On the expressiveness and decidability of higher-order process calculi

Lanese, Ivan; Perez, Jorge A.; Sangiorgi, Davide; Schmitt, Alan

Published in:
Information and Computation

DOI:
10.1016/j.ic.2010.10.001

Information and Computation 209 (2011) 198–226

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

On the expressiveness and decidability of higher-order process calculi[☆,☆☆]

Ivan Lanese[a], Jorge A. Pérez[b], Davide Sangiorgi[a,*], Alan Schmitt[c]

[a] Focus Team, Università di Bologna/INRIA, Italy
[b] CITI – Department of Computer Science, FCT New University of Lisbon, Portugal
[c] Sardes Project-Team, INRIA Grenoble - Rhône-Alpes, France

ARTICLE INFO

Article history:
Received 19 February 2010
Revised 3 June 2010
Available online 16 October 2010

Keywords:
Process calculi
Higher-order communication
Bisimulation
Expressiveness

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 deterministic encoding of Minsky machines, the calculus is shown to be Turing complete. Therefore its termination problem is undecidable. Strong bisimilarity, however, is shown to be decidable. Furthermore, 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 1990s, with CHOCS [1] and Plain CHOCS [2], the higher-order π-calculus [3], 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: in most higher-order process calculi, labeled transition systems must deal with higher-order substitutions and scope extrusion, and ordinary definitions of bisimulation and behavioral equivalences become unsatisfactory as they are over-discriminating (this point is discussed later in more detail). 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 inspired by, and are formally closer to, 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 [4] and Xu [5] 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

☆ Extended and revised version of an homonymous paper that appeared in the Proceedings of LICS'08, IEEE Computer Society, 2008.
☆☆ Research partially supported by European projects FET-GC II IST-2005-16004 Sensoria and FP7-231620 HATS and by French ANR project "CHOCO".
* Corresponding author.
E-mail addresses: lanese@cs.unibo.it (I. Lanese), jorge.perez@di.fct.unl.pt (J.A. Pérez), davide.sangiorgi@cs.unibo.it (D. Sangiorgi), alan.schmitt@inria.fr (A. Schmitt).

0890-5401/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2010.10.001

of a variant of π-calculus into higher-order π-calculus [6] 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., [7,8]) 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 shed light on the expressiveness of higher-order process calculi, and related questions of decidability and of behavioral equivalence.

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 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 impossible. This makes the absence of recursion/replication also relevant, as known encodings of fixed-point combinators in higher-order process calculi exploit the restriction operator to avoid harmful interferences (notably for nested recursion).

Even though HOcore is minimal, it remains non-trivial: in Section 3, we show that it is Turing complete, therefore its termination problem is undecidable, by exhibiting a deterministic encoding of Minsky machines [9]. 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 the proof that bisimilarity is a congruence difficult. 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 labeled bisimilarity for higher-order processes have been put forward. In *higher-order bisimilarity* [3,4], 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 [3], and basic properties, such as congruence, may be very hard to establish. *Context bisimilarity* [3,10] avoids the separation between the argument and the continuation of an output action, this continuation being either syntactically present or consisting of other processes running in parallel. To this end, it explicitly takes into account the context in which the emitted process is supposed to go. Context bisimilarity yields more satisfactory process equalities and coincides with contextual equivalence (i.e., barbed congruence). A drawback of this approach is the universal quantification over contexts in the clause for output actions, which can hinder its use in practice to check equivalences. *Normal bisimilarity* [3,10,11] 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* [12], 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 the above forms (higher-order bisimilarity, context bisimilarity, normal bisimilarity, barbed congruence) coincide, and they also coincide with their asynchronous versions. Furthermore, we show that such a bisimilarity relation is decidable.

In the concurrency literature, there are examples of formalisms which are not Turing complete and where nevertheless (strong) bisimilarity is undecidable (e.g., Petri nets [13] or lossy channel systems [14]). We are not aware however of examples of the opposite situation: 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 [15] 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 [16] and Hirschkoff and Pous's axioms for a fragment of (finite) CCS [17].

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 $\boldsymbol{\nu} a_1, \ldots, \boldsymbol{\nu} a_n P$ where the inner process P is restriction-free. Via an encoding of the Post correspondence problem [18,19], 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), give some concluding remarks, and discuss related work (Section 9).

