23rd Annual IEEE Symposium on Logic in Computer Science

On the Expressiveness and Decidability of Higher-Order Process Calculi *

Ivan Lanese Jorge A. Pérez Davide Sangiorgi
University of Bologna, Italy
{lanese, perez, davide.sangiorgi}@cs.unibo.it

Alan Schmitt
INRIA Rhône-Alpes, France
alan.schmitt@inrialpes.fr

Abstract

In higher-order process calculi *the values exchanged in communications may contain processes. A core calculus of higher-order concurrency is studied; it has only the operators necessary to express higher-order communications: input prefix, process output, and parallel composition. By exhibiting a nearly deterministic encoding of Minsky machines, the calculus is shown to be Turing complete and therefore its termination problem is undecidable. Strong bisimilarity, however, is shown to be decidable. Further, the main forms of strong bisimilarity for higher-order processes (higher-order bisimilarity, context bisimilarity, normal bisimilarity, barbed congruence) coincide. They also coincide with their asynchronous versions. A sound and complete axiomatization of bisimilarity is given. Finally, bisimilarity is shown to become undecidable if at least four static (i.e., top-level) restrictions are added to the calculus.*

1. Introduction

Higher-order process calculi are calculi in which processes (more generally, values containing processes) can be communicated. Higher-order process calculi have been put forward in the early 90s, with CHOCS [22] and Plain CHOCS [24], the Higher-Order π-calculus [15], and others. The basic operators are usually those of CCS: parallel composition, input and output prefix, and restriction. Replication and recursion are often omitted as they can be encoded. However, the possibility of exchanging processes has strong consequences on semantics: ordinary definitions of bisimulation and behavioral equivalences become unsatisfactory, and labelled transition systems must deal with higher-order substitutions and scope extrusion. 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 for the description of mobile systems. Higher-order calculi are formally closer to, and are inspired by, the λ-calculus, whose basic computational step — β-reduction — involves term instantiation. As in the λ-calculus, a computational step in higher-order calculi results in the instantiation of a variable with a term, which is then copied as many times as there are occurrences of the variable, resulting in potentially larger terms.

The expressiveness of higher-order communication has received little attention in the literature. Higher-order calculi (both sequential and concurrent) have been compared with first-order calculi, but mainly as a way of investigating the expressiveness of π-calculus and similar formalisms. Thomsen [23] and Xu [25] have proposed encodings of π-calculus into Plain CHOCS. These encodings make essential use of the relabeling operator of Plain CHOCS. Sangiorgi and Walker's encoding of a variant of π-calculus into Higher-Order π-calculus [19] relies on the abstraction mechanism of the Higher-Order π-calculus (it needs ω-order abstractions). Another strand of work on expressiveness (see, e.g., [13]) has looked at calculi for distributed systems and compared different primitives for migration and movement of processes (or entire locations), which can be seen as higher-order constructs.

The goal of this paper is to contribute to the understanding of expressiveness and behavioral equivalence in higher-order process calculi. We consider a core calculus of Higher-Order processes (briefly HOcore), whose grammar is:

$$P ::= a(x).\, P \;\Big|\; \overline{a}\langle P\rangle \;\Big|\; P \parallel P \;\Big|\; x \;\Big|\; \mathbf{0}$$

An input prefixed process $a(x).\, P$ can receive on name (or channel) 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. We can view the calculus 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. HOcore is *minimal*, in that only the operators strictly necessary to obtain higher-order communications

*Research partially supported by European Project FET-GC II IST-2005-16004 SENSORIA, Italian MIUR Project n. 2005015785, "Logical Foundations of Distributed Systems and Mobile Code", and French ANR project "CHOCO".

1043-6871/08 $25.00 © 2008 IEEE
DOI 10.1109/LICS.2008.8

are retained. For instance, continuations following output messages have been left out. More importantly, HOcore has no restriction operator. Thus all channels are global, and dynamic creation of new channels is forbidden. This makes also the absence of recursion relevant, as known encodings of fixed-point combinators in higher-order process calculi require the restriction operator.

Even though HOcore is minimal, it remains non-trivial: in Section 3 we show that it is Turing complete, and therefore its termination problem is undecidable, by exhibiting a nearly deterministic encoding of Minsky machines. The cornerstone of the encoding, counters that may be tested for zero, consist of nested higher-order outputs. Each register is made of two mutually recursive behaviors capable of spawning processes incrementing and decrementing its counter.

We then turn to the question of definability and decidability of bisimilarity. As hinted at above, the definition of a satisfactory notion of bisimilarity is a hard problem for a higher-order process language, and the "term-copying" feature inherited from the λ-calculus can make it hard to prove that bisimilarity is a congruence. In ordinary bisimilarity, as in CCS, two processes are bisimilar if any action by one of them can be matched by an equal action from the other in such a way that the resulting derivatives are again bisimilar. The two matching actions must be syntactically *identical*. This condition is unacceptable in higher-order concurrency; for instance it breaks fundamental algebraic laws such as the commutativity of parallel composition. Alternative proposals of labelled bisimilarity for higher-order processes have been put forward. In *higher-order bisimilarity* [23, 15] one requires bisimilarity, rather than identity, of the processes emitted in a higher-order output action. This weakening is natural for higher-order calculi and the bisimulation checks involved are simple. However, higher-order bisimilarity is often over-discriminating as a behavioral equivalence [15], and basic properties, such as congruence, may be very hard to establish. *Context bisimilarity* [15, 8] avoids the separation between the argument and the continuation of an output action, this continuation being either explicit or consisting of other processes running in parallel, by explicitly taking into account the context in which the emitted process is supposed to go. Context bisimilarity yields more satisfactory process equalities, and it coincides with contextual equivalence (i.e., barbed congruence). A drawback is the universal quantification over contexts in the clause for output actions, which can make it difficult, in practice, to check equivalences. *Normal bisimilarity* [15, 8, 1] is a simplification of context bisimilarity without universal quantifications in the output clause. The input clause is simpler too: normal bisimilarity can indeed be viewed as a form of *open bisimilarity* [16], where the formal parameter of an input is not substituted in the input clause, and free variables of terms are observable during the bisimulation game. However, the definition of the bisimilarity may depend on the operators in the calculus, and the correspondence with context bisimilarity may be hard to prove.

In Sections 4 and 5 we show that HOcore has a unique reasonable relation of strong bisimilarity: all above forms (higher-order bisimilarity, context bisimilarity, normal bisimilarity, barbed congruence) coincide; and they also coincide with their asynchronous versions. Further, we show that such a bisimilarity relation is decidable.

