The Concurrency Column

BY

Luca Aceto

BRICS, Department of Computer Science
Aalborg University, 9220 Aalborg Ø, Denmark
luca@cs.auc.dk, http://www.cs.auc.dk/~luca/BEATCS

This installment of the concurrency column is devoted to a piece by Jorge Perez, which offers a survey of recent results on the expressiveness and decidability of core higher-order process calculi. In such calculi, processes can be passed around in communications, in very much the same way as functions can be applied to other functions in the λ-calculus.

A celebrated result of Sangiorgi's shows that, in the π-calculus, the higher-order paradigm can be encoded in the first-order one—that is, process passing can be expressed in terms of channel passing. However, as the results in this column clearly indicate, there are still many open questions in the study of the expressiveness of higher-order communication in concurrency theory. I hope that this contribution by Jorge Perez will spur further developments in this research area.

As the readers of this column will certainly know, on March 20, 2010, the concurrency-theory community lost one of its true giants, Robin Milner. An obituary for Robin is available on the EATCS web site. This installment of the Concurrency Column is an example of Robin's legacy to our research community and is dedicated to his memory.

Higher-Order Concurrency: Expressiveness and Decidability Results – A Survey

Jorge A. Pérez

CITI - Department of Computer Science, New University of Lisbon, Portugal

Abstract

Higher-order process calculi are formalisms for concurrency in which processes can be passed around in communications. Higher-order (also known as process-passing) concurrency is often presented as an alternative paradigm to the first order (or name-passing) concurrency of the π-calculus for the description of mobile systems. This paper is a survey of recent results on the expressiveness and decidability of *core* higher-order process calculi.

1 Introduction

Context and Motivation. The challenging nature of *concurrent systems* is no longer a novelty for computer science. In fact, by now there is a consolidated understanding on how concurrent behavior departs from sequential computation. As a result of the sustained progress on the formulation of foundational theories of concurrent processes, notions such as *interaction* and *communication* are now widely accepted to be intimately related to *computing* at large. Given the wealth of abstract languages, theories, and application areas that have emerged from this progress, it is fair to say that *concurrency theory* is no longer in its infancy.

This development of concurrency theory coincides with the transition towards *global ubiquitous computing* we witness nowadays. Supported by a number of technological advances —most notably, the availability of cheaper and more powerful processors, the increase in flexibility and power of communication networks, and the widespread consolidation of the Internet— global ubiquitous computing (GUC in the sequel) is a broad term that refers to computing over massively networked, dynamically reconfigurable infrastructures that interconnect heterogeneous collections of computing devices. As such, systems in GUC represent the natural evolution of traditional distributed systems, and distinguish from these in aspects such as *mobility*, *network-awareness*, and *openness*.

Nowadays we find *mobility* in *devices* that move in our physical world while performing diverse kinds of computation (mobile phones, laptops), as well as in

objects travelling across communication networks (SMSs, XML files, runnable code, software agents). Advances in bandwidth growth and network connectivity have broaden the range of feasible communications; as a result, communication objects not only exhibit now an increasingly complex structure but also an autonomous nature. Examples of the evolution in the nature of communication objects include the online distribution of digital content (music, video, e-books) and forms of service mobility in service-oriented architectures.

In general, mobility cannot abstract from the *locations* of the moving entities (computing devices, communication objects). A location can be as concrete as the wireless network a smartphone connects to, or as abstract as the domains in which wide area networks are usually partitioned. Interestingly, there is a reciprocal relationship between locations and mobility, as (the behavior of) a mobile entity and its surrounding environment (determined by its location) might have direct influence on each other. This can be seen, for instance, in in the websites that change depending on the country in which they are accessed, and in the actions of network reconfiguration triggered by high peaks of user activity. This phenomenon, sometimes referred to as *network-awareness*, embodies a notion of *structure* that not only underlies mobile behavior but that often determines it.

Systems in GUC are said to be *open* for they are built as large collections of loosely coupled, heterogeneous components. Such components might not be known a priori; in general, an open system should allow to add, suspend, update, relocate, and remove entire components transparently. Open systems are seldom meant to terminate; as such, their overall behavior must abstract from changes on the local state of its components, and in particular from their malfunction. Hence, forms of *dynamic system reconfiguration* are most natural within models of open systems. Openness is closely related to mobility and network-awareness for it could occur that the structure of the system is reconfigured as a result of the interactions of mobile components. An example of this is a running component which disconnects from one location and later on reconnects to some other location.

The emergence of GUC therefore represents a challenge for computer science in general, and for concurrency theory in particular. As we have seen, such environments feature complex forms of concurrent behavior that go way beyond the (already complex) interaction patterns present in traditional distributed systems. The challenge therefore consists in the formulation of foundational theories to cope with the features of modern computing environments.

Higher-Order Concurrency. We believe that in this context *higher-order process calculi* have much to offer. These are calculi in which processes can be passed around in communications. Higher-order (or *process-passing*) concurrency is often presented as an alternative paradigm to the first-order (or *name-*

passing) concurrency of the π-calculus [20] for the description of mobile systems. Higher-order process calculi emerged first as concurrent extensions of functional languages (see, e.g., [2, 24]). In fact, these calculi are inspired by, and formally close to, the λ-calculus, whose basic computational step —β-reduction— involves term instantiation. Later on, as a way of studying forms of code mobility and mobile agents, a number of process calculi extended with process-passing features were put forward. The basic operators of these calculi are usually those of CCS [17]: parallel composition, input and output prefix, and restriction. Replication and recursion are often omitted as they can be encoded. Early proposals of higher-order process calculi are CHOCS [32], Plain CHOCS [34], and the Higher-Order π-calculus [27]; more recent ones include Homer [11] and the Kell calculi [31].

Important studies on the theory of higher-order process calculi have been carried out by Sangiorgi [27, 29]. In fact, in what is probably the most prominent result for higher-order process calculi, Sangiorgi showed that in the context of the π-calculus, the higher-order paradigm is *representable* into the first-order one. This is achieved by means of a translation in which the communication of a process is modeled as the communication of a pointer that can activate as many copies of such a process as needed. Crucially, such a translation is *fully-abstract* with respect to barbed congruence, the form of contextual equivalence used in concurrency theory. Hence, the behavioral theory from the first-order setting can be readily transferred to the higher-order one. By demonstrating that the higher-order paradigm only adds modeling convenience, this result greatly contributed to consolidate the π-calculus as a basic formalism for concurrency. It also appears to have contributed to a decline of interest in formalisms for higher-order concurrency. In our view, Sangiorgi's representability result was so conclusive at that time that it indirectly put forward the idea that his translation could be adapted to represent *every* kind of higher-order interaction. This misconception seems to persist nowadays, even if it has been shown that for higher-order process calculi with little more than process communication, translations into some first-order language are unsatisfactory or do not exist.

We believe that process-passing communication is closely related to aspects of mobility, network-awareness, and openness as discussed for GUC. The communication of objects with complex structure can be neatly represented in higher-order process calculi by the communication of *terms* of the language. As in the first-order case, extensions of higher-order process calculi with constructs for network-awareness are natural; process communication adds the possibility of describing richer and more realistic interaction patterns between different computation loci. Furthermore, higher-order communication allows to consider autonomous, self-contained software artifacts —such as components, services, or agents— as *first-class objects* which can be moved, executed, manipulated. This allows for clean and modular descriptions of open systems and their behavior. As a result,

higher-order communication arises in abstract languages for GUC in the form of *specialized constructs* that go beyond mere process communication. Instances of such constructs include forms of *localities* that lead to involved process hierarchies featuring complex communication patterns; operators for *process reflection* that allow to observe and/or modify process execution at runtime; sophisticated forms of *pattern matching* or *cryptographic operations* used over terms representing messages or semi-structured data.

The kind of higher-order interactions underlying the behavior of the specialized constructs casts doubts on the convenience of stuyding their properties by means of first-order representations. Based on this insight, we shall argue that foundational studies for higher-order process calculi must be undertaken *directly* on them and exploit their peculiarities. This is particularly critical for those issues that have remained unexplored in the theory of higher-order concurrency. Here we shall concentrate on two of such issues, namely *expressiveness* and *decidability*, two closely interwoven concerns in process calculi at large.

This Survey. We overview recent expressiveness and decidability results for HOcore, a *core calculus* for higher-order concurrency. In fact, HOcore features only the strictly necessary operators to obtain higher-order concurrency. In particular, HOcore lacks name-passing as in the π-calculus. This makes HOcore a convenient framework to carry out a direct study of expressiveness and decidability concerns in the higher-order setting. We first introduce HOcore and review its basic theory (Section 2). Then, in Section 3, we consider a fragment of the calculus which features *limited forwarding* in communications. We discuss the main properties of such a fragment, and describe an extension of it with *passivation*, a specialized construct that is useful to represent dynamic reconfiguration as in GUC scenarios. Finally, in Section 5, we move on to consider polyadic and synchronous communication in the higher-order setting. We study both issues in suitable extensions of HOcore. Section 5 concludes with some final remarks.