2. The calculus

We now introduce HOcore, the core of calculi for higher-order concurrency such as CHOCS [1], Plain CHOCS [2], and higher-order π-calculus [3,20,21]. 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

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

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 is defined as follows.

Definition 2.1. The *size* of a process P, written $\#(P)$, is inductively defined as:

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

We now describe the labeled transition system, which is defined on open processes. 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 P emits P' at a, and in doing so evolves to P''. We use α to denote a generic label of a transition. The notion of free variables extends to labels as expected: $\mathsf{fv}(\overline{a}\langle P\rangle) = \mathsf{fv}(P)$. For bound variables in labels, we have $\mathsf{bv}(a(x)) = \{x\}$ and $\mathsf{bv}(\overline{a}\langle P\rangle) = \emptyset$.

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

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

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

(We have omitted Act2 and Tau2, the symmetric counterparts of the last two rules.)

Definition 2.2. The *structural congruence* relation is the smallest congruence generated by the following laws:

$$P \parallel \mathbf{0} \equiv P,\quad P_1 \parallel P_2 \equiv P_2 \parallel P_1, \qquad P_1 \parallel (P_2 \parallel P_3) \equiv (P_1 \parallel P_2) \parallel P_3.$$

Reductions $P \longrightarrow P'$ are defined as $P \equiv\xrightarrow{\tau}\equiv P'$. We now state a few results which will be important later.

Lemma 2.3. *If $P \xrightarrow{\alpha} P'$ and $P \equiv Q$ then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv Q'$.*

Proof. By induction on the derivation of $P \equiv Q$, then by case analysis on $P \xrightarrow{\alpha} Q$. □

Definition 2.4. A variable x is *guarded in $P \in$ HOcore* (or simply *guarded*, when P is clear from the context) iff x only occurs free in an output or in subexpressions of P of the form $\pi.P'$, where π is any prefix. A process $P \in$ HOcore is *guarded* (or has *guarded variables*) iff all its free variables are guarded.

In particular, notice that if x is guarded in P, then x does not appear in evaluation contexts (i.e., contexts which allow transitions in the hole position), and if x is not free in P then it is guarded in P. For the next lemma, we recall that an output action from an open process may contain free variables, thus $\alpha\{\widetilde{R}/\widetilde{x}\}$ is the action obtained from α by applying the substitution $\{\widetilde{R}/\widetilde{x}\}$.

Lemma 2.5. *Suppose that $P \in$ HOcore and consider some variables $\widetilde{x}$. Then, for all $\widetilde{R} \in$ HOcore we have:*

1. *If $P \xrightarrow{\alpha} P'$, with free variables in $\widetilde{R}$ disjoint from the variables in P, α, and $\widetilde{x}$, then $P\{\widetilde{R}/\widetilde{x}\} \xrightarrow{\alpha\{\widetilde{R}/\widetilde{x}\}} P'\{\widetilde{R}/\widetilde{x}\}$;*
2. *if $P\{\widetilde{R}/\widetilde{x}\} \xrightarrow{\alpha'} M'$, with variables $\widetilde{x}$ guarded in P, and free variables in $\widetilde{R}$ disjoint from the variables in P and $\widetilde{x}$, then there are P' and α such that $P \xrightarrow{\alpha} P'$, $M' = P'\{\widetilde{R}/\widetilde{x}\}$, $\alpha' = \alpha\{\widetilde{R}/\widetilde{x}\}$, and the free variables in $\widetilde{R}$ are disjoint from the variables in α.*
3. *if $P\{\widetilde{m}/\widetilde{x}\} \xrightarrow{\alpha'} M'$ with $\widetilde{m}$ fresh in P and $\alpha' \neq m_i$ for any $m_i \in \widetilde{m}$, then $P \xrightarrow{\alpha} P'$, $M' = P'\{\widetilde{m}/\widetilde{x}\}$, and $\alpha' = \alpha\{\widetilde{m}/\widetilde{x}\}$.*

Proof. By induction on the transitions. □

Lemma 2.6. *For every $P \in$ HOcore and variable x, there are $P' \in$ HOcore with x guarded in P' and an integer $n \geq 0$ such that*

1. *$P \equiv P' \parallel \prod_1^n x$;*
2. *$P\{R/x\} \equiv P'\{R/x\} \parallel \prod_1^n R$, for every $R \in$ HOcore.*

Proof. By induction on the structure of processes. □

3. HOcore is Turing complete

We present in this section an encoding of Minsky machines [9] 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 encoding is deterministic, i.e., at any step the encoding of any Minsky machine has at most one reduction.