We find, in the concurrency literature, examples of formalisms that are not Turing complete and where nevertheless (strong) bisimilarity is undecidable (e.g., Petri nets [7], lossy channel systems [20]). We are not aware however of examples of the opposite situation; that is, formalisms that, as HOcore, are Turing complete but at the same time maintain decidability of bisimilarity. The situation in HOcore may indeed seem surprising, if not even contradictory: one is able to tell whether two processes are bisimilar, but in general one cannot tell whether the processes will terminate or even whether the sets of their τ-derivatives (the processes obtained via reductions) are finite or not. The crux to obtaining decidability is a further characterization of bisimilarity in HOcore, as a form of open bisimilarity, called IO bisimilarity, in which τ-transitions are ignored.

For an upper bound to the complexity of the bisimilarity problem, we can adapt Dovier et al.'s algorithm [3] to infer that bisimilarity is decidable in time which is linear in the size of the (open and higher-order) transition system underlying IO bisimilarity. In general however this transition system is exponential with respect to the size of the root process. We show in Section 6 that bisimilarity in HOcore can actually be decided in time that is polynomial with respect to the size of the initial pair of processes. We obtain this through an axiomatization of bisimilarity, where we adapt to a higher-order setting both Moller and Milner's unique decomposition of processes [11] and Hirschkoff and Pous' axioms for a fragment of (finite) CCS [4].

The decidability result for bisimilarity breaks down with the addition of restriction, as full recursion can then be faithfully encoded (the resulting calculus subsumes, e.g., CCS without relabeling). This however requires the ability of generating unboundedly many new names (for instance, when a process that contains restrictions is communicated and copied several times). In Section 7, we consider the addition of static restrictions to HOcore. Intuitively this means allowing restrictions only as the outermost constructs, so that processes take the form $\boldsymbol{\nu} a_1 \dots \boldsymbol{\nu} a_n\, P$ where the inner process P is restriction-free. Via an encoding of the Post correspondence problem we show that the addition of four static restrictions is sufficient to produce undecidability. We do not know what happens with fewer restrictions.

In the final part of the paper we examine the impact of some extensions to HOcore on our decidability results (Section 8) and give some concluding remarks (Section 9).

Due to a lack of space, most proofs are only sketched or ommited; they are included in the extended version of this paper [10].

2. The calculus

We now introduce HOcore, the core of calculi for higher-order concurrency such as CHOCS [22], Plain CHOCS [24], and Higher-Order π-calculus [15, 17]. 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 $a(x).\,P$ binds the free occurrences of x in P. 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. We abbreviate $a(x).\,P$, with $x \notin \mathsf{fv}(P)$, as $a.\,P$, $\overline{a}\langle\mathbf{0}\rangle$ as $\overline{a}$, and $P_1 \parallel \ldots \parallel P_k$ as $\prod_{i=1}^{k} P_i$.

The Labelled Transition System of HOcore is defined on open processes. There are three forms of transitions: τ 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 P emits P' at a, and in doing so it evolves to P''. We use α to indicate a generic label of a transition.

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

$$
\textsc{Act1}\quad \frac{P_1 \xrightarrow{\alpha} P_1' \qquad \mathsf{bv}(\alpha)\cap\mathsf{fv}(P_2)=\emptyset}{P_1 \parallel P_2 \xrightarrow{\alpha} P_1' \parallel P_2}
$$

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

(We have omitted $\textsc{Act2}$ and $\textsc{Tau2}$, the symmetric counterpart of the last two rules.)

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 $P \equiv\xrightarrow{\tau}\equiv P'$.

$$
\textsc{M-Inc}\quad \frac{i : \mathtt{INC}(r_j) \qquad m_j' = m_j + 1 \qquad m_{1-j}' = m_{1-j}}{(i, m_0, m_1) \longrightarrow_{\mathrm{M}} (i+1, m_0', m_1')}
$$

$$
\textsc{M-Dec}\quad \frac{\begin{array}{c} i : \mathtt{DECJ}(r_j, k) \\ m_j \neq 0 \qquad m_j' = m_j - 1 \qquad m_{1-j}' = m_{1-j} \end{array}}{(i, m_0, m_1) \longrightarrow_{\mathrm{M}} (i+1, m_0', m_1')}
$$

$$
\textsc{M-Jmp}\quad \frac{i : \mathtt{DECJ}(r_j, k) \quad m_j = 0}{(i, m_0, m_1) \longrightarrow_{\mathrm{M}} (k, m_0, m_1)}
$$

Figure 1. Reduction of Minsky machines

3. HOcore is Turing complete

We present in this section an encoding of Minsky machines [12] into HOcore. The encoding shows that HOcore is Turing complete and, as the encoding preserves termination, it also shows that termination in HOcore is undecidable. The only form of non-determinism in the encoding is due to possible unfoldings of (the encoding of) recursive definitions after they have been used; otherwise, at any step, in the encoding any process has at most one reduction.

3.1. Minsky machines

A Minsky machine is a Turing complete model composed of a set of sequential, labeled instructions, and two registers. Registers r_j ($j \in \{0, 1\}$) can hold arbitrarily large natural numbers. Instructions $(1 : I_1), \ldots, (n : I_n)$ can be of two kinds: $\mathtt{INC}(r_j)$ adds 1 to register r_j and proceeds to the next instruction; $\mathtt{DECJ}(r_j, k)$ jumps to instruction k if r_j is zero, otherwise it decreases register r_j by 1 and proceeds to the next instruction.

A Minsky machine includes a program counter p indicating the label of the instruction being executed. In its initial state, the machine has both registers set to 0 and the program counter p set to the first instruction. The Minsky machine stops whenever the program counter is set to a non-existent instruction, i.e. $p > n$.

A *configuration* of a Minsky machine is a tuple (i, m_0, m_1); it consists of the current program counter and the values of the registers. Formally, the reduction relation over configurations of a Minsky machine, denoted $\longrightarrow_{\mathrm{M}}$, is defined in Figure 1.

3.2. Minsky machines in HOcore

The encoding of a Minsky machine into HOcore is denoted as $[\![\cdot]\!]_{\mathsf{M}}$. We first present an encoding of a simple form of guarded choice and guarded replication. We then show

how to count and test for zero in HOcore and present the main encoding, depicted in Table 1.

Guarded choice. We extend the HOcore syntax with a simple form of guarded choice to choose between different behaviors. Assume, for instance, that a_i should trigger P_i, for $i \in \{1, 2\}$. We write this as $a_1. P_1 + a_2. P_2$, and we write the choice of the behavior P_i as $\widehat{a_i}$. We then have, for each i, the reduction $(a_1. P_1 + a_2. P_2) \parallel \widehat{a_i} \longrightarrow P_i$. We encode $a_1. P_1 + a_2. P_2$ as

$$[\![a_1. P_1 + a_2. P_2]\!]_+ = \overline{a_1}\langle [\![P_1]\!]_+\rangle \parallel \overline{a_2}\langle [\![P_2]\!]_+\rangle$$

the choice $[\![\widehat{a_1}]\!]_+$ as $a_2(x_2). a_1(x_1). x_1$, and the choice $[\![\widehat{a_2}]\!]_+$ as $a_1(x_1). a_2(x_2). x_2$. This way, $[\![\widehat{a_i}]\!]_+$ for $i \in \{1, 2\}$ is a process that consumes both P_i's and spawns the one chosen. The translation is an homomorphism on the other operators. This encoding is correct as long as all guards used in the choices are different and there is at most one message at a guard, $\widehat{a_1}$ or $\widehat{a_2}$ in the previous example, enabled at any given time. The encoding introduces an extra communication for every guarded choice.