Origin of the Results. This survey collects selected results from my PhD Thesis [26]. Such results are based on joint works with Cinzia Di Giusto, Ivan Lanese, Davide Sangiorgi, Alan Schmitt, and Gianluigi Zavattaro, which have been previously published as [14, 7, 15].

2 A Core Calculus for Higher-Order Concurrency

We begin by presenting HOcore, its syntax and semantics. Then, we discuss its expressive power and give intuitions on its behavioral theory.

2.1 The Calculus

HOCORE is the core of calculi for higher-order concurrency such as CHOCS [32], Plain CHOCS [34], and the Higher-Order π-calculus (HOπ) [27, 28, 29]. We use a, b, c to range over *names* (also called *channels*), and x, y, z to range over variables; the sets of names and variables are disjoint.

$$
\begin{array}{rcll}
P, Q & ::= & \overline{a}\langle P\rangle & \text{output} \\
& \mid & a(x).\, P & \text{input prefix} \\
& \mid & x & \text{process variable} \\
& \mid & P \parallel Q & \text{parallel composition} \\
& \mid & \mathbf{0} & \text{nil}
\end{array}
$$

An input prefixed process $a(x).\, P$ can receive on name a a process that will be substituted in the place of x in the body P; an output message $\overline{a}\langle P\rangle$ can send P on a; parallel composition allows processes to interact. HOCORE can be seen as a kind of concurrent λ-calculus, where $a(x).\, P$ is a function, with formal parameter x and body P, located at a; and $\overline{a}\langle P\rangle$ is the argument for a function located at a. Notice that continuations following output messages have been left out, so HOCORE is an asynchronous calculus. Moreover, HOCORE has no restriction operator. Thus all channels are global, and dynamic creation of new channels is impossible.

An input $a(x).\, P$ binds the free occurrences of x in P; this is the only binder in HOCORE. We write $\mathsf{fv}(P)$ for the set of free variables in P, and $\mathsf{bv}(P)$ for the bound variables. We identify processes up to a renaming of bound variables. A process is *closed* if it does not have free variables. In a statement, a name is *fresh* if it is not among the names of the objects (processes, actions, etc.) of the statement. As usual, the scope of an input $a(x).\, P$ extends as far to the right as possible. For instance, $a(x).\, P \parallel Q$ stands for $a(x).\, (P \parallel Q)$. We abbreviate the input $a(x).\, P$, with $x \notin \mathsf{fv}(P)$, as $a.\, P$; the output $\overline{a}\langle \mathbf{0}\rangle$ as $\overline{a}$; and the composition $P_1 \parallel \ldots \parallel P_k$ as $\prod_{i=1}^{k} P_i$. Similarly, we write $\prod_1^n P$ as an abbreviation for the parallel composition of n copies of P. Further, $P\{\widetilde{Q}/\widetilde{x}\}$ denotes the simultaneous substitution of variables $\widetilde{x}$ with processes $\widetilde{Q}$ in P (we assume members of $\widetilde{x}$ are distinct). The *size* of a process P, written $\#(P)$, is inductively defined as:

$$
\begin{array}{c}
\#(\mathbf{0}) = 0 \qquad \#(P \parallel Q) = \#(P) + \#(Q) \qquad \#(x) = 1 \\
\#(\overline{a}\langle P\rangle) = 1 + \#(P) \qquad \#(a(x).\, P) = 1 + \#(P).
\end{array}
$$

A Labeled Transition System (LTS) for HOCORE, defined on open processes, is given in Figure 1. There are three forms of transitions: internal transitions $P \xrightarrow{\tau} P'$; input transitions $P \xrightarrow{a(x)} P'$, meaning that P can receive at a a process that will replace x in the continuation P'; and output transitions $P \xrightarrow{\overline{a}\langle P'\rangle} P''$ meaning that

$$\textsc{Inp}\quad a(x).\,P \xrightarrow{a(x)} P \qquad\qquad \textsc{Out}\quad \overline{a}\langle P\rangle \xrightarrow{\overline{a}\langle P\rangle} \mathbf{0}$$

$$\textsc{Act1}\ \frac{P_1 \xrightarrow{\alpha} P_1' \qquad \mathsf{bv}(\alpha)\cap\mathsf{fv}(P_2)=\emptyset}{P_1 \parallel P_2 \xrightarrow{\alpha} P_1' \parallel P_2} \qquad \textsc{Tau1}\ \frac{P_1 \xrightarrow{\overline{a}\langle P\rangle} P_1' \qquad P_2 \xrightarrow{a(x)} P_2'}{P_1 \parallel P_2 \xrightarrow{\tau} P_1' \parallel P_2'\{P/x\}}$$

Figure 1: An LTS for HOcore. Symmetric rules Act2 and Tau2 are omitted.

P emits P' at a, and in doing so evolves to P''. We use α to denote a generic label of a transition. Notions of free and bound variables extend to labels as expected, in particular x is bound in $a(x)$.

Definition 2.1. *The* structural congruence *relation is the smallest congruence generated by the following laws:*
$P \parallel \mathbf{0} \equiv P,\ \ P_1 \parallel P_2 \equiv P_2 \parallel P_1,\ \ P_1 \parallel (P_2 \parallel P_3) \equiv (P_1 \parallel P_2) \parallel P_3.$

Reductions $P \longrightarrow P'$ are defined as $P \equiv\xrightarrow{\tau}\equiv P'$.

2.2 The Expressive Power of HOcore

We present a deterministic encoding of Minsky machines [21], a well-known Turing complete model, into HOcore. This way, HOcore is shown to be Turing complete. Moreover, since the encoding preserves termination, it also shows that termination in HOcore is undecidable. The encoding exploits basic forms of input-guarded replication and guarded choice that are representable in HOcore; for space reasons, we do not discuss such representations, see [26] for details.

A Minsky machine is composed of two registers and of a set of labelled instructions $(1 : I_1), \ldots, (n : I_n)$ that act over them. Instructions can be of two kinds: (i) *increment* the value of a register j and move on to the following instruction, and (ii) *decrement* the value of register j if it holds a value greater than zero, or jump to instruction k otherwise. We first show how to count and test for zero in HOcore; then, we present the encoding of a Minsky machine into HOcore.

Counting in HOcore. The cornerstone of our encoding is the definition of counters that may be tested for zero. Numbers are represented as nested higher-order processes: the encoding of a number $k+1$ in register j, denoted $(\!|\ k+1\ |\!)_j$, is the parallel composition of two processes: $\overline{r_j^{\mathsf{S}}}\langle (\!|\ k\ |\!)_j\rangle$ (the successor of $(\!|\ k\ |\!)_j$) and a flag $\widehat{n_j}$. The encoding of zero comprises such a flag, as well as the message $\overline{r_j^{0}}$. As an example, $(\!|\ 2\ |\!)_j$ is $\overline{r_j^{\mathsf{S}}}\langle \overline{r_j^{\mathsf{S}}}\langle \overline{r_j^{0}} \parallel \widehat{n_j}\rangle \parallel \widehat{n_j}\rangle \parallel \widehat{n_j}$.

$$
\begin{array}{lcl}
\textsc{Instructions}\ (i : I_i) & & \\
[\![(i : \mathtt{INC}(r_j))]\!]_{\mathsf{M}} & = & !p_i.\,(\widehat{inc_j} \parallel ack.\,\overline{p_{i+1}}) \\
[\![(i : \mathtt{DECJ}(r_j, k))]\!]_{\mathsf{M}} & = & !p_i.\,(\widehat{dec_j} \parallel ack.\,(z_j.\,\overline{p_k} + n_j.\,\overline{p_{i+1}})
\end{array}
$$

$$
\begin{array}{lcl}
\textsc{Registers}\ r_j & & \\
[\![r_j = 0]\!]_{\mathsf{M}} & = & (inc_j.\,\overline{r_j^{\mathsf{S}}}\langle (\!|\, 0 \,|\!)_j\rangle \ + \ dec_j.\,(\overline{r_j^0} \parallel \widehat{z_j})) \parallel \mathsf{REG}_j \\
[\![r_j = m]\!]_{\mathsf{M}} & = & (inc_j.\,\overline{r_j^{\mathsf{S}}}\langle (\!|\, m \,|\!)_j\rangle \ + \ dec_j.\,(\!|\, m-1 \,|\!)_j) \parallel \mathsf{REG}_j
\end{array}
$$

where:

$$
\begin{array}{lcl}
\mathsf{REG}_j & = & !r_j^0.\,(\overline{ack} \parallel inc_j.\,\overline{r_j^{\mathsf{S}}}\langle (\!|\, 0 \,|\!)_j\rangle \ + \ dec_j.\,(\overline{r_j^0} \parallel \widehat{z_j})) \parallel \\
 & & !r_j^{\mathsf{S}}(Y).\,(\overline{ack} \parallel inc_j.\,\overline{r_j^{\mathsf{S}}}\langle \overline{r_j^{\mathsf{S}}}\langle Y\rangle \parallel \widehat{n_j}\rangle \ + \ dec_j.\,Y)
\end{array}
$$

$$
(\!|\, k \,|\!)_j = \begin{cases} \overline{r_j^0} \parallel \widehat{n_j} & \text{if } k = 0 \\ \overline{r_j^{\mathsf{S}}}\langle (\!|\, k-1 \,|\!)_j\rangle \parallel \widehat{n_j} & \text{if } k > 0. \end{cases}
$$