3.1. Minsky machines

A Minsky machine is a model composed of a set of sequential, labeled instructions, and two registers. Minsky machines have been shown to be a Turing complete model (see [9], Chapters 11 and 14), hence termination is undecidable for Minsky machines. 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: $\texttt{INC}(r_j)$, which adds 1 to register r_j and proceeds to the next instruction; $\texttt{DECJ}(r_j, k)$, which 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 Table 1.

3.2. Encoding Minsky machines into HOcore

The encoding of a Minsky machine into HOcore is denoted as $[\![\cdot]\!]_{\mathrm{M}}$. In order to simplify the presentation of the encoding, we introduce two useful notations that represent limited forms of guarded choice and guarded replication. Then we show how to count and test for zero in HOcore and present the main encoding, depicted in Table 2.

Guarded choice. We introduce here a notation for a simple form of guarded choice to choose between different behaviors. Assume, for instance, that a_i should trigger P_i, for $i \in \{1, 2\}$. This is written as $a_1.\ P_1 + a_2.\ P_2$, whereas the choice of the behavior P_i is written as $\widehat{a_i}$.

Table 1
Reduction of Minsky machines.

$$\text{M-INC}\ \frac{i : \mathtt{INC}(r_j) \quad m'_j = m_j + 1 \quad m'_{1-j} = m_{1-j}}{(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (i+1, m'_0, m'_1)}$$

$$\text{M-DEC}\ \frac{i : \mathtt{DECJ}(r_j, k) \quad m_j \neq 0 \quad m'_j = m_j - 1 \quad m'_{1-j} = m_{1-j}}{(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (i+1, m'_0, m'_1)}$$

$$\text{M-JMP}\ \frac{i : \mathtt{DECJ}(r_j, k) \quad m_j = 0}{(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (k, m_0, m_1)}$$

Table 2
Encoding of Minsky machines.

INSTRUCTIONS $(i : I_i)$

$$[\![(i : \mathtt{INC}(r_j))]\!]_{\mathsf{M}} \;=\; !p_i.\,\big(\widehat{inc_j} \parallel ack.\,\overline{p_{i+1}}\big)$$

$$[\![(i : \mathtt{DECJ}(r_j, k))]\!]_{\mathsf{M}} \;=\; !p_i.\,\big(\widehat{dec_j} \parallel ack.\,(z_j.\,\overline{p_k} + n_j.\,\overline{p_{i+1}})\big)$$

REGISTERS r_j

$$[\![r_j = 0]\!]_{\mathsf{M}} \;=\; \Big(inc_j.\,\overline{r_j^{\mathsf{S}}}\langle (\!|\, 0 \,|\!)_j \rangle \;+\; dec_j.\,\big(\overline{r_j^0} \parallel \widehat{z_j}\big)\Big) \parallel \mathsf{REG}_j$$

$$[\![r_j = m]\!]_{\mathsf{M}} \;=\; \Big(inc_j.\,\overline{r_j^{\mathsf{S}}}\langle (\!|\, m \,|\!)_j \rangle \;+\; dec_j.\,(\!|\, m-1 \,|\!)_j\Big) \parallel \mathsf{REG}_j$$

where:

$$\mathsf{REG}_j \;=\; !r_j^0.\,\Big(\overline{ack} \parallel inc_j.\,\overline{r_j^{\mathsf{S}}}\langle (\!|\, 0 \,|\!)_j \rangle \;+\; dec_j.\,\big(\overline{r_j^0} \parallel \widehat{z_j}\big)\Big) \parallel$$
$$!r_j^{\mathsf{S}}(y).\,\Big(\overline{ack} \parallel inc_j.\,\overline{r_j^{\mathsf{S}}}\langle \overline{r_j^{\mathsf{S}}}\langle y \rangle \parallel \widehat{n_j} \rangle \;+\; dec_j.\,y\Big)$$

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

The notation can be seen as a shortcut for HOcore terms as follows.