Input-guarded replication. We follow the standard encoding of replication in higher-order process calculi, adapting it to input-guarded replication so to make sure that diverging behaviors are not introduced. As there is no restriction in HOcore, the encoding is not compositional and replications cannot be nested.

Definition 3.1. *Assume a fresh name c. The encoding of* input-guarded replication *is as follows:*

$$[\![!a(z). P]\!]_{\text{i!}} = a(z). (Q_c \parallel P) \parallel \overline{c}\langle a(z). (Q_c \parallel P)\rangle$$

where $Q_c = c(x). (x \parallel \overline{c}\langle x\rangle)$, P contains no replications (nested replications are forbidden), and $[\![\cdot]\!]_{\text{i!}}$ is an homomorphism on the other process constructs in HOcore.

This encoding preserves termination.

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 $\widehat{n_j}$ indicating the number is not zero. The encoding of zero comprises the message $\overline{r_j^0}$ and a flag $\widehat{z_j}$ indicating it is zero. As an example, $(\!|\ 2\ |\!)_j$ is $\overline{r_j^{\mathsf{S}}}\langle \overline{r_j^{\mathsf{S}}}\langle \overline{r_j^0} \parallel \widehat{z_j}\rangle \parallel \widehat{n_j}\rangle \parallel \widehat{n_j}$. To increment $(\!|\ 2\ |\!)_j$, one puts it as the argument of a message on r_j^{S} along with the $\widehat{n_j}$ choice. To decrement it, one triggers the behavior associated to the $\widehat{n_j}$ choice and consumes the message on r_j^{S}, spawning its contents.

INSTRUCTIONS $(i : I_i)$

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

REGISTERS r_j

$$\begin{aligned}
[\![r_j = 0]\!]_{\mathsf{M}} \;&=\; (inc_j. (\overline{r_j^{\mathsf{S}}}\langle (\!|\ 1\ |\!)_j\rangle \parallel \overline{ack}) \;+ \\
&\qquad dec_j. ((\!|\ 0\ |\!)_j \parallel \overline{ack})) \parallel \mathsf{REG}_j \\
[\![r_j = m]\!]_{\mathsf{M}} \;&=\; (inc_j. (\overline{r_j^{\mathsf{S}}}\langle (\!|\ m\ |\!)_j\rangle \parallel \overline{ack}) \;+ \\
&\qquad dec_j. ((\!|\ m-1\ |\!)_j \parallel \overline{ack})) \parallel \mathsf{REG}_j
\end{aligned}$$

where:

$$\begin{aligned}
\mathsf{REG}_j \;=\; & !r_j^0. (inc_j. (\overline{r_j^{\mathsf{S}}}\langle (\!|\ 1\ |\!)_j\rangle \parallel \overline{ack}) \;+ \\
& \quad dec_j. ((\!|\ 0\ |\!)_j \parallel \overline{ack})) \parallel \\
& !r_j^{\mathsf{S}}(Y). (inc_j. (\overline{r_j^{\mathsf{S}}}\langle \overline{r_j^{\mathsf{S}}}\langle Y\rangle \parallel \widehat{n_j}\rangle \parallel \overline{ack}) \;+ \\
& \quad dec_j. (Y \parallel \overline{ack}))
\end{aligned}$$

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

Table 1. Encoding of Minsky machines

Registers. Registers are counters that may be incremented and decremented. We distinguish registers whose current value is zero as their value should remain zero after a decrement, as specified by the Msemantics. A register j consists of two mutually recursive behaviors, triggered respectively by messages on r_j^0 and r_j^{S}. Each of them spawns a process waiting for an increment or a decrement choice. In case of an increment, a new process is spawned using a message on r_j^{S} containing the successor of the current register value, along with an acknowledgment. In case of a decrement of a non-zero register, the current encoding of the number is spawned, resulting in the recreation of the register process with the decremented value and the spawning of a non-zero flag $\widehat{n_j}$. In case of the decrement of a zero register, the zero register is recreated and a zero flag $\widehat{z_j}$ is spawned. In both cases an acknowledgment is sent.

Instructions. The encoding of instructions goes hand in hand with the encoding of registers. 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 the next instruction otherwise.

The encoding of a configuration of a Minsky machine thus requires a finite number of fresh names (linear on n, the number of instructions).

Definition 3.2. *Let N be a Minsky machine with registers $r_0 = m_0$, $r_1 = m_1$ and instructions $(1 : I_1), \ldots, (n : I_n)$. Suppose fresh, pairwise different names r_j^0, r_j^{S}, $p_1, \ldots, p_n$, inc_j, dec_j, ack (for $j \in \{0, 1\}$). Given the encodings in Table 1, a configuration (i, m_0, m_1) of N is encoded as*

$$\overline{p_i} \parallel [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathsf{M}} .$$

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 $\longrightarrow_{\mathrm{M}}^*$ is the reflexive and transitive closure of $\longrightarrow_{\mathrm{M}}$, and $N \Uparrow_{\mathrm{M}}$ means that N has an infinite sequence of reductions.

Lemma 3.3. *Let N be a Minsky machine. We have:*

1. *$N \longrightarrow_{\mathrm{M}}^* N'$ 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 $[\![N]\!]_{\mathsf{M}} \Uparrow$.*

Proof. We show there is a tight relationship between the execution of N and an invariant structure of its encoding, the only non-determinism coming from choosing when to unfold recursive processes after their use. One can then show that every finite computation of the Minsky machine is mimicked by a finite computation of the encoding. □

The results above guarantee that HOcore is Turing complete, and since the encoding preserves termination, it entails the following corollary.

Corollary 3.4. *Termination in HOcore is undecidable.*

4. Bisimilarity in HOcore

In this section we prove that the main forms of strong bisimilarity for higher-order process calculi coincide in HOcore, and that such a relation is decidable. As a key ingredient for our results, we introduce *open Input/Output (IO) bisimulation* in which the variable of input prefixes is never instantiated and τ-transitions are not observed. We define different kinds of bisimulations by appropriate combinations of the clauses below.

