (P'_1 \parallel P'_2 \{ \tilde{P} / \tilde{x} \})} \\
\text{RES } \frac{P \xrightarrow{\alpha} P' \quad r \notin \text{n}(\alpha)}{\nu r P \xrightarrow{\alpha} \nu r P'} \\
\text{INTRES } \frac{P \xrightarrow{a\tau} P'}{\nu a P \xrightarrow{\tau} \nu a P} \\
\text{OPEN } \frac{P \xrightarrow{(\nu \tilde{y})\bar{a}(\tilde{P}'')} P' \quad x \neq a, x \in \text{fn}(\tilde{P}'') - \tilde{y}}{\nu x P \xrightarrow{(\nu x \tilde{y})\bar{a}(\tilde{P}'')} P'}
\end{array}$$

Fig. 1. The LTS of SHO. Rules ACT2 and TAU2, the symmetric counterparts of rules ACT1 and TAU1 are omitted.

Definition 3 *The language of AHO processes corresponds to the fragment of SHO processes where output actions have no continuations. All the definitions extend to AHO processes in the expected way; hence AHOⁿ corresponds to the asynchronous calculus with n-adic communication.*

The following definition is standard.

Definition 4 (Strong and Weak Barbs) *Given a process P and a name a , we write*

- $P \downarrow_a$ —a strong input barb— *if P can perform an input action with subject a ;*
- $P \downarrow_{\bar{a}}$ —a strong output barb— *if P can perform an output action with subject a .*

Given $\mu \in \{a, \bar{a}\}$, we define a weak barb $P \Downarrow_\mu$ if, for some P' , $P \Rightarrow P' \downarrow_\mu$.

3 The Notion of Encoding

The following definition of encoding is inspired by that of Gorla [11], who proposed five criteria a “good encoding” should satisfy.

Definition 5 *A language \mathcal{L} is defined as:*

- a set of processes \mathcal{P} ;
- 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.
- a weak behavioral equivalence \approx (i.e., a behavioral equivalence that abstracts from internal actions in \mathcal{A}).

Definition 6 (Translation) *Given a source language $\mathcal{L}_s = (\mathcal{P}_s, \longrightarrow_s, \approx_s)$ and a target language $\mathcal{L}_t = (\mathcal{P}_t, \longrightarrow_t, \approx_t)$, a translation of \mathcal{L}_s into \mathcal{L}_t is a function $\llbracket \cdot \rrbracket : \mathcal{P}_s \rightarrow \mathcal{P}_t$.*

Our interest is in translations that respect both syntactic and semantic conditions.

Definition 7 (Syntactic Conditions) Let $\llbracket \cdot \rrbracket : \mathcal{P}_s \rightarrow \mathcal{P}_t$ be a translation of \mathcal{L}_s into \mathcal{L}_t . We say that $\llbracket \cdot \rrbracket$ is

1. *Compositional*: if for every k -ary operator op of \mathcal{L}_s and for all S_1, \dots, S_k with $\text{fn}(S_1, \dots, S_k) = N$, then there exists a k -ary context $C_{\text{op}}^N \in \mathcal{P}_t$ such that

$$\llbracket \text{op}(S_1, \dots, S_k) \rrbracket = C_{\text{op}}^N[\llbracket S_1 \rrbracket, \dots, \llbracket S_k \rrbracket].$$

2. *Name invariant*: if $\llbracket \sigma(P) \rrbracket = \sigma(\llbracket P \rrbracket)$, for any injective renaming of names σ .