Definition 3.1. Let $\sigma = \{(a_1, a_2) \mid a_1 \neq a_2\}$ be a fixed set of pairs of distinct names. The notation for *guarded choice* can be defined as follows:

$$a_1.\; P_1 + a_2.\; P_2 \triangleq \overline{a_1}\langle P_1 \rangle \parallel \overline{a_2}\langle P_2 \rangle$$
$$\widehat{a_1} \triangleq a_2(x_2).\, a_1(x_1).\, x_1$$
$$\widehat{a_2} \triangleq a_1(x_1).\, a_2(x_2).\, x_2$$

where, in all cases, $(a_1, a_2) \in \sigma$.

We consider only *binary* guarded choice as it is sufficient to encode Minsky machines. This way, given a pair $(a_1, a_2) \in \sigma$ and the process $a_1.\; P_1 + a_2.\; P_2$, the trigger $\widehat{a_i}$ (with $i \in \{1, 2\}$) consumes both P_i's and spawns the one chosen, i.e., $(a_1.\; P_1 + a_2.\; P_2) \parallel \widehat{a_i} \xrightarrow{\tau}\xrightarrow{\tau} P_i$. This notation has the expected behavior as long as 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, and as long as concurrently running guarded choices use distinct names.

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

Definition 3.2. Assume a fresh name c. The definition of *input-guarded replication* in HOcore is:

$$!a(z).\, P \triangleq (a(z).\, c(x).\, x \parallel \overline{c}\langle x \rangle \parallel P) \parallel \overline{c}\langle a(z).\, c(x).\, x \parallel \overline{c}\langle x \rangle \parallel P \rangle$$

where P contains no replications (nested replications are forbidden).

After having been activated by an output message, replication requires an additional τ step to enable the continuation P, i.e., $!a(z).\, P \xrightarrow{a(z)}\xrightarrow{\tau} (!a(z).\, P) \parallel P$.

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 $k+1$ stored 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 $\overline{r_j^0}$. This way, for instance, $(\!|\ 2\ |\!)_j$ is $\overline{r_j^{\mathsf{S}}}\langle \overline{r_j^{\mathsf{S}}}\langle \overline{r_j^0} \parallel \widehat{n_j}\rangle \parallel \widehat{n_j}\rangle \parallel \widehat{n_j}$.

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, as specified in the reduction relation in Table 1. 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. 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 of 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 3.3. 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 $r_j^0, r_j^{\mathsf{S}}, p_1, \ldots, p_n, inc_j, dec_j, ack$ (for $j \in \{0, 1\}$). Given the encodings in Table 2, 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 $\longrightarrow_{\mathsf{M}}^*$ is the reflexive and transitive closure of $\longrightarrow_{\mathsf{M}}$, and $N \Uparrow_{\mathsf{M}}$ means that N has an infinite sequence of reductions.

Lemma 3.4. *Let N be a Minsky machine. We have*:

1. $N \longrightarrow_{\mathsf{M}}^* N'$ iff $[\![N]\!]_{\mathsf{M}} \longrightarrow^* [\![N']\!]_{\mathsf{M}}$;
2. if $[\![N]\!]_{\mathsf{M}} \longrightarrow^* P_1, P_1 \longrightarrow P_2$, *and* $P_1 \longrightarrow P_3$ *then* $P_2 \equiv P_3$;
3. if $[\![N]\!]_{\mathsf{M}} \longrightarrow^* P_1$ *then there exists* N' *such that* $P_1 \longrightarrow^* [\![N']\!]_{\mathsf{M}}$ *and* $N \longrightarrow_{\mathsf{M}}^* N'$;
4. $N \Uparrow_{\mathsf{M}}$ iff $[\![N]\!]_{\mathsf{M}} \Uparrow$.

Proof. The proof of the lemma, which is detailed in Appendix A, relies on two properties. The first one ensures that for every computation of the Minsky machine the encoding can perform a finite, non-empty sequence of reductions that correspond to the one made by the machine. The second property ensures that if the process encoding a Minsky machine has a reduction to P' then (i) the machine also has a reduction to N', and (ii) P' has a finite deterministic sequence of reductions to the encoding of N'. $\square$

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

Corollary 3.5. *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 are not aware of other results on process calculi where processes can perform τ-transitions and yet a bisimulation that does not mention τ-transitions is discriminating enough. (One of the reasons that make this possible is that bisimulation in HOcore is very discriminating.)