Definition 4.1 (HOcore bisimulation clauses, open processes)**.** *A symmetric relation $\mathcal{R}$ on HOcore processes is*

1. *a* τ-bisimulation *if $P \;\mathcal{R}\; Q$ and $P \xrightarrow{\tau} P'$ imply that there is Q' s.t. $Q \xrightarrow{\tau} Q'$ and $P' \;\mathcal{R}\; Q'$;*
2. *a* higher-order output bisimulation *if $P \;\mathcal{R}\; Q$ and $P \xrightarrow{\overline{a}\langle P'' \rangle} P'$ imply that there are Q', Q'' s.t. $Q \xrightarrow{\overline{a}\langle Q'' \rangle} Q'$ with $P' \;\mathcal{R}\; Q'$ and $P'' \;\mathcal{R}\; Q''$;*
3. *an* output normal bisimulation *if $P \;\mathcal{R}\; Q$ and $P \xrightarrow{\overline{a}\langle P'' \rangle} P'$ imply that there are Q', Q'' s.t. $Q \xrightarrow{\overline{a}\langle Q'' \rangle} Q'$ with $m.\, P'' \parallel P' \;\mathcal{R}\; m.\, Q'' \parallel Q'$, where m is fresh.*
4. *an* open *bisimulation if whenever $P \;\mathcal{R}\; Q$:*
   - *$P \xrightarrow{a(x)} P'$ implies that there is Q' s.t. $Q \xrightarrow{a(x)} Q'$ and $P' \;\mathcal{R}\; Q'$,*
   - *$P \equiv x \parallel P'$ implies that there is Q' s.t. $Q \equiv x \parallel Q'$ and $P' \;\mathcal{R}\; Q'$.*

Definition 4.2 (HOcore bisimulation clauses, closed processes)**.** *A symmetric relation $\mathcal{R}$ on closed HOcore processes is*

1. *an* output context bisimulation *if $P \;\mathcal{R}\; Q$ and $P \xrightarrow{\overline{a}\langle P'' \rangle} P'$ imply that there are Q', Q'' s.t. $Q \xrightarrow{\overline{a}\langle Q'' \rangle} Q'$ and for all S with $\mathsf{fv}(S) \subseteq x$, it holds that $S\{P''/x\} \parallel P' \;\mathcal{R}\; S\{Q''/x\} \parallel Q'$;*
2. *an* input normal bisimulation *if $P \;\mathcal{R}\; Q$ and $P \xrightarrow{a(x)} P'$ imply that there is Q' s.t. $Q \xrightarrow{a(x)} Q'$ and $P'\{\overline{m}\langle \mathbf{0} \rangle/x\} \;\mathcal{R}\; Q'\{\overline{m}\langle \mathbf{0} \rangle/x\}$, where m is fresh;*
3. closed *if $P \;\mathcal{R}\; Q$ and $P \xrightarrow{a(x)} P'$ imply that there is Q' s.t. $Q \xrightarrow{a(x)} Q'$ and for all closed R, it holds that $P'\{R/x\} \;\mathcal{R}\; Q'\{R/x\}$.*

A combination of the bisimulation clauses in Definitions 4.1 and 4.2 is *complete* if it includes exactly one clause for input and output transitions (in contrast, it need not include a clause for τ-transitions).[1] We will show that all complete combinations coincide. We only give a name to those combinations that represent known forms of bisimulation for higher-order processes or that are needed in our proofs. In each case, as usual, a *bisimilarity* is the union of all bisimulations, and is itself a bisimulation (the functions from relations to relations that represent the bisimulation clauses in Definitions 4.1 and 4.2 are all monotonic).

Definition 4.3. Higher-order bisimilarity*, written $\sim_{\mathtt{HO}}$, is the largest relation on closed HOcore processes that is a τ-bisimulation, a higher-order output bisimulation, and is closed.*

Context bisimilarity*, written $\sim_{\mathtt{CON}}$, is the largest relation on closed HOcore processes that is a τ-bisimulation, an output context bisimulation, and is closed.*

Normal bisimilarity*, written $\sim_{\mathtt{NOR}}$, is the largest relation on closed HOcore processes that is a τ-bisimulation, an*

[1] The clauses of Definition 4.2 are however tailored to closed processes, therefore combining them with the open clause of Definition 4.1(4) has little interest.

output normal bisimulation, and an input normal bisimulation.

IO bisimilarity*, written* $\sim^{\circ}_{\mathtt{IO}}$*, is the largest relation on HOcore processes that is a higher-order output bisimulation and is open.*

Open normal bisimilarity*, written* $\sim^{\circ}_{\mathtt{NOR}}$*, is the largest relation on HOcore processes that is a* τ*-bisimulation, an output normal bisimulation, and is open.*

Given a bisimilarity $\mathcal{R}$ on closed processes, its extension to open processes is defined by

$$\{(P,Q) :\ a(x_1).\cdots.a(x_n).\,P\,\mathcal{R}\,a(x_1).\cdots.a(x_n).\,Q\}$$

with $\mathsf{fv}(P) \cup \mathsf{fv}(Q) = \{x_1,\ldots,x_n\}$, and a fresh in P, Q.

Environmental bisimilarity [18], a recent proposal of bisimilarity for higher-order calculi, in HOcore roughly corresponds to (and indeed coincides with) the complete combination that is a τ-bisimulation, an output normal bisimulation, and is closed.

Remark 4.4. *The input clause of Definition 4.2(3) is in the* late *style. It is known [15] that in calculi of pure higher-order concurrency early and late clauses are equivalent.*

Remark 4.5. *In contrast with ordinary normal bisimulation [15, 8], our clause for output normal bisimulation does not use a replication in front of the fresh name introduced. Such a replication would be needed in extensions of the calculus (e.g., with recursion or restriction).*

The simplest complete form of bisimilarity is $\sim^{\circ}_{\mathtt{IO}}$. Not only $\sim^{\circ}_{\mathtt{IO}}$ is the less demanding for proofs; it also has a straightforward proof of congruence. This is significant because congruence is notoriously a hard problem in bisimilarities for higher-order calculi.

Lemma 4.6. $\sim^{\circ}_{\mathtt{IO}}$ *is a congruence relation.*

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 4.7. $\sim^{\circ}_{\mathtt{IO}}$ *is preserved by substitutions: i.e., 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 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. □

The most striking property of $\sim^{\circ}_{\mathtt{IO}}$ is its decidability. 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 4.8. *Relation* $\sim^{\circ}_{\mathtt{IO}}$ *is decidable.*

Next we show that $\sim^{\circ}_{\mathtt{IO}}$ is also τ-preserving. This will allow us to prove that $\sim^{\circ}_{\mathtt{IO}}$ coincides with other bisimilarities, and to transfer to them its properties, in particular congruence and decidability.

Lemma 4.9. *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 (i.e., it is an asynchronous calculus) the two transitions from Q can be combined into a τ-transition, which matches the initial τ-transition from P. We conclude using Lemmas 4.6 and 4.7. □

Corollary 4.10. $\sim_{\mathtt{HO}}$ *and* $\sim^{\circ}_{\mathtt{IO}}$ *coincide.*

Proof (Sketch). The hard implication is the one right to left. One shows that $\sim^{\circ}_{\mathtt{IO}}$, restricted to closed processes, is a higher-order bisimulation. The clause for output actions is trivial (they are the same); for inputs, we apply Lemma 4.7; for τ we apply Lemma 4.9. □

We thus infer that $\sim_{\mathtt{HO}}$ is a congruence relation. A direct proof of this result (by exhibiting an appropriate bisimulation), in particular congruence for parallel composition, would have been harder. Congruence of higher-order bisimilarity is usually proved by appealing to, and adapting, Howe's method for the λ-calculus [6].