Definition 8 (Semantic Conditions) Let $\llbracket \cdot \rrbracket : \mathcal{P}_s \rightarrow \mathcal{P}_t$ be a translation of \mathcal{L}_s into \mathcal{L}_t . We say that $\llbracket \cdot \rrbracket$ is operational corresponding if the following properties hold:

1. *Completeness/Preservation*: For every $S, S' \in \mathcal{P}_s$ and $\alpha \in \mathcal{A}_s$ such that $S \xrightarrow{\alpha}_s S'$, it holds that $\llbracket S \rrbracket \xrightarrow{\beta}_t \approx_t \llbracket S' \rrbracket$, where $\beta \in \mathcal{A}_t$ and $\text{sub}(\alpha) = \text{sub}(\beta)$.
2. *Soundness/Reflection*: For every $S \in \mathcal{P}_s$, $T \in \mathcal{P}_t$, $\beta \in \mathcal{A}_t$ such that $\llbracket S \rrbracket \xrightarrow{\beta}_t T$ there exists an $S' \in \mathcal{P}_s$ and an action $\alpha \in \mathcal{A}_s$ such that $S \xrightarrow{\alpha}_s S'$, $T \Rightarrow_{\approx_t} \llbracket S' \rrbracket$, and $\text{sub}(\alpha) = \text{sub}(\beta)$.

Furthermore, we shall require adequacy: if $P \approx_s Q$ then $\llbracket P \rrbracket \approx_t \llbracket Q \rrbracket$.

Adequacy is necessary as we make no assumptions on the nature of \approx_s and \approx_t . It is also crucial to obtain *composability* of encodings (see Prop. 10).

Definition 9 We call encoding any translation that satisfies both the syntactic conditions in Definition 7 and the semantic conditions in Definition 8.

Proposition 10 (Some properties of encodings) Let $\llbracket \cdot \rrbracket$ be an encoding of \mathcal{L}_s into \mathcal{L}_t . Then $\llbracket \cdot \rrbracket$ satisfies:

Barb preservation For every $S \in \mathcal{P}_s$ it holds that $S \Downarrow_{\bar{a}}$ (resp. $S \Downarrow_a$) if and only if $\llbracket S \rrbracket \Downarrow_{\bar{a}}$ (resp. $\llbracket S \rrbracket \Downarrow_a$).

Preservation of free names Let a be a name. If $a \in \text{fn}(P)$ then also $a \in \text{fn}(\llbracket P \rrbracket)$.

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

4 An Encodability Result for Synchronous Communication

We begin by studying the relationship between synchronous and asynchronous communication. The main result of this section is an encoding of SHO^n into AHO^n .

A naive encoding would simply consist in sending both the communication object and the continuation of the output action in a single synchronization. The continuation is sent explicitly as a parameter, and so a synchronous calculus with polyadicity n would have to be encoded into an asynchronous calculus with polyadicity $n + 1$. To illustrate this, consider the naive encoding of SHO^1 into AHO^2 :

$$\llbracket \bar{a}\langle P \rangle . S \rrbracket = \bar{a}\langle \llbracket P \rrbracket, \llbracket S \rrbracket \rangle \quad \llbracket a(x) . R \rrbracket = a(x, y) . (y \parallel \llbracket R \rrbracket)$$

where $\llbracket \cdot \rrbracket$ is an homomorphism for the other operators in SHO¹. This encoding allows to appreciate how in the higher-order setting the synchronous/asynchronous distinction can be considered as a particular case of the polyadic/monadic distinction. Notice that the fact that the continuation is supposed to be executed only once is crucial for the simplicity of the encoding.

Interestingly, asynchronous process-passing is expressive enough so as to encode synchronous communication of the *same arity*. 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:

$$\llbracket \bar{a}\langle P \rangle . S \rrbracket = \nu k, l (\bar{a}\langle k . (\llbracket P \rrbracket \parallel \bar{k}) + l . (\llbracket S \rrbracket \parallel \bar{k}) \rangle \parallel \bar{l}) \quad \llbracket a(x) . R \rrbracket = a(x) . (x \parallel \llbracket R \rrbracket)$$

where “+” stands for the encoding of disjoint choice proposed for HOCORE [13]; k, l are names not in $\text{fn}(P, S)$; and $\llbracket \cdot \rrbracket$ is an homomorphism for the other operators in SHO¹. The synchronous output action is thus encoded by sending a guarded, disjoint choice between the encoding of the communication object and the encoding of the continuation of the output. The encoding exploits the fact that the continuation should be executed exactly once, while the communication object can be executed zero or more times. Notice that there is only one copy of the trigger that executes the encoding of the continuation (denoted \bar{l} in the encoding above), which guarantees that it is executed exactly once. This can only occur after the synchronization has taken place, thus ensuring a correct encoding of synchronous communication. Notice that \bar{l} releases both the encoding of the continuation and a trigger for executing the encoding of the communication object (denoted \bar{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 \bar{k} is always available. This idea can be generalized to encode synchronous calculi of arbitrary polyadicity as follows:

Definition 11 (Synchronous into Asynchronous) *For some $n > 0$, the encoding of SHOⁿ into AHOⁿ is defined as follows:*

$$\begin{aligned} \llbracket \bar{a}\langle P_1, \dots, P_n \rangle . S \rrbracket &= \nu k, l (\bar{a}\langle \llbracket P_1 \rrbracket, \dots, \llbracket P_{n-1} \rrbracket, T_{k,l}[\llbracket P_n \rrbracket, \llbracket S \rrbracket] \rangle \parallel \bar{l}) \\ \llbracket a(x_1, \dots, x_n) . R \rrbracket &= a(x_1, \dots, x_n) . (x_n \parallel \llbracket R \rrbracket) \end{aligned}$$

with $T_{k,l}[M_1, M_2] \stackrel{def}{=} k . (M_1 \parallel \bar{k}) + l . (M_2 \parallel \bar{k})$ and where $\{k, l\} \cap \text{fn}(P_1, \dots, P_n, S) = \emptyset$, and $\llbracket \cdot \rrbracket$ is an homomorphism for the other operators in SHOⁿ.

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 *distinguished 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. The hierarchy of SHO calculi based on polyadic communication is reported in Section 5.2. Stability conditions are essential in defining the hierarchy.

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 $[\cdot]_1, \dots, [\cdot]_n$, 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 $\text{bn}(\cdot)$ and $\text{fn}(\cdot)$, respectively.

Definition 12 *The syntax of (guarded, multihole) contexts is defined as:*

$$\begin{aligned} C, C', \dots &::= a(x). D \mid \bar{a}(D). D \\ D, D', \dots &::= [\cdot]_i \mid P \mid C \mid D \parallel D \mid \nu r D \end{aligned}$$

Definition 13 (Disjoint Form) *Let $T \equiv \nu \tilde{n}(P \parallel C[\tilde{R}])$ be a SHO^m process where*

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

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

Intuitively, a disjoint form captures the fact that processes \tilde{R} and context C do not share private names, i.e., that their sets of names are *disjoint*. The intuition can be extended to see a disjoint form as the result of the communication of two processes that do not share private names; processes \tilde{R} would be then components of some process P_0 that evolved into P by communicating \tilde{R} to C . The above definition decrees an arbitrary arity for the context. We shall sometimes 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 14 (Monadic Disjoint Form, MDF) *Assume a process T that is in disjoint form with respect to some \tilde{n} , \tilde{R} , and P . If $|\tilde{R}| = 1$ then T is said to be in monadic disjoint form (or MDF) with respect to \tilde{n} , R , and P .*

Recall that even if MDFs have monadic contexts, the content of the hole (i.e., the single process R) can appear more than once in the process. It could even be the case the content does not appear at all. This is a special case of MDF, as we define below:

Definition 15 (Zero-adic Disjoint Form, ZDF) *Let $T \equiv \nu \tilde{n}(P \parallel C[R])$ be in MDF with respect to \tilde{n} , R , and P . If $C[R] \neq \mathbf{0}$ and $R = \mathbf{0}$ then T is said to be in zero-adic disjoint form (ZDF) with respect to \tilde{n} and P .*

Notice that if T is in ZDF, it can be rewritten as $T \equiv \nu_{\tilde{n}_1} P \parallel \nu_{\tilde{n}_2} Q$, for some $Q \equiv C[\mathbf{0}]$ and for some disjoint sets of names \tilde{n}_1 and \tilde{n}_2 such that both $\tilde{n} = \tilde{n}_1 \cup \tilde{n}_2$ and $\tilde{n}_1 \cap \tilde{n}_2 = \emptyset$ hold.