Figure 2: Encoding of Minsky machines into HOcore

Registers. Registers are counters that may be incremented and decremented. They consist of two parts: their current state and two mutually recursive processes used to generate a new state after an increment or decrement of the register. The state depends on whether the current value of the register is zero or not, but in both cases it consists of a choice between an increment and a decrement. In case of an increment, a message on r_j^{S} is sent containing the current register value, for instance m. This message is then received by the recursive definition of r_j^{S} that creates a new state with value $m + 1$, ready for further increment or decrement. In case of a decrement, the behavior depends on the current value. If the current value is zero, then it stays at zero, recreating the state corresponding to zero for further operations using the message on r_j^0, and it spawns a flag $\widehat{z_j}$ indicating that a decrement on a zero-valued register has occurred. If the current value m is strictly greater than zero, then the process $(\!|\, m-1 \,|\!)_j$ is spawned. If m was equal to 1, this puts the state of the register to zero (using a message on r_j^0). Otherwise, it keeps the message in a non-zero state, with value $m - 1$, using a message on r_j^{S}. In both cases a flag $\widehat{n_j}$ is spawned to indicate that the register was not equal to zero before the decrement. When an increment or decrement has been processed, that is when the new current state has been created, an acknowledgment is sent to proceed with the execution of the next instruction.

Instructions. Each instruction $(i : I_i)$ is a replicated process guarded by p_i, which represents the program counter when $p = i$. Once p_i is consumed, the instruction is active and an interaction with a register occurs. In case of an increment instruction, the corresponding choice is sent to the relevant register and,

upon reception of the acknowledgment, the next instruction is spawned. In case of a decrement, the corresponding choice is sent to the register, then an acknowledgment is received followed by a choice depending on whether the register was zero, resulting in a jump to the specified instruction, or the spawning of the next instruction otherwise.

The Encoding. The encoding of Minsky machines into HOcore, denoted as $[\![\cdot]\!]_{\mathsf{M}}$, is presented in Table 2. The encoding of a configuration of a Minsky machine (i.e., the label of the current instruction and the current value of the two registers) requires a finite number of fresh names (linear on n, the number of instructions). Before stating the correctness of the encoding, some notation is necessary. In HOcore, we write $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$, and $P \Uparrow$ if P has an infinite sequence of reductions. Similarly, in Minsky machines $\longrightarrow^*_{\mathsf{M}}$ is the reflexive and transitive closure of $\longrightarrow_{\mathsf{M}}$, and $N \Uparrow_{\mathsf{M}}$ means that N has an infinite sequence of reductions. We then have the following lemma, whose proof is detailed in [26]:

Lemma 2.1. *Let N be a Minsky machine. We have:*

1. $N \longrightarrow^*_{\mathsf{M}} N'$ *iff* $[\![N]\!]_{\mathsf{M}} \longrightarrow^* [\![N']\!]_{\mathsf{M}}$;
2. *if* $[\![N]\!]_{\mathsf{M}} \longrightarrow^* P_1$ *and* $[\![N]\!]_{\mathsf{M}} \longrightarrow^* P_2$*, then there exists* N' *such that* $P_1 \longrightarrow^* [\![N']\!]_{\mathsf{M}}$ *and* $P_2 \longrightarrow^* [\![N']\!]_{\mathsf{M}}$;
3. $N \Uparrow_{\mathsf{M}}$ *iff* $[\![N]\!]_{\mathsf{M}} \Uparrow$.

Lemma 2.1 guarantees that HOcore is Turing complete, and since the encoding preserves termination, it entails the following corollary.

Corollary 2.1. *Termination in* HOcore *is undecidable.*

2.3 Behavioral Theory of HOcore

While notions of behavioral equivalences are well understood for first-order calculi, in the higher-order setting the situation is less clear. Notions of behavioral equivalences for higher-order process calculi include *higher-order bisimilarity* [33], *context* and *normal* bisimilarities [28] (the latter being an economic characterization of the former), and *enviromental* bisimilarity [30]. (See [26, Chapter 2] for extended discussions on bisimilarities in the higher-order setting.)

In HOcore, the main forms of strong bisimilarity for higher-order process calculi coincide. Moreover, such a relation is *decidable*. A key ingredient for these results is the notion of *Input/Output (IO) bisimulation*, a bisimulation in which the variable of input prefixes is never instantiated and τ-transitions are not

observed. To the best of our knowledge, HOcore is the first calculus where a bisimulation without a clause for τ-transitions is discriminating enough so as to provide a useful characterization of process behavior (actually, bisimulation is very discriminating in HOcore). In fact, it can be verified that this is not the case in first-order calculi such as CCS and the π-calculus, and in other higher-order process calculi such as HOπ.

In what follows, we present the basic definitions and results for IO bisimulation, as they are essential in the behavioral theory of HOcore. We refer to [14, 26] for full details on the collapsing and decidability results.

Definition 2.2 (IO bisimulation)**.** *A symmetric relation $\mathcal{R}$ on* HOcore *processes is an* Input/Output (IO) bisimulation *if $P \mathcal{R} Q$ implies*

1. *whenever $P \xrightarrow{\overline{a}\langle P''\rangle} P'$ there are Q', Q'' such that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$ with $P' \mathcal{R} Q'$ and $P'' \mathcal{R} Q''$;*
2. *whenever $P \xrightarrow{a(x)} P'$ there is Q' such that $Q \xrightarrow{a(x)} Q'$ and $P' \mathcal{R} Q'$,*
3. *whenever $P \equiv x \parallel P'$ there is Q' such that $Q \equiv x \parallel Q'$ and $P' \mathcal{R} Q'$.*

Input/Output bisimilarity, written $\sim^{\circ}_{\mathtt{IO}}$, is the largest relation on open HOcore *processes that is an IO bisimulation.*

Remarkably, $\sim^{\circ}_{\mathtt{IO}}$ has a straightforward proof of congruence. This is significant because congruence is usually a hard problem in bisimilarities for higher-order calculi.

Lemma 2.2. *Let P_1, P_2 be open* HOcore *processes. $P_1 \sim^{\circ}_{\mathtt{IO}} P_2$ implies:*

1. *$a(x).\, P_1 \sim^{\circ}_{\mathtt{IO}} a(x).\, P_2$;*
2. *$P_1 \parallel R \sim^{\circ}_{\mathtt{IO}} P_2 \parallel R$, for every R;*
3. *$\overline{a}\langle P_1\rangle \sim^{\circ}_{\mathtt{IO}} \overline{a}\langle P_2\rangle$.*

Proof (Sketch). By showing that $\sim^{\circ}_{\mathtt{IO}}$ is preserved by each operator of the calculus. All cases are easy. For parallel composition, it is essential that $\sim^{\circ}_{\mathtt{IO}}$ does not require to match τ actions in the bisimulation game. □

Lemma 2.3. *If $P \sim^{\circ}_{\mathtt{IO}} Q$ then for all x and R, also $P\{R/x\} \sim^{\circ}_{\mathtt{IO}} Q\{R/x\}$.*

Proof (Sketch). We take the relation on HOcore processes with all pairs of the form $(P'\{R/x\} \parallel L,\ Q'\{R/x\} \parallel L)$ where P', Q' are guarded (i.e., free variables occur only in sub-expressions of the form $\pi.\, S$, where π is a prefix) and $P' \sim^{\circ}_{\mathtt{IO}} Q'$,

and show that this is an open IO bisimulation up to $\equiv$ (this simple form of "up-to technique" is common for bisimilarities). The proof makes use of lemmas showing the effect of process substitutions on the behaviors of open processes, and of a few simple algebraic manipulations. □

In contrast with the other bisimilarities, in $\sim^{\circ}_{\mathtt{IO}}$ the size of processes always decreases during the bisimulation game. This is because $\sim^{\circ}_{\mathtt{IO}}$ is an open relation and does not have a clause for τ transitions, hence process copying never occurs.

Lemma 2.4. *Relation $\sim^{\circ}_{\mathtt{IO}}$ is decidable.*

A symmetric relation $\mathcal{R}$ on HOcore processes is called a *τ-bisimulation* if $P \mathrel{\mathcal{R}} Q$ and $P \xrightarrow{\tau} P'$ imply that there is Q' such that $Q \xrightarrow{\tau} Q'$ and $P' \mathrel{\mathcal{R}} Q'$. The following lemma shows that $\sim^{\circ}_{\mathtt{IO}}$ is also a τ-bisimulation. In [14, 26] we have used this result to prove that $\sim^{\circ}_{\mathtt{IO}}$ coincides with other bisimilarities in the higher-order setting, and to transfer to them its properties, in particular congruence and decidability.