For the remaining characterizations we first establish a few properties of normal bisimulation.

Lemma 4.11. *If* $m.\,P_1 \parallel P_2 \sim^{\circ}_{\mathtt{NOR}} m.\,Q_1 \parallel Q_2$ *with* m *fresh for* P_i, Q_i *(*$i = 1,2$*), then we have* $P_i \sim^{\circ}_{\mathtt{NOR}} Q_i$ *(*$i = 1,2$*).*

Proof (Sketch). We get $P_2 \sim^{\circ}_{\mathtt{NOR}} Q_2$ by maintaining the prefix at m, and thus preventing runs of processes P_1 and Q_1.

We can also consume entirely the processes P_2 and Q_2 (by observing only their inputs or outputs), then consume prefix m; we are thus left with P_1 and Q_1, which should therefore be bisimilar. □

Lemma 4.12. *Relations* $\sim_{\mathtt{HO}}$*,* $\sim^{\circ}_{\mathtt{NOR}}$ *and* $\sim_{\mathtt{CON}}$ *coincide on HOcore.*

Proof (Sketch). One shows the following implications (on open processes): $\sim_{\mathtt{HO}}$ implies $\sim_{\mathtt{CON}}$ (this is essentially a consequence of the congruence of $\sim_{\mathtt{HO}}$); $\sim_{\mathtt{CON}}$ implies $\sim_{\mathtt{NOR}}$; $\sim_{\mathtt{NOR}}$ implies $\sim^{\circ}_{\mathtt{NOR}}$; $\sim^{\circ}_{\mathtt{NOR}}$ implies $\sim^{\circ}_{\mathtt{IO}}$ (here we use Lemma 4.11); $\sim^{\circ}_{\mathtt{IO}}$ implies $\sim_{\mathtt{HO}}$ (Corollary 4.10). □

We then extend the result to all complete combinations of the HOcore bisimulation clauses (Definitions 4.1 and 4.2).

Theorem 4.13. *All complete combinations of the HOcore bisimulation clauses coincide, and are decidable.*

Proof (Sketch). In Lemma 4.12 we have proved that the least demanding combination ($\sim^{\circ}_{\mathtt{IO}}$) coincides with the most demanding ones ($\sim_{\mathtt{HO}}$ and $\sim_{\mathtt{CON}}$). Decidability then follows from Lemma 4.8. □

5. Barbed congruence and asynchronous equivalences

We now show that the labeled bisimilarities of Section 4 coincide with *barbed congruence*, the form of contextual equivalence used in concurrency to justify bisimulation-like relations. Below we use *reduction-closed* barbed congruence [5, 19], as this makes some technical details simpler; however the results also hold for ordinary barbed congruence. More importantly, we consider the *asynchronous* version of barbed congruence, where barbs are only produced by output messages; in synchronous barbed congruence inputs may also contribute. We use the asynchronous version for two reasons. First, asynchronous barbed congruence is a weaker relation, which makes the results stronger (they imply the corresponding results for the synchronous relation). Second, asynchronous barbed congruence is more natural in HOcore because it is asynchronous — it has no output prefix.

Note also that the labeled bisimilarities of Section 4 have been defined in the synchronous style. In an *asynchronous labeled bisimilarity* the input clause is weakened so to allow, in certain conditions, an input action to be matched also by a τ-action. For instance, input normal bisimulation (Definition 4.2(2)) would become:

- if $P \xrightarrow{a(x)} P'$ then
  1. either $Q \xrightarrow{a(x)} Q'$ and $P'\{\overline{m}\langle \mathbf{0}\rangle/x\}$ $\mathcal{R}$ $Q'\{\overline{m}\langle \mathbf{0}\rangle/x\}$, where m is fresh;
  2. or $Q \xrightarrow{\tau} Q'$ and $P' \,\mathcal{R}\, Q' \parallel \overline{a}\langle \overline{m}\langle \mathbf{0}\rangle\rangle$.

We write $P \downarrow_{\overline{a}}$ (resp. $P \downarrow_a$) if P can perform an output (resp. input) transition at a.

Definition 5.1. Asynchronous barbed congruence, $\simeq$, *is the largest relation on closed processes that is symmetric, is a τ-bisimulation (Definition 4.1(1)), context-closed (i.e., $P \simeq Q$ implies $C[P] \simeq C[Q]$, for all closed contexts $C[\cdot]$), and barb preserving (i.e., if $P \simeq Q$ and $P \downarrow_{\overline{a}}$, then also $Q \downarrow_{\overline{a}}$).*

Lemma 5.2. *Asynchronous barbed congruence coincides with normal bisimilarity.*

Proof (Sketch). As we are comparing a synchronous relation against an asynchronous one, some details of the proof differ from proofs in the literature involving barbed congruences. Specifically, we have to show that two asynchronous barbed congruent processes P and Q satisfy the synchronous clause for input transitions as by Definition 4.2(2). To this end, for every message $\overline{a}\langle R\rangle$ that appears at top level in P or Q, we add a process $a(y).\,\overline{b}\langle y\rangle$ that renames this message to message $\overline{b}\langle R\rangle$, where b is some fresh channel. The resulting processes $P \parallel T$ and $Q \parallel T$ are still asynchronous barbed congruent, and as the channels b are fresh, any τ action generated by the renaming of one message is matched by a τ action from a similar renaming by the other process, yielding eventually two asynchronous barbed congruent processes P' and Q' that cannot immediately perform any τ-transitions. Therefore on these processes the τ-transitions that appear in the input clause of asynchronous bisimilarity (and which make the difference w.r.t. synchronous bisimilarity) are impossible. Thus we can show that, on such special asynchronous processes, any input from P' is matched by an input from Q' and conversely, as in synchronous bisimilarities. Finally, reverting the renaming we are able to derive an analogous match on inputs for P and Q. □

Remark 5.3. *The proof relies on the fact that HOcore has no operators of recursion, choice, and restriction. The higher-orderness of HOcore does not really play a role. The proof could indeed be adapted to CCS-like, or π-calculus-like, languages in which the same operators are missing.*

In *synchronous* barbed congruence, input barbs $P \downarrow_a$ are also observable (in the "barb preserving" condition).

Corollary 5.4. *In HOcore asynchronous and synchronous barbed congruence coincide, and they also coincide with all complete combinations of the HOcore bisimulation clauses of Theorem 4.13.*

Further, Corollary 5.4 can be extended to include the asynchronous versions of the labeled bisimilarities in Section 4 (precisely, the *complete asynchronous combinations* of the HOcore bisimulation clauses; that is, complete combinations that make use of an asynchronous input clause as outlined before Definition 5.1). This holds because: (i) all proofs of Section 4 can be easily adapted to the corresponding asynchronous labeled bisimilarities; (ii) using standard reasoning for barbed congruences, one can show that asynchronous normal bisimilarity coincides with asynchronous barbed congruence; (iii) via Corollary 5.4 one can then relate the asynchronous labeled bisimilarities to the synchronous ones.