Proposition 16 (Encodings preserve ZDFs) *Let $\llbracket \cdot \rrbracket$ be an encoding as in Def. 9. If T is in ZDF with respect to some \tilde{n} and P then $\llbracket T \rrbracket$ is in ZDF with respect to \tilde{n} and $\llbracket P \rrbracket$.*

Properties of Disjoint Forms I: Stability Conditions. Stability conditions are central to capture the following insight: without name-passing, the set of names that are private to a given process remains invariant along computations. As such, two processes that interact respecting the stability conditions and do not share any private name will never be able to establish a private link on it. The distinction on internal actions enforced by the LTS of SHO is essential to define a stability condition on internal synchronizations (Lemma 17). A stability condition for output actions holds too (Lemma 19). We discuss properties that apply to arbitrary NDFs; for the sake of readability, however, in what follows we restrict ourselves to the case of MDFs.

Lemma 17 *Let $T \equiv \nu_{\tilde{n}} (P \parallel C[\tilde{R}])$ be a process in NDF with respect to \tilde{n} , \tilde{R} , and P . If $T \xrightarrow{\tau} T'$ then: $T' \equiv \nu_{\tilde{n}} (P' \parallel C'[\tilde{R}])$; $\text{fn}(P', \tilde{R}) \subseteq \text{fn}(P, \tilde{R})$ and $\text{fn}(C') \subseteq \text{fn}(C)$; T' is in NDF with respect to \tilde{n} , \tilde{R} , and P' .*

Corollary 18 *Let T be a process in ZDF with respect to some \tilde{n} and P . If $T \xrightarrow{\tau} T'$, then T' is in ZDF with respect to \tilde{n} and P too.*

The stability condition for output actions holds for those output actions that do not involve extrusion of names:

Lemma 19 *Let $T \equiv \nu_{\tilde{n}} (P \parallel C[\tilde{R}])$ be a process in NDF with respect to \tilde{n} , \tilde{R} , and P . If $T \xrightarrow{\bar{a}(Q)} T'$ then: there exist P' and C' so that $T' \equiv \nu_{\tilde{n}} (P' \parallel C'[\tilde{R}])$; both $\text{fn}(P', \tilde{R}) \subseteq \text{fn}(P, \tilde{R})$ and $\text{fn}(C') \subseteq \text{fn}(C)$ hold; T' is in MDF with respect to \tilde{n} , \tilde{R} , and P' .*

The following lemma formalizes the way in which the structure of a ZDF evolves after a *public* synchronization.

Lemma 20 *Let T be a SHO^n process in ZDF with respect to \tilde{n} and P . Suppose $T \xrightarrow{a\tau} T'$ where $\xrightarrow{a\tau}$ is a public n -adic synchronization. Then T' is in n -adic disjoint form with respect to \tilde{n} , some \tilde{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 21 *Let $T = \nu_{\tilde{n}} (A \parallel C[\tilde{R}])$ be an NDF with respect to \tilde{n} , \tilde{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 $C[\tilde{R}] \xrightarrow{\alpha} C'[\tilde{R}]$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$.

- Let $\alpha = \tau$. We say that α originates in A if, for some $a \in \tilde{n}$, $A \xrightarrow{a\tau} A'$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$, and that α originates in C if $C[\tilde{R}] \xrightarrow{\tau} C'[\tilde{R}]$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$.

The following lemma states the conditions under which two actions of a disjoint form can be safely *swapped*. This is useful to formalize certain causal dependencies between the components of a disjoint form.

Lemma 22 (Swapping Lemma) *Let $T = \nu\tilde{n}(A \parallel C[\tilde{R}])$ be an NDF with respect to \tilde{n} , \tilde{R} , and A . Consider two actions α and β that can be either an output action or an internal synchronization. Suppose that α originates in A , β originates in C , and that there exists a T' such that $T \xrightarrow{\alpha\beta} T'$. Then $T \xrightarrow{\beta\alpha} T'$ also holds, i.e., action β can be performed before α without affecting the final behavior.*

Notice that the converse of the Swapping Lemma does not hold: since an action β originated in C can enable an action α originated in A (e.g., an action enabled by an extra copy of R), these cannot be swapped. We now generalize the Swapping Lemma to a sequence of internal synchronizations and output actions.