Lemma 2.5. *Relation $\sim^{\circ}_{\mathtt{IO}}$ is a τ-bisimulation.*

Proof (Sketch). Suppose $P \sim^{\circ}_{\mathtt{IO}} Q$ and $P \xrightarrow{\tau} P'$. We have to find a matching transition from Q. We can decompose P's transition into an output $P \xrightarrow{\overline{a}\langle R\rangle} P_1$ followed by an input $P_1 \xrightarrow{a(x)} P_2$, with $P' = P_2\{R/x\}$. By definition of $\sim^{\circ}_{\mathtt{IO}}$, Q is capable of matching these transitions, and the final derivative is a process Q_2 with $Q_2 \sim^{\circ}_{\mathtt{IO}} P_2$. Further, as HOcore has no output prefixes the two transitions from Q can be combined into a τ-transition, which matches the initial τ-transition from P. We conclude using Lemmas 2.2 and 2.3. □

Other Results. In addition to being decidable, in [14, 26] strong bisimilarity in HOcore has been shown to coincide with *barbed congruence*, the form of contextual equivalence used in concurrency. In fact, we have considered *asynchronous* barbed congruence, where barbs are only produced by output messages. Moreover, it has been shown that bisimilarity in HOcore enjoys of a sound and complete axiomatization; such results rely on an adaptation to the higher-order setting of results on unique decomposition of processes [19, 22] and on an axiomatization of bisimilarity for a finite fragment of CCS [12]. The axiomatization has been exploited for obtaining complexity bounds for bisimilarity checking. In fact, using the axiomatization it has been determined that bisimilarity checking is polynomial on the size of the processes. Furthermore, decidability of bisimilarity is shown to break when HOcore is extended with *four top-level restrictions*, i.e., restrictions that can only occur at the outermost part of a process. This *undecidability* result is obtained using a reduction from the Post correspondence problem (PCP). See [14, 26] for details on these results.

3 The Expressive Power of Forwarding

As we have seen, in spite of its minimality, HOcore is a very expressive formalism. It is then natural to aim at identifying the intrinsic source of expressive power in HOcore. A crucial observation in higher-order communication is that communicated processes have only two capabilities: execution and forwarding. Based on this insight, we have studied a fragment of HOcore in which higher-order communication obeys a limited form of *forwarding*. The fragment, denoted $\text{Ho}^{-\text{f}}$, arises from a syntactic restriction on the shape of output objects.

We study the expressiveness of $\text{Ho}^{-\text{f}}$ using decidability of termination and convergence of processes as a yardstick. We define termination as the non existence of divergent computations, and convergence as existence of a terminating computation.[1] Both these properties are undecidable in HOcore. Our main result for $\text{Ho}^{-\text{f}}$ shows that while convergence remains *undecidable*, termination becomes *decidable*. This suggests a loss in expressive power when moving from HOcore to $\text{Ho}^{-\text{f}}$. Then, as a way of recovering such an expressive power, we extend the calculus with a form of process suspension called *passivation*. In the resulting calculus (called $\text{HoP}^{-\text{f}}$) both termination and convergence are undecidable.

Limited Forwarding. We find that a substantial part of the expressive power of a higher-order process calculus resides in the ability of *forwarding* a received process within an *arbitrary context*. For instance, consider the process $R = a(x).\, \overline{b}\langle P_x \rangle$, where P_x stands for a process P with free occurrences of a variable x. Intuitively, R receives a process on name a and forwards it on name b. It is easy to see that the actual structure of P_x can be fairly complex. One could even "wrap" the process to be received in x using an arbitrary number of "output layers", i.e., by letting $P_x = \overline{b_1}\langle \overline{b_2}\langle \ldots \overline{b_k}\langle x \rangle\rangle \ldots \rangle$. This *nesting capability* embodies a great deal of the expressiveness of HOcore: in fact, the encoding of Minsky machines in HOcore depends critically on nested-based counters. Therefore, investigating limitations to the forwarding capabilities in higher-order communications is a legitimate approach to assess the expressive power of higher-order concurrency.

With the above in mind, we have studied $\text{Ho}^{-\text{f}}$, a subcalculus of HOcore in which output actions are limited so as to rule out the nesting capability. In $\text{Ho}^{-\text{f}}$, output actions can communicate the parallel composition of two kinds of objects:

1. closed processes (i.e., processes that do not contain free variables), and

2. processes received through previously executed input actions.

[1] Termination and convergence are sometimes also referred to as *universal* and *existential* termination, respectively.

Hence, the context in which the output action resides can only contribute to communication by "appending" pieces of code that admit no inspection, available in the form of a black-box. More precisely, the grammar of $\mathrm{Ho}^{-\mathrm{f}}$ processes is the same as that of HOcore, except for the production for output actions, which is replaced by the following one:

$$\overline{a}\langle x_1 \parallel \cdots \parallel x_k \parallel P\rangle$$

where $k \geq 0$ and P is a closed process. This modification directly restricts forwarding capabilities for output processes, which in turn, leads to a more limited structure of processes along reductions.

The limited style of higher-order communication enforced in $\mathrm{Ho}^{-\mathrm{f}}$ is inspired by those cases in which a process P is communicated in a translated format $[\![P]\!]$, and the translation is not compositional. That is, the cases in which, for any process context C, the translation of $C[P]$ cannot be seen as a function of the translation of P, i.e., there exists no context D such that $[\![C[P]]\!] = D[P]$. This can be related to several existing programming scenarios. The simplest example is perhaps mobility of already compiled code, on which it is not possible to apply inverse translations (such as reverse engineering).

3.1 Some Preliminaries

The syntax of $\mathrm{Ho}^{-\mathrm{f}}$ is exactly the same of HOcore, excepting the production for output actions, given above. All the notations and definitions given for HOcore (including structural congruence) apply to $\mathrm{Ho}^{-\mathrm{f}}$ as expected. As for the LTS, for technical reasons in the proof of decidability of termination we shall appeal to a *finitely branching* LTS over $\mathrm{Ho}^{-\mathrm{f}}$ processes; see [26, Chapter 5] for details.

As usual, the internal runs of a process are given by sequences of *reductions*. Given a process P, its reductions $P \longrightarrow P'$ are defined as $P \xrightarrow{\tau} P'$. We denote with $\longrightarrow^*$ the reflexive and transitive closure of $\longrightarrow$; notation $\longrightarrow^j$ is to stand for a sequence of j reductions. We use $P \not\longrightarrow$ to denote that there is no P' such that $P \longrightarrow P'$. Following [5] we define process convergence and process termination. Observe that termination implies convergence while the opposite does not hold.

Definition 3.1. *Let P be a* $\mathrm{Ho}^{-\mathrm{f}}$ *process. We say that P* converges *iff there exists P' such that $P \longrightarrow^* P'$ and $P' \not\longrightarrow$. We say that P* terminates *iff there exist no $\{P_i\}_{i\in\mathbb{N}}$ such that $P_0 = P$ and $P_j \longrightarrow P_{j+1}$ for any j.*

3.2 Undecidability of Convergence

We present an encoding of Minsky machines into $\mathrm{Ho}^{-\mathrm{f}}$. However, unlike the encoding of Minsky machines in HOcore, the encoding into $\mathrm{Ho}^{-\mathrm{f}}$ is *not faithful*: it

$$\textsc{Register}\ r_j \qquad [\![r_j = m]\!]_{\mathsf{M}} = \prod_1^m \overline{u_j}$$

$$\begin{array}{lcl}
\multicolumn{3}{l}{\textsc{Instructions}\ (i : I_i)} \\
[\![(i : \mathtt{INC}(r_j))]\!]_{\mathsf{M}} & = & !p_i.\,(\overline{u_j} \parallel set_j(x).\,(\overline{set_j}\langle x \parallel \textsc{Inc}\rangle \parallel \overline{p_{i+1}})) \\
[\![(i : \mathtt{DECJ}(r_j, s))]\!]_{\mathsf{M}} & = & \ \ !p_i.\,(\overline{loop} \parallel u_j.\,loop.\,set_j(x).\,(\overline{set_j}\langle x \parallel \textsc{Dec}\rangle \parallel \overline{p_{i+1}})) \\
 & & \parallel\ !p_i.\,set_j(x).\,(x \parallel \overline{set_j}\langle x\rangle \parallel \overline{p_s})
\end{array}$$

where

$$\textsc{Inc} \ = \ \overline{loop} \qquad\qquad \textsc{Dec} = loop$$

Figure 3: Encoding of Minsky machines into $\mathrm{Ho}^{-\mathrm{f}}$

may introduce computations which do not correspond to the expected behavior of the modeled machine. Such computations are forced to be infinite and thus regarded as non-halting computations which are therefore ignored. More precisely, given a Minsky machine N, its encoding $[\![N]\!]$ has a terminating computation if and only if N terminates. This allows to prove that convergence is undecidable.

Encoding Minsky Machines into $\mathrm{Ho}^{-\mathrm{f}}$. The encoding of Minsky machines into $\mathrm{Ho}^{-\mathrm{f}}$ is denoted by $[\![\cdot]\!]_{\mathsf{M}}$ and presented in Figure 3. As in the encoding into HOcore, we use a form of input-guarded replication which can be encoded in $\mathrm{Ho}^{-\mathrm{f}}$. The encoding is assumed to execute in parallel with a process $loop.\,\textsc{Div}$, which represents divergent behavior that is spawned in certain cases with an output on name $loop$. We first discuss the encodings of registers and instructions.