6. Axiomatization and Complexity

We have shown in the previous section that in HOcore the main forms of bisimilarity for higher-order process calculi coincide. We therefore simply call *bisimilarity* such a relation, and indicate it as $\sim$. Here we present a sound and complete axiomatization of bisimilarity. We then exploit it to derive complexity bounds for bisimilarity checking.

The *size* of a process P, written $\#(P)$, is inductively defined as:

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

Lemma 6.1. *$P \sim Q$ implies $\#(P) = \#(Q)$.*

Following [11] we prove a result of unique prime decomposition of processes and a derived cancellation property.

Definition 6.2 (Prime decomposition)**.** *A process P is* prime *if $P \not\sim \mathbf{0}$ and $P \sim P_1 \parallel P_2$ imply $P_1 \sim \mathbf{0}$ or $P_2 \sim \mathbf{0}$. When $P \sim \prod_{i=1}^{n} P_i$ where each P_i is prime, we call $\prod_{i=1}^{n} P_i$ a* prime decomposition *of P.*

Proposition 6.3 (Unique decomposition)**.** *Any process P admits a prime decomposition $\prod_{i=1}^{n} P_i$ which is unique up to bisimilarity and permutation of indices (i.e., given two prime decompositions $\prod_{i=1}^{n} P_i$ and $\prod_{i=1}^{n} P'_i$ there is a permutation σ of $\{1,\dots,n\}$ such that $P_i \sim P'_{\sigma(i)}$ for each $i \in \{1,\dots,n\}$).*

Corollary 6.4 (Cancellation)**.** *For all P, Q, and R, if $P \parallel R \sim Q \parallel R$ then also $P \sim Q$.*

The key law for the axiomatization, and the following results, are inspired by similar results by Hirschkoff and Pous [4] for pure CCS. Using their terminology, we call *distribution law*, briefly (`DIS`), the axiom schema below ($\prod_1^k Q$ denotes the parallel composition of k copies of Q.)

$$a(x).\,(P \parallel \textstyle\prod_1^{k-1} a(x).\,P) \;=\; \prod_1^{k} a(x).\,P \qquad (\texttt{DIS})$$

We then call *extended structural congruence*, written $\equiv_{\mathtt{E}}$, the extension of the structural congruence relation ($\equiv$, Definition 2.1) with the axiom schema (`DIS`). Below we prove that $\equiv_{\mathtt{E}}$ gives us an algebraic characterization of $\sim$ in HOcore. Establishing the soundness of $\equiv_{\mathtt{E}}$ is easy; below we discuss completeness.

A process P is in *normal form* if it cannot be further simplified in the system $\equiv_{\mathtt{E}}$ by applications of law (`DIS`) from left to right. Formally, there are no processes P' and Q' such that $P \equiv P'$, and Q' is obtained from the rewriting of a subterm of P' using law (`DIS`) from left to right. Any process P has a normal form that is unique up to $\equiv$, and which will be denoted by $\mathsf{n}(P)$. Below A and B range over normal forms, and a process is said to be *non-trivial* if its size is different from 0.

Lemma 6.5. *If $a(x).\,P \sim Q \parallel Q'$ with $Q, Q' \not\sim \mathbf{0}$, then $a(x).\,P \sim \prod_1^k a(x).\,A$, where $k > 1$ and $a(x).\,A$ is in normal form.*

The proof of Lemma 6.5 exploits Proposition 6.3 and Corollary 6.4.

Lemma 6.6. *For A, B in normal form, if $A \sim B$ then $A \equiv B$.*

The proof of Lemma 6.6 is similar to the one in [4], with some extra cases. The theorem below follows.

Theorem 6.7. *For any processes P and Q, we have $P \sim Q$ iff $\mathsf{n}(P) \equiv \mathsf{n}(Q)$.*

Corollary 6.8. *$\equiv_{\mathtt{E}}$ is a sound and complete axiomatization of bisimilarity in HOcore.*

To analyze the complexity of deciding whether two processes are bisimilar, one could apply the technique from [3], and derive that bisimilarity is decidable in time which is linear in the size of the LTS for $\sim^{\circ}_{\mathtt{IO}}$ (which, e.g., avoids τ transitions). However the LTS is exponential in the size of the process. A more efficient solution exploits the axiomatization above: one can normalize processes and reduce bisimilarity to syntactic equivalence of normal forms.

For simplicity, we assume a process is represented as an ordered tree (but we will transform it into a DAG during normalization), with variables represented by De Bruijn indices. The tree has nodes for $\mathbf{0}$, variables in De Bruijn notation, input prefixes (with the continuation as only child), output (with the argument as only child), and parallel composition. In particular, each node for parallel composition has m leaves, where m is the number of parallel components (none of them containing top-level parallel compositions).

The first step of normalization is the following.

Normalization step 1. *Remove all $\mathbf{0}$ nodes that are children of a parallel composition node. Subsequently, if the parallel composition has 0 child, replace it with $\mathbf{0}$, if it has 1 child, replace it with the child.*

After this first step, the tree is traversed bottom-up, applying the following normalization steps.

Normalization step 2. *If the node is a parallel composition, sort all the children lexicographically. If n children are equal, leave just one and make n references to it.*

Normalization step 3. *If the node is an input prefix, apply* DIS *from left to right if possible.*

Lemma 6.9. *Let P, Q be processes and T_P, T_Q their tree representations normalized according to steps 1, 2 and 3. Then $P \sim Q$ iff $T_P = T_Q$.*

Theorem 6.10. *$P \sim Q$ can be decided in $O(n^2 \log m)$ where n is the maximum between the number of nodes in the tree representations of P and of Q, and m is the maximum branching factor (i.e., the number of parallel components) in them.*

Proof. The most expensive part of the bisimilarity check is normalization step 2. In particular, $O(n)$ nodes may require to apply normalization step 2, which requires time $O(n \log m)$ (considering that the cost of a comparison is linear in the minimum size of the compared subtrees). $\square$

7. Bisimilarity is undecidable with four static restrictions

If the restriction operator is added to HOcore, as in Plain CHOCS or Higher-Order π-calculus, then recursion can be encoded [23, 19] and most of the results in Sections 4-6 would break. In particular, higher-order and context bisimilarities are different and both undecidable [15, 17].