Lemma 23 (Commuting Lemma) *Let $T = \nu\tilde{n}(A \parallel C[\tilde{R}])$ be a NDF with respect to \tilde{n} , \tilde{R} , and A . Suppose $T \xrightarrow{\vec{\alpha}} T'$, where $\vec{\alpha}$ is a sequence of output actions and internal synchronizations only. Let $\vec{\alpha}_C$ (resp. $\vec{\alpha}_A$) be the sequence of actions that is exactly as $\vec{\alpha}$ but in which actions originated in A (resp. C) or its derivatives are not included. Then, there exists a process T_1 such that*

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

5.2 A Hierarchy of Synchronous Higher-Order Process Calculi

We present an expressiveness hierarchy for the higher-order process calculi in the family given by SHO. The hierarchy is defined in terms of the impossibility of encoding SHO^n into SHO^{n-1} . We begin by showing the impossibility result that sets the basic case of the hierarchy, namely that biadic process passing cannot be encoded into monadic process passing (Lemma 24). 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 (Lemma 25).

Lemma 24 *There is no encoding of SHO^2 into SHO^1 .*

Proof (Sketch). Assume, towards a contradiction, that an encoding $\llbracket \cdot \rrbracket : \text{SHO}^2 \rightarrow \text{SHO}^1$ does indeed exist. In what follows, we use i, j to range over $\{1, 2\}$, assuming that $i \neq j$. Assume processes $S_1 = \overline{m_1} \parallel m_1.\overline{s_1}$ and $S_2 = \overline{m_2} \parallel m_2.\overline{s_2}$. Consider the SHO^2 process $P = E^{(2)} \parallel F^{(2)}$, where $E^{(2)}$ and $F^{(2)}$ are defined as follows:

$$\begin{aligned} E^{(2)} &= \nu m_1, m_2 (\overline{a}(S_1, S_2). \mathbf{0}) \\ F^{(2)} &= \nu b(a(x_1, x_2). (\overline{b}(\overline{b_1}. x_1). \mathbf{0} \parallel \overline{b}(\overline{b_2}. x_2). \mathbf{0} \parallel b(y_1). b(y_2). y_1)) \end{aligned}$$

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

$$P \xrightarrow{a\tau} P_0 \xrightarrow{\tau} \xrightarrow{\tau} P_1 \xrightarrow{\overline{b_1}} P_2 \xrightarrow{\tau} \xrightarrow{\overline{s_1}} \mathbf{0} \quad (1)$$

$$P \xrightarrow{a\tau} P_0 \xrightarrow{\tau} \xrightarrow{\tau} P'_1 \xrightarrow{\overline{b_2}} P'_2 \xrightarrow{\tau} \xrightarrow{\overline{s_2}} \mathbf{0}. \quad (2)$$

In P_0 there is an internal choice on b , which has direct influence on: (i) the output action on b_i , (ii) the internal synchronization on m_i , and (iii) 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, m_j , and s_j . Based on (1) and (2) above the behavior of $\llbracket P \rrbracket$ —the encoding of P — can be described as follows:

$$\llbracket P \rrbracket \xrightarrow{a\tau} \approx \llbracket P_0 \rrbracket \Rightarrow \approx \llbracket P_1 \rrbracket \xrightarrow{\overline{b_1}} \approx \llbracket P_2 \rrbracket \xrightarrow{\overline{s_1}} \approx \mathbf{0} \quad \text{and} \quad (3)$$

$$\llbracket P \rrbracket \xrightarrow{a\tau} \approx \llbracket P_0 \rrbracket \Rightarrow \approx \llbracket P'_1 \rrbracket \xrightarrow{\overline{b_2}} \approx \llbracket P'_2 \rrbracket \xrightarrow{\overline{s_2}} \approx \mathbf{0}. \quad (4)$$