A register r_j that stores the number m is encoded as the parallel composition of m copies of the unit process $\overline{u_j}$. To implement the test for zero, we need to record how many increments and decrements have been performed on the register r_j. This is done by using a process $\textsc{Log}_j$, which is communicated back and forth on name set_j. When an increment instruction occurs, a new copy of the process $\overline{u_j}$ is created, and the process $\textsc{Log}_j$ is updated by adding the process $\textsc{Inc}$ in parallel. Similarly for decrements: a copy of $\overline{u_j}$ is consumed and the process $\textsc{Dec}$ is added to $\textsc{Log}_j$. As a result, after k increments and l decrements on register r_j, we have that $\textsc{Log}_j = \prod_k \textsc{Inc} \parallel \prod_l \textsc{Dec}$.

Each instruction $(i : I_i)$ is a replicated process guarded by p_i, which represents the program counter when $p = i$. Once p_i is consumed, the instruction is active and, in the case of increments and decrements, an interaction with a register occurs. We already described the behavior of increments. Let us now focus on decrements, the instructions that can introduce divergent —unfaithful— computations. In this case, the process can internally choose either to actually perform a decrement and proceed with the next instruction, or to jump. This internal choice takes place on p_i; it can be seen as a *guess* the process makes on the actual number stored by the register r_j. Therefore, two situations can occur:

1. *The process chooses to decrement r_j.* In this case a process $\overline{loop}$ as well as an input on u_j become immediately available. The purpose of the latter is to produce a synchronization with a complementary output on $\overline{u_j}$ (that represents a unit of r_j).

   If this operation succeeds (i.e., the guess is right as the content of r_j is greater than 0) then a synchronization between the output $\overline{loop}$ —available at the beginning— and the input on $loop$ that guards the update of Log_j takes place. After this synchronization, the log of the register is updated (this is represented by two synchronizations on name set_j) and instruction p_{i+1} is enabled. Otherwise, if the synchronization on u_j fails then it is because the content of r_j is zero and the process made a wrong guess. The process $\overline{loop}$ available at the beginning then synchronizes with the external process $loop.\,\text{Div}$, thus spawning a divergent computation.

2. *The process chooses to jump to instruction p_s.* In this case, the encoding checks if the actual value stored by r_j is zero. To do so, the process receives the process Log_j on name set_j and launches it. The log contains a number of Inc and Dec processes; depending on the actual number of increments and decrements, two situations can occur.

   In the first situation, the number of increments is equal to the number of decrements (say k); hence, the value of the r_j is indeed zero and the process made a right guess. In this case, k synchronizations on name $loop$ take place and instruction p_s is enabled. In the second situation, the number of increments is greater than the number of decrements; hence, the value of r_j is greater than zero and the process made a wrong guess. As a result, at least one of the $\overline{loop}$ signals remains active; by means of a synchronization the process $loop.\,\text{Div}$ this is enough to to spawn a divergent computation.

The correctness for the encoding is stated in terms of configurations of the Minsky machines. The encoding of a Minsky machine configuration (i, m_0, m_1), according to the encodings in Table 3, is denoted $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$. We omit the formal definition of the encoding; see [26, Chapter 5] for details. Such a definition states that, before executing the instructions, both registers in the Minsky machine to be set to zero. This is to guarantee correctness: starting with values different from zero in the registers (without proper initialization of the logs) can lead to inconsistencies.

Theorem 3.1. *Let N be a Minsky machine with registers $r_0 = m_0$, $r_1 = m_1$, instructions $(1 : I_1), \ldots, (n : I_n)$, and configuration (i, m_0, m_1). Let $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$ be the encoding of (i, m_0, m_1) into* $\text{Ho}^{-\mathsf{f}}$. *Then (i, m_0, m_1) terminates if and only if process $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$ converges.*

As a consequence of the results above we have the following.

Corollary 3.1. *Convergence is undecidable in* $\mathrm{Ho}^{-\mathrm{f}}$.

3.3 Decidability of Termination

In this section we discuss decidability of termination for $\mathrm{Ho}^{-\mathrm{f}}$ processes. This is in sharp contrast with the analogous result for HOcore. This decidability result is obtained by appealing to the theory of *well-structured transition systems* [8, 1, 9], following the approach used in [5]. We begin by summarizing main definitions and results for well-structured transition systems.

Well-Structured Transition Systems. The following results and definitions are from [9], unless differently specified. Recall that a *quasi-order* (or, equivalently, preorder) is a reflexive and transitive relation.

Definition 3.2 (Well-quasi-order)**.** *A* well-quasi-order *(wqo) is a quasi-order $\leq$ over a set X such that, for any infinite sequence $x_0, x_1, x_2 \ldots \in X$, there exist indexes $i < j$ such that $x_i \leq x_j$.*

Note that if $\leq$ is a wqo then any infinite sequence $x_0, x_1, x_2, \ldots$ contains an infinite increasing subsequence $x_{i_0}, x_{i_1}, x_{i_2}, \ldots$ (with $i_0 < i_1 < i_2 < \ldots$). Thus well-quasi-orders exclude the possibility of having infinite strictly decreasing sequences. We also need a definition for (finitely branching) transition systems. Here and in the following $\rightarrow^*$ denotes the reflexive, transitive closure of $\rightarrow$.

Definition 3.3 (Transition system)**.** *A* transition system *is a structure $TS = (S, \rightarrow)$, where S is a set of states and $\rightarrow \subseteq S \times S$ is a set of transitions. We define $Succ(s)$ as the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of S. We say that TS is finitely branching if, for each $s \in S$, $Succ(s)$ is finite.*

The key tool to decide several properties of computations is the notion of *well-structured transition system (or WSTS).* This is a transition system equipped with a well-quasi-order on states which is compatible with the transition relation.

Definition 3.4 (WSTS)**.** *A* well-structured transition system with strong compatibility *is a transition system $TS = (S, \rightarrow)$, equipped with a quasi-order $\leq$ on S, such that the two following conditions hold: (1) $\leq$ is a well-quasi-order; (2) $\leq$ is strongly (upward) compatible with $\rightarrow$, that is, for all $s_1 \leq t_1$ and all transitions $s_1 \rightarrow s_2$, there exists a state t_2 such that $t_1 \rightarrow t_2$ and $s_2 \leq t_2$ holds.*

The following theorem is a special case of Theorem 4.6 in [9] and will be used to obtain our decidability result.

$$\textsc{Inp}\quad a(x).\,P \xrightarrow{a(x)} P \qquad\qquad \textsc{Out}\quad \overline{a}\langle P\rangle \xrightarrow{\overline{a}\langle P\rangle} \mathbf{0}$$

$$\textsc{Act1}\ \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \parallel P_2 \xrightarrow{\alpha} P_1' \parallel P_2} \qquad \textsc{Tau1}\ \frac{P_1 \xrightarrow{\overline{a}\langle P\rangle} P_1' \qquad P_2 \xrightarrow{a(x)} P_2'}{P_1 \parallel P_2 \xrightarrow{\tau} P_1' \parallel P_2'\{P/x\}}$$

Figure 4: A finitely branching LTS for $\mathrm{Ho}^{-\mathrm{f}}$. Rules Act2 and Tau2, the symmetric counterparts of Act1 and Tau1, have been omitted.

Theorem 3.2. *Let $TS = (S, \rightarrow, \leq)$ be a finitely branching, well-structured transition system with strong compatibility, decidable $\leq$, and computable $Succ$. Then the existence of an infinite computation starting from a state $s \in S$ is decidable.*

A WSTS for $\mathrm{Ho}^{-\mathrm{f}}$. Theorem 3.2 ensures decidability of termination for generic WSTSs. Hence, in order to prove decidability of termination for $\mathrm{Ho}^{-\mathrm{f}}$ we require to instantiate the theorem appropriately, i.e., we need:

- to define an well-quasi-order $\leq$ and a finitely branching LTS $\xrightarrow{\alpha}$ over $\mathrm{Ho}^{-\mathrm{f}}$ processes;
- to show that $\leq$ is strongly compatible with respect to $\longmapsto$.

The detailed proof for decidability of termination is quite technical, and is given in [26, Chapter 5]. In what follows we provide an extended sketch of it, in which we discuss how to carry out the above two tasks.

The first step in the proof consists in defining a finitely branching LTS $\longmapsto$ over $\mathrm{Ho}^{-\mathrm{f}}$ processes. This is not a demanding requirement in our case; the sensible issue here is the treatment of alpha-conversion. We thus introduce an alternative LTS *without* alpha-conversion. The alternative LTS, given in Figure 4, is restricted to closed processes. In proofs, we shall concentrate on reductions (denoted $\longmapsto$), which are defined as the internal actions of the alternative LTS (i.e., $\xrightarrow{\tau}$).

Intuitively, the crux of the proof consists in finding an upper bound for a process and its derivatives. This is possible in $\mathrm{Ho}^{-\mathrm{f}}$ because of the limited structure allowed in output actions. We now proceed to characterize such an upper bound and to define an ordering over $\mathrm{Ho}^{-\mathrm{f}}$ processes. Central to both issues is a notion of *normal form* for $\mathrm{Ho}^{-\mathrm{f}}$ processes:

Definition 3.5 (Normal Form)**.** *Let $P \in \mathrm{Ho}^{-\mathrm{f}}$. P is in* normal form *iff*

$$P = \prod_{k=1}^{l} x_k \parallel \prod_{i=1}^{m} a_i(y_i).\,P_i \parallel \prod_{j=1}^{n} \overline{b_j}\langle P_j'\rangle$$

where each P_i and P_j' are in normal form.