We discuss here the addition of a limited form of restriction, which we call *static restriction*. These restrictions may not appear inside output messages: in any output $\overline{a}\langle P\rangle$, P is restriction-free. This limitation is important: it prevents for instance the above-mentioned encoding of recursion from being written. Static restrictions could also be defined as top-level restrictions since, by means of standard structural congruence laws, any static restriction can be pulled out at the top-level. Thus the processes would take the form $\boldsymbol{\nu} a_1 \ldots \boldsymbol{\nu} a_n\ P$, where $\boldsymbol{\nu} a_i$ indicates the restriction on the name a_i, and where restriction cannot appear inside P itself. (The operational semantics—LTS and bisimilarities—are extended as expected; we omit them.)

We show that *four* static restrictions are enough to make undecidable any bisimilarity that has little more than a clause for τ-actions. For this, we reduce the Post correspondence problem (PCP) [14, 21] to such a problem. We call *complete τ-bisimilarity* any complete combination of the HOcore bisimulation clauses (as defined in Section 4) that includes the clause for τ actions (Definition 4.1(1)); the bisimilarity can even be asynchronous (Section 5).

Definition 7.1 (PCP). *An* instance *of the problem consists of an alphabet A containing at least two symbols, and a finite list $T_1, \ldots, T_n$ of tiles, where each tile is a pair of words over A. We use $T_i = (u_i, l_i)$ to denote a tile T_i with upper word u_i and lower word l_i. A solution to this instance is a non-empty sequence of indices $i_1, \ldots, i_k$, $1 \leq i_j \leq n$ ($j \in 1 \cdots k$), such that $u_{i_1} \cdots u_{i_k} = l_{i_1} \cdots l_{i_k}$. The decision problem is then to determine whether such a solution exists or not.*

Having (static) restrictions, we refine the encoding of non-nested replications (Definition 3.1) and define it in the unguarded case:

$$[\![!P]\!]_! = \boldsymbol{\nu} c\,(Q_c \parallel \overline{c}\langle Q_c\rangle)$$

where $Q_c = c(x).\,(x \parallel \overline{c}\langle x\rangle \parallel P)$ and P is a HOcore process (i.e., it is restriction-free).

Now, $[\![!\mathbf{0}]\!]_!$ is a purely divergent process, as it can only make τ-transitions, indefinitely; it is written using only one static restriction. Given an instance of PCP we build a set of processes $P_1, \ldots, P_n$, one for each tile $T_1, \ldots, T_n$, and show that, for each i, P_i is bisimilar to $[\![!\mathbf{0}]\!]_!$ iff the instance of PCP has no solution ending with T_i. Thus PCP is solvable iff there exists j such that P_j is not bisimilar to $[\![!\mathbf{0}]\!]_!$.

The processes $P_1, \ldots, P_n$ execute in two distinct phases: first they build a possible solution of PCP, then non-deterministically they stop building the solution and execute it. If the chosen composition is a solution then a signal on a free channel *success* is sent, thus performing a visible action, which breaks bisimilarity with $[\![!\mathbf{0}]\!]_!$.

The precise encoding of PCP into HOcore is shown in Table 2, and described below. We consider an alphabet of two letters, a_1 and a_2. The upper and lower words of a tile are treated as separate *strings*, which are encoded letter by letter. The encoding of a letter is then a process whose continuation encodes the rest of the string, and varies depending on whether the letter occurs in the upper or in the lower word. We use a single channel to encode both letters: for the upper word, a_1 is encoded as $\overline{a}.\,P$ and a_2 as $a.\,P$, where P is the continuation; for the lower word the encodings are switched. In Table 2, $[\![a_i, P]\!]_w$ denotes the encoding of the letter a_i with continuation P, with $w = u$ if the encoding is on the upper word, $w = l$ otherwise. Hence, given a string $s = a_i \cdot s'$, its encoding $[\![s, P]\!]_w$ is $[\![a_i, [\![s', P]\!]_w]\!]_w$, i.e., the first letter with the encoding of the rest as continuation. Notice that the encoding of an a_i in the upper word can synchronize only with the encoding of a_i for the lower word. This scheme requires output prefix, but it can be adapted to the asynchronous syntax of HOcore.

The whole system P_j is composed by a *creator* C_i for each tile T_i, a *starter* $S_{u,l}$ that launches the building of a tile composition ending with (u, l), and an *executor* E. The starter makes the computation begin; creators are called deterministically to add their tile to the beginning of the composition. The executor non-deterministically blocks the building of the composition and starts its execution. This proceeds if no difference is found: if both strings end at the same character, then synchronization on channel b can

Letters	$[\![a_1, P]\!]_u = [\![a_2, P]\!]_l = \overline{a}\langle P\rangle$
	$[\![a_2, P]\!]_u = [\![a_1, P]\!]_l = a(x).\,(x \parallel P)$
Strings	$[\![a_i \cdot s, P]\!]_w = [\![a_i, [\![s, P]\!]_w]\!]_w$
	$[\![\epsilon, P]\!]_w = P$ (ϵ is the empty word)
Creators	$C_i = up(x).\,low(y).$ $(\overline{up}\langle[\![u_i, x]\!]_u\rangle \parallel \overline{low}\langle[\![l_i, y]\!]_l\rangle)$
Starters	$S_{u,l} = \overline{up}\langle[\![u, \overline{b}]\!]_u\rangle \parallel \overline{low}\langle[\![l, b.\,\overline{success}]\!]_l\rangle$
Executor	$E = up(x).\,low(y).\,(x \parallel y)$
System	$P_j = \boldsymbol{\nu} up\, \boldsymbol{\nu} low\, \boldsymbol{\nu} a\, \boldsymbol{\nu} b\,(\, S_{u_j,l_j} \parallel !\prod_i C_i \parallel E)$

Table 2. Encoding of PCP

be performed, which in turn, makes action $\overline{success}$ visible. Notice that without synchronizing on b, action $\overline{success}$ could be visible even for two strings containing differences.

The encoding of replication requires another restriction, thus P_j has five restrictions. However, names low and a are used in different phases; thus choosing $low = a$ does not create interferences, and four restrictions are sufficient.

Theorem 7.2. *Given an instance of PCP and one of its tiles T_j, P_j is bisimilar to $[\![!\mathbf{0}]\!]_!$ according to any complete τ-bisimilarity iff there is no solution of the instance of PCP ending with T_j.*

Proof (Sketch). Note that all computations are infinite, and the only possible observable action is $\overline{success}$. One can show that for each $n \geq 1$ and each tile composition $(u_1 \cdots u_n, l_1 \cdots l_n)$ with $T_j = (u_n, l_n)$ there is a computation whose transitions have label τ leading to:

$$\begin{aligned} P'_{m'} \quad = \quad & \nu up, low, a, b.\ \ !\textstyle\prod_i C_i \parallel \prod_k C_{i_k} \parallel \\ & [\![u_1 \cdots u_n, \overline{b}]\!]_u \parallel [\![l_1 \cdots l_n, b.\,\overline{success}]\!]_l \end{aligned} \tag{1}$$

where i_k are values in $\{1, \ldots, n\}$ (not necessarily distinct). The proof is concluded by showing that each process of the form above has a computation containing an output at $success$ iff $u_1 \cdots u_n = l_1 \cdots l_n$. □

Corollary 7.3. *Barbed congruence and any complete τ-bisimilarity are undecidable in HOcore with four static restrictions.*

Theorem 7.2 actually shows that even *asynchronous barbed bisimilarity* (defined as the largest τ-bisimilarity that is output-barb preserving, and used in the definition of ordinary—as opposed to reduction-closed—barbed congruence) is undecidable. The corollary above then follows from the fact that all the relations there mentioned are at least as demanding as asynchronous barbed bisimilarity.