The first (weak) transition, namely $\llbracket P \rrbracket \xrightarrow{a\tau} \approx \llbracket P_0 \rrbracket$ is the same in both possibilities. Let us analyze it, by relying on Definition 4. For SHO^1 processes T, T' , and T_0 , it holds

$$\llbracket P \rrbracket \Rightarrow T \xrightarrow{a\tau} T' \Rightarrow T_0 \approx \llbracket P_0 \rrbracket. \quad (5)$$

By examining the distinguished forms in the processes in (5) and exploiting the stability conditions (Proposition 16, Corollary 18, Lemma 20, Lemma 17) it can be shown that T_0 is in MDF with respect to $\{m_1, m_2\}$, some $R \neq \mathbf{0}$, and some process A_0 . Indeed, for some context C_0 (with private name b), we have that $T_0 = \nu m_1, m_2 (A_0 \parallel C_0[R])$. Notice that (5) ensures that process $T_0 \approx \llbracket P_0 \rrbracket$. Hence, by definition of \approx , T_0 should be able to match each action possible from $\llbracket P_0 \rrbracket$ 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 17 and 19, every derivative of T_0 intended to mimic the behavior of $\llbracket P_0 \rrbracket$ (and its derivatives) is a process in MDF with respect to $\{m_1, m_2\}$, R , and some A_i .

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

$$T_0 \Rightarrow T_1 \xrightarrow{\overline{b_1}} T_2 \xrightarrow{\overline{s_1}} \approx \mathbf{0} \quad \text{and} \quad (6)$$

$$T_0 \Rightarrow T'_1 \xrightarrow{\overline{b_2}} T'_2 \xrightarrow{\overline{s_2}} \approx \mathbf{0}. \quad (7)$$

where, by definition of \approx , $\llbracket P_i \rrbracket \approx T_i$ for $i \in \{0, 1, 2\}$ and $\llbracket P'_j \rrbracket \approx T'_j$ 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 $T_0 \xrightarrow{\alpha_1} \mathbf{0}$ and $T_0 \xrightarrow{\alpha_2} \mathbf{0}$, 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 23) on T_0 , we know there exist processes T_1^* , and T_2^* such that

1. $T_1^* \equiv \nu \tilde{n} (A_0 \parallel \prod^{m'} R \parallel C_1^*[R])$ and $T_2^* \equiv \nu \tilde{n} (A_0 \parallel \prod^{m'} R \parallel C_2^*[R])$, for some $m, m' \geq 0$. Recall that these processes are obtained by performing every action originated in C_0 (which are only output actions and internal synchronizations); as a result, $C_1^*[R] \not\sim$ and $C_2^*[R] \not\sim$.

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 $T_1^* \Downarrow_{s_1}, T_1^* \not\Downarrow_{s_2}$ and $T_2^* \Downarrow_{s_2}, T_2^* \not\Downarrow_{s_1}$ should hold.

From item (1) above it is easy to observe that the only difference between T_1^* and T_2^* is in m and m' , the number of copies of R released as a result of executing first all actions originating in C_0 . We then find that the number of copies of R has direct influence on performing an output action on s_1 or on s_2 ; in turn, this has influence on the bisimulation game between $\llbracket P_2 \rrbracket$ and T_2 , and that between $\llbracket P_2' \rrbracket$ and T_2' . More precisely, we have both $T_0 \xrightarrow{\bar{b}_1} T_1^*$ (with $T_1^* \Downarrow_{s_1}$) and $T_0 \xrightarrow{\bar{b}_2} T_2^*$ (with $T_2^* \Downarrow_{s_2}$). By assuming $m > m'$, we obtain that T_1^* corresponds to the composition of T_2^* and a number of copies of R . Hence, $T_1^* \Downarrow_{s_2}$ and $T_0 \xrightarrow{\bar{b}_1} T^*$ with $T^* \Downarrow_{s_2}$. By operational correspondence, we have $P_0 \xrightarrow{\bar{b}_1} P'$ such that $T^* \Rightarrow T'$ with $T' = \llbracket P' \rrbracket$. Notice that since the strong barb on s_2 in T^* cannot disappear (there is no reception on s_2 , because otherwise we would observe a corresponding input barb), it is still in T' . Thus P' has a weak barb on s_2 , which is impossible. \square

The scheme used in the proof of Lemma 24 can be generalized for calculi with arbitrary polyadicity. Therefore we have the following.