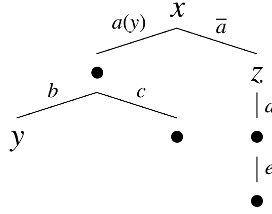
We now define an ordering over normal forms. Intuitively, a process is larger than another if it has more parallel components.

Definition 3.6 (Relation $\preceq$)**.** *Let $P, Q \in \mathrm{Ho}^{-\mathrm{f}}$. We write $P \preceq Q$ iff there exist $x_1 \dots x_l$, $P_1 \dots P_m$, $P'_1 \dots P'_n$, $Q_1 \dots Q_m$, $Q'_1 \dots Q'_n$, and R such that*

$$\begin{array}{rcl} P & \equiv & \prod_{k=1}^{l} x_k \parallel \prod_{i=1}^{m} a_i(y_i).\, P_i \parallel \prod_{j=1}^{n} \overline{b_j}\langle P'_j \rangle \\ Q & \equiv & \prod_{k=1}^{l} x_k \parallel \prod_{i=1}^{m} a_i(y_i).\, Q_i \parallel \prod_{j=1}^{n} \overline{b_j}\langle Q'_j \rangle \parallel R \end{array}$$

with $P_i \preceq Q_i$ and $P'_j \preceq Q'_j$, for $i \in [1..m]$ and $j \in [1..n]$.

The normal form of a process admits an intuitive tree-like representation. Given the process in normal form $P = \prod_{k=1}^{l} x_k \parallel \prod_{i=1}^{m} a_i(y_i).\, P_i \parallel \prod_{j=1}^{n} \overline{b_j}\langle P'_j \rangle$ we shall decree its associated tree to have a root node labeled $x_1, \dots, x_k$. This root node has $m + n$ children, corresponding to the trees associated to processes $P_1, \dots, P_m$ and $P'_1, \dots, P'_m$; the outgoing edges connecting the root node and the children are labeled $a_1(y_1), \dots, a_m(y_m)$ and $\overline{b_1}, \dots, \overline{b_n}$. As an example, the $\mathrm{Ho}^{-\mathrm{f}}$ process $P = x \parallel a(y).\,(b.\,y \parallel c) \parallel \overline{a}\langle z \parallel d.\,e \rangle$ has the following tree representation:



Representing processes in normal form as trees is useful to reason about the structure of $\mathrm{Ho}^{-\mathrm{f}}$ terms. We say that the *depth* of a process corresponds to the maximum depth of its tree representation. Moreover, given a natural number n and a process P, $\mathcal{P}_{P,n}$ denotes the set that contains all those processes in normal form that can be built using the alphabet of P and whose depth is at most n.

When compared to processes in languages such as CCS, higher-order processes have a more complex structure. This is because, by virtue of reductions, an arbitrary process can take the place of possibly several occurrences of a single variable. As a consequence, the depth of (the syntax tree of) a process cannot be determined (or even approximated) before its execution: it can vary arbitrarily along reductions. Crucially, in $\mathrm{Ho}^{-\mathrm{f}}$ *it is possible to bound such a depth*. Our approach is the following: rather than solely depending on the depth of a process, we define measures on the relative position of variables within a process. Informally speaking, such a position will be determined by the number of prefixes guarding a variable. Since variables are allowed only at the top level of the output objects, their relative distance will remain invariant during reductions. This allows to obtain a bound on the structure of $\mathrm{Ho}^{-\mathrm{f}}$ processes. Finally, it is worth stressing that

even if the same notions of normal form, depth, and distance can be defined for HOcore, a finite upper bound for such a language does not exist.

Considering all the above, and defining the set $\mathsf{Deriv}(P)$ as $\{Q \mid P \longmapsto^{*} Q\}$, we have shown the following lemma, which formalizes the upper bound on $\mathrm{Ho}^{-\mathrm{f}}$ processes along reductions:

Lemma 3.1. *Let $P \in \mathrm{Ho}^{-\mathrm{f}}$. Then $\mathsf{Deriv}(P) \subseteq \mathcal{P}_{P, 2\cdot\mathsf{depth}(P)}$.*

We have the following two results, whose proofs are omitted for space reasons (see [26, Chapter 5] for details):

Proposition 3.1. *The relation $\preceq$ is a quasi-order.*

Theorem 3.3 (Well-quasi-order)**.** *Let $P \in \mathrm{Ho}^{-\mathrm{f}}$ be a closed process and $n \geq 0$. The relation $\preceq$ is a well-quasi-order over $\mathcal{P}_{P,n}$.*

The final component in our proof is the strong compatibility of $\preceq$ with respect to $\longmapsto$. We have the following:

Theorem 3.4 (Strong Compatibility)**.** *Let $P, Q, P' \in \mathrm{Ho}^{-\mathrm{f}}$. If $P \preceq Q$ and $P \longmapsto P'$ then there exists Q' such that $Q \longmapsto Q'$ and $P' \preceq Q'$.*

Proof (Sketch). By case analysis on the rule used to infer reduction $P \longmapsto P'$. We content ourselves with illustrating the case derived from the use of rule Tau1; the other ones are similar or simpler. We then have that $P = P' \parallel P''$ with $P' \xmapsto{\overline{a}\langle P_1 \rangle} N$ and $P'' \xmapsto{a(y)} P_2$. Hence, $P \equiv \overline{a}\langle P_1 \rangle \parallel a(y).\, P_2 \parallel N$. Since by hypothesis $P \preceq Q$, we obtain a similar structure for Q. Indeed, $Q \equiv \overline{a}\langle Q_1 \rangle \parallel a(y).\, Q_2 \parallel N'$ with $P_1 \preceq Q_1$, $P_2 \preceq Q_2$, and $N \preceq N'$.

Now, if $P \longmapsto P' \equiv P_2\{^{P_1}/y\} \parallel N$ then also $Q \longmapsto Q' \equiv Q_2\{^{Q_1}/y\} \parallel N'$. It can be shown that if P, P', Q, and Q' are $\mathrm{Ho}^{-\mathrm{f}}$ processes in normal form such that $P \preceq P'$ and $Q \preceq Q'$, then it holds that $P\{^{Q}/x\} \preceq P'\{^{Q'}/x\}$. Using this, we have $P_2\{^{P_1}/y\} \preceq Q_2\{^{Q_1}/y\}$; using this and the hypothesis the thesis follows. □

The following theorem relies on Lemma 3.1, and on Theorems 3.3 and 3.4:

Theorem 3.5. *Let $P \in \mathrm{Ho}^{-\mathrm{f}}$ be a closed process. The transition system*

$$(\mathsf{Deriv}(P), \longmapsto, \preceq)$$

is a finitely branching well-structured transition system with strong compatibility, decidable $\preceq$, and computable $Succ$.

We are thus in place to state the desired result:

Corollary 3.2. *Let $P \in \mathrm{Ho}^{-\mathrm{f}}$ be a closed process. Then, termination of P is decidable.*

3.4 The Interplay of Forwarding and Process Suspension

The decidability result for termination in $\mathrm{Ho}^{-\mathrm{f}}$ reveals a loss of expressive power associated to the restriction on the shape of output objects. It is therefore worth investigating alternatives for recovering such an expressive power while preserving the essence of limited forwarding. One such alternatives is to extend $\mathrm{Ho}^{-\mathrm{f}}$ with a *passivation* construct, an operator that allows to *suspend* the execution of a process at run time. Passivation thus comes in handy to represent mechanisms for dynamic system reconfiguration, which are common in GUC scenarios. Passivation has been considered by higher-order calculi such as the Kell calculus [31] and Homer [11], and finds several applications (see, e.g., [4]).

We consider $\mathrm{HoP}^{-\mathrm{f}}$, the extension of $\mathrm{Ho}^{-\mathrm{f}}$ with a passivation construct of the form $\widetilde{a}\{P\}$, which represents a *passivation unit* named a that contains a process P that respects the forwarding limitation of $\mathrm{Ho}^{-\mathrm{f}}$. A passivation unit is a *transparent locality*, in that there are no restrictions on the interactions between P and processes surrounding a. The execution of P can be *suspended* at an arbitrary time; these intuitions are formalized by the LTS for $\mathrm{HoP}^{-\mathrm{f}}$, which extends that of $\mathrm{Ho}^{-\mathrm{f}}$ with the two following rules

$$\frac{P \xrightarrow{\alpha} P'}{\widetilde{a}\{P\} \xrightarrow{\alpha} \widetilde{a}\{P'\}}\ \text{Loc} \qquad \widetilde{a}\{P\} \xrightarrow{\overline{a}\langle P\rangle} \mathbf{0}\ \text{Pas}\,.$$

A Faithful Encoding of Minsky Machines into $\mathbf{HoP}^{-\mathrm{f}}$. Here we investigate the expressiveness of $\mathrm{HoP}^{-\mathrm{f}}$ by exhibiting an encoding of Minsky machines into $\mathrm{HoP}^{-\mathrm{f}}$. Interestingly, unlike the encoding discussed before into $\mathrm{Ho}^{-\mathrm{f}}$, the encoding into $\mathrm{HoP}^{-\mathrm{f}}$ is *faithful*. As such, in $\mathrm{HoP}^{-\mathrm{f}}$ *both* termination and convergence are *undecidable* problems. Hence, it is fair to say that the passivation construct — even with the limitation on the shape of output processes— allows to recover the expressive power lost in restricting HOcore as $\mathrm{Ho}^{-\mathrm{f}}$.