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' such that $Q \xrightarrow{\tau} Q'$ 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'' such that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$ 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'' such that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$ 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' such that $Q \xrightarrow{a(x)} Q'$ and $P' \mathcal{R} Q'$,
   - $P \equiv x \parallel P'$ implies that there is Q' such that $Q \equiv x \parallel Q'$ 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'' such that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$ 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' such that $Q \xrightarrow{a(x)} Q'$ and $P'\{^{\overline{m}}\!/x\} \mathcal{R}\, Q'\{^{\overline{m}}\!/x\}$, where m is fresh;
3. *closed* if $P \mathcal{R} Q$ and $P \xrightarrow{a(x)} P'$ imply that there is Q' such that $Q \xrightarrow{a(x)} Q'$ 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 needs 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 $\sim_{HO}$, is the largest relation on closed HOcore processes that is a τ-bisimulation, a higher-order output bisimulation, and is closed.

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

Normal bisimilarity, written $\sim_{NOR}$, is the largest relation on closed HOcore processes that is a τ-bisimulation, an output normal bisimulation, and an input normal bisimulation.

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

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

Environmental bisimilarity [22], 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 [3] that in calculi of pure higher-order concurrency early and late clauses are equivalent.

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

Remark 4.5. In contrast with normal bisimulation (as defined in, e.g., [3,10]), our clause for output normal bisimulation does not use a replication in front of the introduced fresh name. Such a replication would be needed in extensions of the calculus (e.g., with recursion or restriction).

A bisimilarity on closed processes is extended to open processes as follows.

Definition 4.6 (*Extension of bisimilarities*). Let $\mathcal{R}$ be a bisimilarity on closed HOcore processes. The extension of $\mathcal{R}$ to open HOcore processes, denoted $\mathcal{R}^\star$, is defined by

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

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

The simplest complete form of bisimilarity is $\sim^o_{IO}$. Not only $\sim^o_{IO}$ is the less demanding for proofs; it also has a straightforward proof of congruence. This is significant because congruence is usually a hard problem in bisimilarities for higher-order calculi. Before describing the proof of congruence for $\sim^o_{IO}$, we first define an auxiliary up-to technique that will be useful later.

Definition 4.7. A symmetric relation $\mathcal{R}$ on HOcore is an *open IO bisimulation up-to* $\equiv$ if $P \;\mathcal{R}\; Q$ implies:

1. if $P \xrightarrow{a(x)} P'$ then $Q \xrightarrow{a(x)} Q'$ and $P' \equiv\mathcal{R}\equiv Q'$;
2. if $P \xrightarrow{\overline{a}\langle P''\rangle} P'$ then $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$ with $P' \equiv\mathcal{R}\equiv Q'$ and $P'' \equiv\mathcal{R}\equiv Q''$;
3. if $P \equiv x \parallel P'$ then $Q \equiv x \parallel Q'$ and $P' \equiv\mathcal{R}\equiv Q'$.

Lemma 4.8. *If $\mathcal{R}$ is an open IO bisimulation up-to $\equiv$ and $(P, Q) \in \mathcal{R}$ then $P \sim^o_{IO} Q$.*

Proof. The proof proceeds by a standard diagram-chasing argument (as in, e.g., [23]): using Lemma 2.3 one shows that $\equiv\mathcal{R}\equiv$ is a $\sim^o_{IO}$-bisimulation. □

We now give the congruence result for $\sim^o_{IO}$.

Lemma 4.9 (Congruence of $\sim^o_{IO}$). *Let P_1, P_2 be open HOcore processes. $P_1 \sim^o_{IO} P_2$ implies:*