Lemma 25 *For every $n > 1$, there is no encoding of SHO^n into SHO^{n-1} .*

Remark 26 (A hierarchy for asynchronous calculi) *Lemma 25 holds for calculi in AHO as well. In fact, a detailed proof would consist in adapting the definition of guarded contexts (Definition 12), the stability lemmas (Lemmas 17 and 19), the conditions under which the Swapping Lemma holds (Lemma 22), and the counterexample used in Lemma 24. Roughly speaking, there are no substantial differences between the synchronous and the asynchronous case: having one less prefix does not change the structure of the proof; the definition of disjoint form becomes somewhat weaker, as copies of the process inside context would be only released after an input action.*

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_a^n , the extension of SHO^n with the communication of abstractions of order one, i.e., functions from processes into processes. The language of SHO_a^n processes is obtained by extending the syntax of SHO^n processes (Definition (1)) in the following way:

$$P, Q, \dots ::= \dots \mid (x)P \mid P_1[P_2]$$

That is, we consider abstractions of the form $(x)P$ and *applications* of the form $P_1[P_2]$, that allows to assign an argument P_2 to an abstraction P_1 . As usual, $(x_1) \dots (x_n)P$ is abbreviated as $(x_1, \dots, x_n)P$. The operational semantics of SHO_a^n is that of SHO^n , extended with the following rule:

$$\text{APP} \frac{}{(x)P[Q] \xrightarrow{\tau} P\{Q/x\}}.$$

Moreover, for SHO_a^n we rely on notions of types as in [10], and consider only well-typed processes.

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 27 (Monadic abstraction-passing can encode polyadic communication)

The encoding $\llbracket \cdot \rrbracket : \text{SHO}^2 \rightarrow \text{SHO}_a^1$ is defined as:

$$\begin{aligned} \llbracket \bar{a}\langle P_1, P_2 \rangle . R \rrbracket &= a(z). (\llbracket R \rrbracket \parallel \nu m, n, c (\bar{n} \parallel z \lfloor n. (\bar{c} \parallel \bar{m}) + m. (\llbracket P_1 \rrbracket \parallel \bar{m}) \rrbracket \parallel c. z \lfloor \llbracket P_2 \rrbracket \rrbracket)) \\ \llbracket a(x_1, x_2) . Q \rrbracket &= \nu b (\bar{a}\langle (y) \bar{b}\langle y \rangle \rrbracket \parallel b(x_1). (x_1 \parallel b(x_2). \llbracket Q \rrbracket)) \end{aligned}$$

where $\llbracket \cdot \rrbracket$ is an homomorphism for the other operators in SHO^2 .

Definition 27 can be generalized so as to obtain an encoding $\llbracket \cdot \rrbracket : \text{SHO}^n \rightarrow \text{SHO}_a^1$, for any $n > 1$. This encoding leads to the following separation result:

Lemma 28 *There is no encoding of SHO_a^n into SHO^n .*

Proof. Let us just consider the case $n = 1$; the other cases are similar. Suppose, for the sake of contradiction, there is an encoding $\mathcal{A}[\llbracket \cdot \rrbracket] : \text{SHO}_a^1 \rightarrow \text{SHO}^1$. By Definition 27, we know there is an encoding $\mathcal{B}[\llbracket \cdot \rrbracket] : \text{SHO}^2 \rightarrow \text{SHO}_a^1$. Since the composition of two encodings is an encoding (Proposition 10), this means that $(\mathcal{A} \cdot \mathcal{B})[\llbracket \cdot \rrbracket]$ is an encoding of SHO^2 into SHO^1 . However, by Lemma 25 we know such an encoding does not exist, and we reach a contradiction.