The encoding is given in Figure 5; we now give some intuitions on it. A register k with value m is represented by a passivation unit r_k that contains the encoding of number m, denoted $(\!|\ m\ |\!)_k$. In turn, $(\!|\ m\ |\!)_k$ consists of a chain of m nested input prefixes on name u_k; it also contains other prefixes on a_1 and a_2 which are used for synchronization purposes during the execution of instructions. The encoding of zero is given by an input action on z_k that prefixes a trigger $\overline{a_z}$.

As expected, the encoding of an increment operation on the value of register k consists in the enlargement of the chain of nested input prefixes it contains. For that purpose, the content of passivation unit r_k is obtained with an input on r_k. We therefore need to recreate the passivation unit r_k with the encoding of the incremented value. Notice that we require an additional synchronization on c_k in order to "inject" such a previous content in a new passivation unit called r_k.

$$\textsc{Register}\ r_k \qquad [\![r_k = n]\!]_{\mathsf{M}} = \widetilde{r_k}\{(\!|\ n\ |\!)_k\}$$

where

$$(\!|\ n\ |\!)_k = \begin{cases} z_k.\,\overline{a_z} & \text{if } n = 0 \\ u_k.\,(\overline{a_1} \parallel a_2.\,(\!|\ n-1\ |\!)_k) & \text{if } n > 0. \end{cases}$$

$$\textsc{Instructions}\ (i : I_i)$$

$$\begin{aligned}
[\![(i : \mathtt{INC}(r_k))]\!]_{\mathsf{M}} &= \ !p_i.\,(r_k(x).\,(\overline{c_k}\langle x\rangle \parallel \widetilde{r_k}\{c_k(y).\,(\overline{a_p} \parallel u_k.\,(\overline{a_1} \parallel a_2.\,y))\} \parallel a_p.\,\overline{p_{i+1}})) \\
[\![(i : \mathtt{DECJ}(r_k, s))]\!]_{\mathsf{M}} &= \ !p_i.\,(m(x).\,x \\
&\qquad \parallel \widetilde{d}\{\overline{u_k} \parallel a_1.\,\overline{m}\langle s(x).\,d(x).\,(\overline{a_2} \parallel \overline{p_{i+1}})\rangle\} \\
&\qquad \parallel \widetilde{s}\{\overline{z_k} \parallel a_z.\,\overline{m}\langle d(x).\,s(x).\,r_k(t).\,(\widetilde{r_k}\{z_k.\,\overline{a_z}\} \parallel \overline{p_s})\rangle\})
\end{aligned}$$

Figure 5: Encoding of Minsky machines into $\mathrm{HoP}^{-\mathrm{f}}$.

This way, the chain of nested inputs in r_k can be enlarged while respecting the limitation on the shape of processes inside passivation units. As a result, the chain is enlarged by putting it behind some prefixes, and the next instruction can be invoked. This is done by a synchronization on name a_p.

The encoding of a decrement of the value of register k consists of an internal, exclusive choice implemented as two passivation units that execute in parallel: the first one, named d, implements the behavior for decrementing the value of a register, while the second one, named s, implements the behavior for performing the jump to some given instruction. Unlike the encoding of Minsky machines into $\mathrm{Ho}^{-\mathrm{f}}$, this internal choice behaves faithfully with respect to the encoding instruction, i.e., the behavior inside d will only execute if the value in r_k is greater than zero, whereas the behavior inside s will only execute if that value is equal to zero. It is indeed a deterministic choice in that it is *not* the case that both an input prefix on u_k (which triggers the "decrement branch" defined by d) and one on z_k (which triggers the "jump branch" defined by s) are available at the same time; this is because of the way in which we encode numbers, i.e., as a chain of input prefixes. In addition to the passivation units, the encoding of decrement features a "manager" (implemented as a synchronization on m) that enables the behavior of the chosen passivation unit by placing it at the top-level, and consumes both s and d afterwards, thus leaving no residual processes after performing the instruction. In case the value of the register is equal to some $n > 0$, then a decrement is implemented by consuming the input prefixes on u_k and a_2 and the output prefix on a_1 through suitable synchronizations. It is worth noticing that these synchronizatons are only possible because the passivation units containing the encoding of n behave as transparent localities, and hence able to interact with its surrounding context. As a result, the encoding of $n - 1$ remains inside r_k and the next instruc-

tion is invoked. In case the value of the register is equal to zero, the passivation unit r_k is consumed and recreated with the encoding of zero inside. The jump is then performed by invoking the respective instruction.

A correctness statement for the faithful encoding of Minsky machines into $\text{HoP}^{-\text{f}}$ is along the lines of that for the encoding into HOcore, and we omit it. Full details can be found in [26, Chapter 5]. As a consequence, we have:

Theorem 3.6. *Termination and convergence are undecidable in* $\text{HoP}^{-\text{f}}$.

4 The Expressiveness of (A)synchronous and Polyadic Communication

Here we present results on the expressive power of extensions of HOcore with restriction, concentrating 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., [25, 23]) have studied the *asynchronous* π*-calculus* [3, 13] and its relationship with the (synchronous) π-calculus. Also, the encoding of polyadic communication into monadic communication in the π-calculus [18] is simple and very robust. 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) [29]. We shall consider SHO and AHO, the synchronous and asynchronous variants of HOπ with polyadic communication (Section 4.1). 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.[2] In a process-passing setting, sharing of (private) names is thus rather limited.

[2]Here we refer to process-passing *without* passing of abstractions, i.e. functions from processes to processes. As studied in [15, 26], the situation is rather different with abstraction-passing.

We first describe results on the relationship between synchrony and asynchrony. In fact, we present an encoding of SHO^n into AHO^n. 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. We then 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. In this case, we discuss 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.

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. Such a distinction is 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.

4.1 The Calculi

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

Definition 4.1. *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:*

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

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 $|\ \widetilde{x}\ |$ 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 [29]. 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

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

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

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

Figure 6: The LTS of SHO. Symmetric rules Act2 and Tau2 are omitted.

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 6; 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 4.1 actually gives a *family* of higher-order process calculi. We have the following convention:

Convention 4.1. *For each $n > 0$, SHO^n is the calculus obtained from the syntax given in Definition 4.1 in which polyadic communication has arity at most n.*

Definition 4.2. 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 4.3 (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', $P \Rightarrow P' \downarrow_\mu$*.*

4.2 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 4.4 (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* 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* $C^N_{\mathsf{op}} \in \mathcal{P}_\mathsf{t}$ *that depends on N and* op *such that* $[\![\mathsf{op}(S_1, \ldots, S_k)]\!] = C^N_{\mathsf{op}}[[\![S_1]\!], \ldots, [\![S_k]\!]]$;
2. name invariant *if* $[\![\sigma(P)]\!] = \sigma([\![P]\!])$*, for any injective renaming of names* σ*.*

Definition 4.5 (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. 4.1 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 4.1. *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* $C[\![\cdot]\!]$ *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* $(\mathcal{D} \cdot C)[\![\cdot]\!]$ *is an encoding of* $\mathcal{L}_1$ *into* $\mathcal{L}_3$.

4.3 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 [14]; 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 4.6 (Synchronous to Asynchronous)**.** *For each $n > 0$, the encoding of* SHO^n *into* AHO^n *is defined as follows:*

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

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.

The encoding provides compelling evidence on the expressive power of (asynchronous) process-passing. One may then wonder if a similar encoding, for the case of polyadic into monadic communication, can be defined. In the next section we show that this is *not* the case.

4.4 Separation Results for Polyadic Communication

Here we present the separation results for SHO. We discuss 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.

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 4.7. *The syntax of (guarded, multihole) contexts is defined as:*

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

Definition 4.8 (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 4.9 (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 $|\widetilde{R}| = 1$ then T is said to be in* monadic disjoint form *(or* MDF*) with respect to $\widetilde{n}$, R, and P. If $|\widetilde{R}| = 0$ then T is said to be in* zero-adic disjoint form *(or* ZDF*) with respect to $\widetilde{n}$ and P.*

Proposition 4.2 (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 $[\![P]\!]$.*

4.4.1 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 4.1) and output actions (Lemma 4.2).

Lemma 4.1. *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 4.2. *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 4.3. *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'.*

4.4.2 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 4.10. *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 4.4. *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 Lemma 4.4 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 Lemma 4.4 to a sequence of internal synchronizations and output actions.