8. Other extensions

We now examine the impact on decidability of bisimilarity of some extensions of HOcore. We omit the details, including precise statements of the results.

Abstractions. An abstraction is a parametrized process of the form $(x)P$ that has a functional type $T \to T'$. Applying an abstraction $(x)P$ to an argument W yields the process $P\{W/x\}$. Since W can itself be an abstraction, the *order* of an abstraction —the level of arrow nesting in its type— can be arbitrarily high (even ω, if there are recursive types). Allowing the exchange of abstractions, as in the Higher-Order π-calculus, requires an application construct, as a destructor for abstractions. By setting bounds on the order of abstractions, one can define a hierarchy of subcalculi of the Higher-Order π-calculus [19]; and when this bound is ω, one obtains a calculus able to represent the π-calculus (all operators of the Higher-Order π-calculus are needed, including full restriction).

We have proved that extending HOcore with abstractions of order smaller than ω maintains the decidability of bisimilarity. Decidability then fails if the ω bound is removed, intuitively because in this case it is possible to simulate the λ-calculus.

Choice. Decidability remains with the addition of a choice operator to HOcore. The proofs require little modifications. The addition of both choice and output prefix is harder. It might be possible to extend the decidability proof for output prefix mentioned above so to accommodate also choice, but the details become much more complex.

Recursion. We do not know whether decidability is maintained by the addition of recursion (or similar operators such as replication).

9. Concluding Remarks

Process calculi are usually Turing complete and have an undecidable bisimilarity (and barbed congruence). Subcalculi have been studied where bisimilarity becomes decidable but then one loses Turing completeness. Examples are BPA and BPP (see, e.g., [9]) and CCS without restriction and relabeling [2]. In this paper we have identified a Turing complete formalism, HOcore, for which bisimilarity is decidable. We do not know other concurrency formalisms where the same happens. Other peculiarities of HOcore are: (1) it is higher-order, and contextual bisimilarities (barbed congruence) coincide with higher-order bisimilarity (as well as with others, such as context and normal bisimilarity); and (2) it is asynchronous (in that there is no continuation underneath an output), yet asynchronous and synchronous bisimilarities coincide. We do not know other non-trivial formalisms in which property (1) or (2) holds (of course (1) makes sense only on higher-order models).

We have also given an axiomatization for bisimilarity. From this we have derived polynomial upper bounds to the decidability of bisimilarity. The axiomatization also intuitively explains why results such as decidability, and the collapse of many forms of bisimilarity, are possible even though HOcore is Turing complete: the bisimilarity relation is very discriminating.

We have used encodings of Minsky machines and of the Post correspondence problem (PCP) for our undecidability results. The encodings are tailored to analyzing different problems: undecidability of termination, and undecidability of bisimilarity with static restrictions. The PCP encoding is always divergent, and therefore cannot be used to reason about termination. On the other hand, the encoding of Minsky machines would require at least one restriction for each instruction of the machine, and therefore would have given us a (much) worst result for static restrictions. We find both encodings interesting: they show different ways to exploit higher-order communications for modeling.

We have shown that bisimilarity becomes undecidable with the addition of four static restrictions. We do not know what happens with one, two, or three static restrictions. We also do not know whether the results presented would hold when one abstracts from τ-actions and moves to *weak* equivalences. The problem seems much harder; it reminds us of the situation for BPA and BPP, where strong bisimilarity is decidable but the decidability of weak bisimilarity is a long-standing open problem.

Acknowledgments. This research was initiated by some remarks and email exchange with Naoki Kobayashi. We also benefited from exchanges with Cinzia Di Giusto, Maurizio Gabbrielli, Antonín Kučera, and Gianluigi Zavattaro, and from feedback from the users of the Moca and Concurrency mailing lists. We are also grateful to the anonymous reviewers for their remarks and suggestions.

References

[1] Z. Cao. More on bisimulations for higher order *pi*-calculus. In *Proc. of FoSSaCS'06*, volume 3921 of *LNCS*, pages 63–78. Springer, 2006.

[2] S. Christensen, Y. Hirshfeld, and F. Moller. Decidable subsets of CCS. *Comput. J.*, 37(4):233–242, 1994.

[3] A. Dovier, C. Piazza, and A. Policriti. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci.*, 311(1-3):221–256, 2004.

[4] D. Hirschkoff and D. Pous. A distribution law for CCS and a new congruence result for the *pi*-calculus. In *Proc. of FoSSaCS'07*, volume 4423 of *LNCS*, pages 228–242. Springer, 2007.

[5] K. Honda and N. Yoshida. On reduction-based process semantics. *Theor. Comput. Sci.*, 151(2):437–486, 1995.

[6] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124(2):103–112, 1996.

[7] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theor. Comput. Sci.*, 148(2):281–301, 1995.

[8] A. Jeffrey and J. Rathke. Contextual equivalence for higher-order pi-calculus revisited. *Log. Meth. Comput. Sci.*, 1(1):1–22, 2005.

[9] A. Kučera and P. Jančar. Equivalence-checking on infinite-state systems: Techniques and results. *TPLP*, 6(3):227–264, 2006.

[10] I. Lanese, J. A. Pérez, D. Sangiorgi, and A. Schmitt. On the Expressiveness and Decidability of Higher-Order Process Calculi (Extended Version), 2008. `http://www.cs.unibo.it/~perez/hocore`.

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

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

[13] I. Phillips and M. G. Vigliotti. Symmetric electoral systems for ambient calculi. *Inf. Comput.*, 206(1):34–72, 2008.

[14] E. L. Post. A variant of a recursively unsolvable problem. *Bull. of the Am. Math. Soc*, 52:264–268, 1946.

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

[16] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Inf. Comput.*, 111(1):120–153, 1994.

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

[18] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. In *Proc. of LICS'07*, pages 293–302. IEEE Computer Society, 2007.

[19] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[20] P. Schnoebelen. Bisimulation and other undecidable equivalences for lossy channel systems. In *Proc. of TACS'01*, volume 2215 of *LNCS*, pages 385–399. Springer, 2001.

[21] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 2005.

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

[23] B. Thomsen. *Calculi for Higher Order Communicating Systems*. PhD thesis, Imperial College, 1990.

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

[25] X. Xu. *On the Bisimulation Theory and Axiomatization of Higher-order Process Calculi*. PhD thesis, Shanghai Jiao Tong University, 2007.