7 Concluding Remarks

Summary. In first-order process calculi such as the π -calculus (a)synchronous and polyadic communication are well-understood mechanisms. In this paper we have studied the expressiveness of *strictly* higher-order process calculi, and have focused on synchronous and polyadic communication. Our results strengthen and complement expressiveness studies for higher-order process calculi in [13,14,12,10,15]. 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 the case of polyadicity the absence of name-passing does entail a loss in expressiveness; this is represented by the non-existence of an encoding of SHO^n into SHO^{n-1} . This *non-encodability* result determines 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 such a result to prove our final contribution: the non-existence of an encoding of abstraction passing into process passing of any arity. The notion of encoding used in our results exploits a distinction on internal actions which is useful to focus on encodings robust to interferences from arbitrary contexts.

Related Work. The most related work is [10], which proposes a hierarchy of $\text{HO}\pi$ fragments, based on the degree of the abstractions allowed (the level of arrow nesting in the type of the abstraction). It is shown that this hierarchy matches 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 to [10], the hierarchy we consider here is given by the size of the tuples that can be passed around in polyadic communications. Also related are [13,14,12] in which expressiveness and decidability issues of HOCORE —the fragment of $\text{HO}\pi$ without restriction— are addressed.

The distinction between internal and public synchronizations here proposed for our notion of encoding has been used and/or proposed in other contexts. In [16] labels of internal actions are annotated with the name on which synchronization occurs so as to define *located* semantics which are then used to study concurrent semantics for the π -calculus using standard labeled transition systems. In the higher-order setting, [17] obtains a finitely-branching bisimilarity for CHOCS by means of a reduction into bisimulation for a variant of the π -calculus. In such a variant, processes are only allowed to exchange names of *activation channels* (i.e., the channels that trigger a copy of a process in the representation of higher-order communication with first-order one). The desired finitely-branching bisimilarity is obtained by relying on a labeled transition system in which synchronizations on activation channels are distinguished.

Future Work. An immediate issue is to explore whether the hierarchy of expressiveness for polyadic communication presented in Section 5 holds with a definition of encoding without the distinction on internal actions. This would probably require substantial modifications in our proofs, that crucially rely on the distinction on the two forms of internal actions. Other approaches to the separation result, in particular those based on experiments and divergence, as in testing semantics [18], appear worth exploring. Also, the separation result in Section 6 is only the starting point for an expressiveness hierarchy based on abstraction passing. We have only considered abstractions of order one; we plan to generalize such a result to abstraction of arbitrary orders so as to define a complete expressiveness hierarchy based on abstraction passing.

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. Amadio, R.M., Castellani, I., Sangiorgi, D.: On bisimulations for the asynchronous pi-calculus. *Theor. Comput. Sci.* **195**(2) (1998) 291–324
5. Boudol, G.: Asynchrony and the π -calculus (note). Technical report, Rapport de Recherche 1702, INRIA, Sophia-Antipolis (1992)
6. 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
7. Milner, R.: The Polyadic pi-Calculus: A Tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh (1991)

8. Quaglia, P., Walker, D.: Types and full abstraction for polyadic pi-calculus. *Inf. Comput.* **200**(2) (2005) 215–246
9. Yoshida, N.: Graph types for monadic mobile processes. In: *Proc. of FSTTCS*. Volume 1180 of *Lecture Notes in Computer Science.*, Springer (1996) 371–386
10. Sangiorgi, D.: π -calculus, internal mobility and agent-passing calculi. *Theor. Comput. Sci.* **167**(2) (1996) 235–274
11. 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
12. Pérez, J.A.: Higher-Order Concurrency: Expressiveness and Decidability Results. PhD thesis, University of Bologna (2010) Draft in www.cs.unibo.it/~perez/hocore.
13. 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
14. 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
15. 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)
16. 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
17. Amadio, R.M.: On the reduction of chocs bisimulation to pi-calculus bisimulation. In: *Proc. of CONCUR*. Volume 715 of *Lecture Notes in Computer Science.*, Springer (1993) 112–126
18. Nicola, R.D., Hennessy, M.: Testing equivalences for processes. *Theor. Comput. Sci.* **34** (1984) 83–133