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

Proof. Items (1) and (3) are straightforward by showing the appropriate $\sim^o_{IO}$-bisimulations. We consider only (2). We show that, for every R, P_1, and P_2

$$\mathcal{S} = \{(P_1 \parallel R,\ P_2 \parallel R) \ :\ P_1 \sim^o_{IO} P_2\}$$

is a $\sim^o_{IO}$-bisimulation. We first suppose $P_1 \parallel R \xrightarrow{\alpha} P'$; we need to find a matching action from $P_2 \parallel R$. We proceed by case analysis on the rule used to infer α. There are two cases. In the first one $P_1 \xrightarrow{\alpha} P_1'$ and $P' = P_1' \parallel R$ is inferred using rule Act1 (by α-conversion we can ensure that R respects the side condition of the rule). By definition of $\sim^o_{IO}$-bisimulation, $P_2 \xrightarrow{\alpha} P_2'$ with $P_1' \sim^o_{IO} P_2'$. Using rule Act1 we infer that also $P_2 \parallel R \xrightarrow{\alpha} P_2' \parallel R$. We conclude that $(P_1' \parallel R, P_2' \parallel R) \in \mathcal{S}$. The second case follows by an analogous argument and occurs when $R \xrightarrow{\alpha} R'$ so that $P' = P_1 \parallel R'$ by rule Act2.

The last thing to consider is when $P_1 \parallel R \equiv x \parallel P'$; we need to show that $P_2 \parallel R \equiv x \parallel Q'$ and that $(P', Q') \in \mathcal{S}$. We distinguish two cases, depending on the shape of P'. First, assume that $P' \equiv P_1' \parallel R$, that is, x is a subprocess of P_1. Since $P_1 \sim^o_{IO} P_2$, then it must be that $P_2 \equiv x \parallel P_2'$ for some P_2', with $P_1' \sim^o_{IO} P_2'$. Taking $Q' \equiv P_2' \parallel R$ we thus have $(P', Q') \in \mathcal{S}$. Second, assume x is a subprocess of R, and we have $P' \equiv P_1 \parallel R'$, we then take $Q' \equiv P_2 \parallel R'$. Since $\mathcal{S}$ is defined over every R, then $(P', Q') \in \mathcal{S}$. □

Lemma 4.10 ($\sim^o_{IO}$ is preserved by substitutions). *If $P \sim^o_{IO} Q$ then for all x and R, also $P\{R/x\} \sim^o_{IO} Q\{R/x\}$.*

Proof. We first show the property for processes P, Q in which x is guarded, namely

$$\mathcal{R} = \{(P\{R/x\} \parallel L,\ Q\{R/x\} \parallel L) \ :\ P \sim^o_{IO} Q\}$$

is a $\sim^o_{IO}$-bisimulation up-to $\equiv$ (Definition 4.7). Take a pair $(P\{R/x\} \parallel L,\ Q\{R/x\} \parallel L) \in \mathcal{R}$. We shall concentrate on the possible moves from $P\{R/x\}$, say $P\{R/x\} \xrightarrow{\alpha} P'$; transitions from L, if any, can be handled analogously. We proceed by case analysis on the rule used to infer α.

We only detail the case in which α is an input action $a(y)$ inferred using rule Inp; the case in which α is an output is similar (there may be a substitution on the label), and the case where α is $a(x)$ is simpler (the substitution does not occur under the prefix). Since x is guarded in P, using Lemma 2.5(2), there is P_1 such that $P \xrightarrow{a(y)} P_1$ and $P' = P_1\{R/x\}$. By definition of $\sim^o_{IO}$-bisimulation, also $Q \xrightarrow{a(y)} Q_1$ with $P_1 \sim^o_{IO} Q_1$. Hence, by Lemma 2.5(1), $Q\{R/x\} \xrightarrow{a(y)} Q_1\{R/x\}$. It remains to show that $P_1\{R/x\}$ and $Q_1\{R/x\}$ can be rewritten into the form required in the bisimulation. Using Lemma 2.6(1), we have

$$P_1 \equiv P_1' \parallel \prod^n x \quad \text{and} \quad Q_1 \equiv Q_1' \parallel \prod^m x$$

for P_1', Q_1' in which x is guarded. As $P_1 \sim^o_{IO} Q_1$, it must be $n = m$ and $P_1' \sim^o_{IO} Q_1'$. Finally, using Lemmas 2.6(2) and 4.9 we have

$$P_1\{R/x\} \equiv P_1'\{R/x\} \parallel \prod^n R \quad \text{and} \quad Q_1\{R/x\} \equiv Q_1'\{R/x\} \parallel \prod^n R$$

which closes up the bisimulation up-to $\equiv$. We then conclude by Lemma 4.8.