Lemma 4.5. *Let $T = \nu\widetilde{n}\,(A \parallel C[\widetilde{R}])$ be a NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and A. Suppose $T \xRightarrow{\vec{\alpha}} T'$, 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. *$T \xRightarrow{\vec{\alpha}_C} T_1 \xRightarrow{\vec{\alpha}_A} T'$.*

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 $m_1, \ldots, m_k \geq 0$.*

4.4.3 A Hierarchy of Synchronous Higher-Order Process Calculi

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 give details for the basic case of the hierarchy, namely that biadic process-passing cannot be encoded into monadic process-passing (Theorem 4.1). The proof exploits the notion of MDF and its associated stability conditions. The general result, i.e., the impossibility of encoding SHO^{n+1} into SHO^n (Theorem 4.2), results as a generalization of the base case.

Theorem 4.1. *There is no encoding of* SHO^2 *into* SHO^1.

Proof (Sketch). The proof proceeds by contradiction, assuming such an encoding does exist. Take the following SHO^2 process

$$\begin{aligned} P_0 &= \nu m_1, m_2\, (\overline{a}\langle S_1,\, S_2\rangle.\, \mathbf{0}) \parallel \\ &\quad \nu b\, (a(x_1, x_2).\, (\overline{b}\langle \overline{b_1}.\, x_1\rangle.\, \mathbf{0} \parallel \overline{b}\langle \overline{b_2}.\, x_2\rangle.\, \mathbf{0} \parallel b(y_1).\, b(y_2).\, y_1) \end{aligned}$$

where S_1, S_2 have different observable behavior. After the communication on a, P_0 evolves into a P; it can be shown that the SHO^1 process $[\![P]\!]$ is in MDF. Once in P, either S_1 or S_2 is executed; this depends on a mutually exclusive choice that relies on the private name b. Notice that P only involves output actions and *internal* synchronizations. By completeness, P and $[\![P]\!]$ should be behaviorally equivalent; during the bisimulation game, since output actions and *internal* synchronizations preserve MDFs (Lemmas 4.1 and 4.2), $[\![P]\!]$ (and its derivatives) will be in MDF as well. Using Lemma 4.5 it is possible to formalize the fact that certain causality properties of the source term are lost in its encoding. As a result, the encoded term can exhibit observable behavior that is different from the observable behavior in the source term, thus leading to a fail in the bisimulation game and therefore to a contradiction. □

The scheme used in the proof of Theorem 4.1 can be generalized for calculi with arbitrary polyadicity. Therefore we have the following.

Theorem 4.2. *For every* $n > 1$, *there is no encoding of* SHO^n *into* SHO^{n-1}.

5 Final Remarks

We have presented expressiveness and decidability results for HOcore, a *core calculus* for higher-order concurrency. Previous works on the expressiveness of higher-order calculi (e.g. [27]) have opted for an *indirect* approach, as they have largely relied on translations into a first-order language. However, as argued in the

introduction, higher-order communication as required in emerging applications in concurrency is often difficult (and sometimes impossible) to represent into a first-order setting. This observation has motivated a *direct* approach to expressiveness, in which basic properties of higher-order process calculi are studied directly on them, exploiting their peculiarities. Such an approach is most convenient when addressing issues related to *expressiveness* and *decidability*, two aspects of the theory of calculi for higher-order concurrency that have been little explored.

We have concentrated on the study of the expressiveness of a handful of concerns in the higher-order setting. They include name restriction, the shape and capabilities of the objects in higher-order communications, forms of process suspension, polyadic and asynchronous communication. Seen as a whole, they all reveal aspects of higher-order concurrency that could shed light on the design of abstract languages for concurrency involving a process-passing discipline. We are of the opinion that calculi for emerging applications should result from the careful combination of higher-order constructs and first-order features.

We regard HOcore as a framework for the study of the fundamental studies of higher-order concurrency. While our interest has been the study of expressiveness and decidability issues, HOcore is compact and versatile enough to be exploited for other purposes. As a matter of fact, HOcore and/or variants of it have shown useful in the study of the behavioral theory of higher-order process calculi [16], and in the development of type systems for higher-order concurrent languages [6].

Acknowledgments. I am grateful to Davide Sangiorgi, who supervised the thesis work on which this survey is based on, for helpful comments on a draft of this paper. This research was carried out while the author was a PhD student at University of Bologna. This research has been partially supported by INRIA Equipe Associée BACON and by FCT / MCTES (CMU-PT/NGN44-2009-12).

References

[1] Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inf. Comput.*, 160(1-2):109–127, 2000.

[2] Gérard Boudol. Towards a lambda-calculus for concurrent and communicating systems. In *Proc. of TAPSOFT, Vol.1*, volume 351 of *Lecture Notes in Computer Science*, pages 149–161. Springer, 1989.

[3] Gérard Boudol. Asynchrony and the π-calculus (note). Technical report, Rapport de Recherche 1702, INRIA, Sophia-Antipolis, 1992.

[4] Mikkel Bundgaard, Arne J. Glenstrup, Thomas T. Hildebrandt, Espen Højsgaard, and Henning Niss. Formalizing higher-order mobile embedded business processes

with binding bigraphs. In *Proc. of COORDINATION*, volume 5052 of *Lecture Notes in Computer Science*, pages 83–99. Springer, 2008.

[5] Nadia Busi, Maurizio Gabbrielli, and Gianluigi Zavattaro. On the expressive power of recursion, replication and iteration in process calculi. *Math. Struct. in Comp. Sci.*, 19(6):1191–1222, 2009.

[6] Romain Demangeon, Daniel Hirschkoff, and Davide Sangiorgi. Termination in higher-order concurrent calculi. In *Proc. of FSEN'09*, volume 5961 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2010.

[7] Cinzia Di Giusto, Jorge A. Pérez, and Gianluigi Zavattaro. On the expressiveness of forwarding in higher-order communication. In *Proc. of ICTAC*, volume 5684 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2009.

[8] Alain Finkel. Reduction and covering of infinite reachability trees. *Inf. Comput.*, 89(2):144–179, 1990.

[9] Alain Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

[10] Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. In *Proc. of CONCUR*, volume 5201 of *Lecture Notes in Computer Science*, pages 492–507. Springer, 2008.

[11] Thomas Hildebrandt, Jens Chr. Godskesen, and Mikkel Bundgaard. Bisimulation congruences for Homer — a calculus of higher order mobile embedded resources. Technical Report TR-2004-52, IT University of Copenhagen, 2004.

[12] Daniel Hirschkoff and Damien Pous. A distribution law for ccs and a new congruence result for the p-calculus. *Logical Methods in Computer Science*, 4(2), 2008.

[13] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *Proc. of ECOOP*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 1991.

[14] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. In *Proc. of LICS'08*, pages 145–155. IEEE Computer Society, 2008.

[15] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness of polyadic and synchronous communication in higher-order process calculi. In *Proc. of ICALP 2010*. Springer, 2010. To appear.

[16] Sergueï Lenglet, Alan Schmitt, and Jean-Bernard Stefani. Normal bisimulations in calculi with passivation. In *Proc. of FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 257–271. Springer, 2009.

[17] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[18] Robin Milner. The Polyadic pi-Calculus: A Tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh, 1991.

[19] Robin Milner and Faron Moller. Unique decomposition of processes. *Theor. Comput. Sci.*, 107(2):357–363, 1993.

[20] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Inf. Comput.*, 100(1):1–40, 1992. A preliminary version appeared as Technical Report ECS-LFCS-89-85, LFCS, University of Edinburgh, June 1989.

[21] Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[22] Faron Moller. *Axioms for Concurrency*. PhD thesis, University of Edinburgh, Dept. of Comp. Sci., 1989. PhD thesis CST–59–89.

[23] Uwe Nestmann. What is a "good" encoding of guarded choice? *Inf. Comput.*, 156(1-2):287–319, 2000. A preliminary version appeared in EXPRESS'97.

[24] Flemming Nielson. The typed lambda-calculus with first-class processes. In *Proc. of PARLE (2)*, volume 366 of *Lecture Notes in Computer Science*, pages 357–373. Springer, 1989.

[25] Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003. Extended abstract in Proc. of POPL'97.

[26] Jorge A. Pérez. *Higher-Order Concurrency: Expressiveness and Decidability Results*. PhD thesis, University of Bologna, 2010. Draft in `www.japerez.phipages.com/`.

[27] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, University of Edinburgh, Dept. of Comp. Sci., 1992.

[28] Davide Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Inf. Comput.*, 131(2):141–178, 1996.

[29] Davide Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theor. Comput. Sci.*, 167(2):235–274, 1996.

[30] Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. In *Proc. of LICS'07*, pages 293–302. IEEE Computer Society, 2007.

[31] Alan Schmitt and Jean-Bernard Stefani. The kell calculus: A family of higher-order distributed process calculi. In *Proc. of Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 146–178. Springer, 2004.

[32] Bent Thomsen. A calculus of higher order communicating systems. In *Proc. of POPL'89*, pages 143–154. ACM Press, 1989.

[33] Bent Thomsen. *Calculi for Higher Order Communicating Systems*. PhD thesis, Dept. of Comp. Sci., Imperial College, 1990.

[34] Bent Thomsen. Plain CHOCS: A second generation calculus for higher order processes. *Acta Inf.*, 30(1):1–59, 1993.