For processes where x is not guarded, we proceed exactly as above: first we rewrite P into $P' \parallel \prod^n x$ and Q into $Q' \parallel \prod^m x$ using Lemma 2.6(1), then we show that $m = n$ and $P' \sim^o_{IO} Q'$ by definition of $\sim^o_{IO}$, and we conclude using the previous result for guarded processes and Lemmas 2.6(2) and 4.9. □

The most striking property of $\sim^o_{IO}$ is its decidability.

Lemma 4.11. *Relation $\sim^o_{IO}$ is decidable.*

Proof. We have to check whether $P \sim^o_{IO} Q$. We show that this is decidable by induction on $\#(P)$. The base case $\#(P) = 0$ is trivial since in this case P has no transitions and no free variables. Let us consider the inductive case. We have one check to perform for each possible output transition, input transition, and unguarded variable (thus, a finite number of checks): $P \sim^o_{IO} Q$ iff all the checks succeed. We show that the checks are decidable:

- if $P \xrightarrow{\overline{a}\langle P''\rangle} P'$ then we have to verify that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$ with $P' \sim^o_{IO} Q'$ and $P'' \sim^o_{IO} Q''$ (otherwise the check fails). We have $P \equiv \overline{a}\langle P''\rangle \parallel P'$, thus $\#(P') < \#(P)$ and $\#(P'') < \#(P)$ and we can decide $P' \sim^o_{IO} Q'$ and $P'' \sim^o_{IO} Q''$ by inductive hypothesis. Thus the check is decidable.
- if $P \xrightarrow{a(x)} P'$ then we have to verify that $Q \xrightarrow{a(x)} Q'$ with $P' \sim^o_{IO} Q'$. We have $P \equiv a(x).R \parallel R'$ and $P' \equiv R \parallel R'$, thus $\#(P') < \#(P)$ and we can decide $P' \sim^o_{IO} Q'$ by inductive hypothesis.
- if $P \equiv x \parallel P'$ then we have to verify that $Q \equiv x \parallel Q'$ with $P' \sim^o_{IO} Q'$. Since $\#(P') < \#(P)$ we can decide $P' \sim^o_{IO} Q'$ by inductive hypothesis. □

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

Lemma 4.12. *Relation $\sim^o_{IO}$ is a τ-bisimulation.*

Proof. Suppose $(P, Q) \in \sim^o_{IO}$ and $P \xrightarrow{\tau} P'$; we have to find a matching transition $Q \xrightarrow{\tau} Q'$. We proceed by induction on the derivation $P \xrightarrow{\tau} P'$. The cases Act1 and Act2 are immediate by induction. The remaining cases are using either rule Tau1 or Tau2. We consider only the first one as the second is symmetric. If rule Tau1 was used, then we can decompose P's transition into an output $P \xrightarrow{\overline{a}\langle R\rangle} P_1$ followed by an input $P_1 \xrightarrow{a(x)} P_2 \parallel P_3$, with $P' = P_2\{R/x\} \parallel P_3$ (that is, the structure of P is $P \equiv \overline{a}\langle R\rangle \parallel a(x).P_2 \parallel P_3$). By definition of $\sim^o_{IO}$, Q is capable of matching these two transitions, and the final derivative is a process Q_2 with $Q_2 \sim^o_{IO} P_2 \parallel P_3$. 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. Finally, since $\sim^o_{IO}$ is preserved by substitutions (Lemma 4.10), we can use rule Tau1 to derive a process $Q' = Q_2\{R/x\} \parallel Q_3$ that matches the τ-transition from P, with $(P', Q') \in \sim^o_{IO}$. □

Next lemmas prove useful properties of higher-order bisimulations.

Lemma 4.13. *Let P, Q be two processes and $\{x_1, \ldots, x_n\}$ be their free variables. We have $P \sim^\star_{HO} Q$ iff $P\{\widetilde{R_i}/\widetilde{x_i}\} \sim_{HO} Q\{\widetilde{R_i}/\widetilde{x_i}\}$ for any tuple of closed processes $\widetilde{R_i}$.*

Proof. First assume that $P \sim^{\star}_{HO} Q$, and let a be a fresh name. By definition of $\sim^{\star}_{HO}$, we thus have $a(x_1). \cdots . a(x_n). P \sim_{HO} a(x_1). \cdots . a(x_n). Q$. Let $R_1, \ldots, R_n$ be closed processes. By a repeated application of the input clause for $\sim_{HO}$, we conclude that $P\{\widetilde{R_i}/\widetilde{x_i}\} \sim_{HO} Q\{\widetilde{R_i}/\widetilde{x_i}\}$.

For the other direction, by definition we have to prove $a(x_1). \cdots . a(x_n). P \sim_{HO} a(x_1). \cdots . a(x_n). Q$. The only possible transitions of $a(x_1). \cdots . a(x_n). P$ are the inputs, which lead to $P\{\widetilde{R_i}/\widetilde{x_i}\}$ for some $\widetilde{R_i}$, and are matched by corresponding inputs from $a(x_1). \cdots . a(x_n). Q$ leading to $Q\{\widetilde{R_i}/\widetilde{x_i}\}$. These last are bisimilar for any choice of the $\widetilde{R_i}$ by hypothesis, thus the thesis follows. □

Lemma 4.14. *$\sim^{o}_{IO}$ and $\sim^{\star}_{HO}$ coincide.*

Proof. To show that $\sim^{\star}_{HO}$ is a IO bisimulation we show a slightly more general result: the relation $\{(P, Q) \mid P\{\widetilde{m_i}/\widetilde{x_i}\} \sim^{\star}_{HO} Q\{\widetilde{m_i}/\widetilde{x_i}\}, \widetilde{m_i}$ fresh in relation to $P, Q\}$ is a IO bisimulation. Note that we still use the open extension $\sim^{\star}_{HO}$, and do not require every variable of P and Q to be substituted by fresh names.

Let P, Q be two processes such that $P\{\widetilde{m_i}/\widetilde{x_i}\} \sim^{\star}_{HO} Q\{\widetilde{m_i}/\widetilde{x_i}\}$. If either P or Q has an unguarded variable x, we let m stand for a fresh variable and consider $P\{\widetilde{m_i}/\widetilde{x_i}\}\{m/x\}$ and $Q\{\widetilde{m_i}/\widetilde{x_i}\}\{m/x\}$. We easily show that $P\{\widetilde{m_i}/\widetilde{x_i}\}\{m/x\} \sim^{\star}_{HO} Q\{\widetilde{m_i}/\widetilde{x_i}\}\{m/x\}$. To keep notations short, we still write $P\{\widetilde{m_i}/\widetilde{x_i}\}$ and $Q\{\widetilde{m_i}/\widetilde{x_i}\}$ for the saturated processes with no unguarded variable.

We proceed by case on $P \xrightarrow{\alpha} P'$ (we also have an additional case for unguarded variables). Let $\{y_1, \ldots, y_n\}$ be the free guarded variables of $P\{\widetilde{m_i}/\widetilde{x_i}\}$ and $Q\{\widetilde{m_i}/\widetilde{x_i}\}$, and let $R_1, \ldots, R_n$ be closed processes. By Lemma 4.13, $P\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\} \sim_{HO} Q\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}$. As the R_j are closed, by Lemma 2.5(1), we can deduce that $P\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\} \xrightarrow{\alpha\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}} P'\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}$. By definition of $\sim_{HO}$, we have $Q\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\} \xrightarrow{\alpha\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}} Q'\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}$. By Lemma 2.5(2) we have $Q\{\widetilde{m_i}/\widetilde{x_i}\} \xrightarrow{\alpha\{\widetilde{m_i}/\widetilde{x_i}\}} Q'\{\widetilde{m_i}/\widetilde{x_i}\}$ as the y_j are guarded. Finally, by Lemma 2.5(3), we have $Q \xrightarrow{\alpha} Q'$, as the m_i are fresh thus $\alpha\{\widetilde{m_i}/\widetilde{x_i}\} = m_i$ is impossible. In the output case, by $\sim_{HO}$, we have directly $P'\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\} \sim_{HO} Q'\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}$ and we conclude by Lemma 4.13 that $P'\{\widetilde{m_i}/\widetilde{x_i}\} \sim^{\star}_{HO} Q'\{\widetilde{m_i}/\widetilde{x_i}\}$ since this is true for any $\widetilde{R_j}$, thus P' and Q' are still in the relation. We apply the same approach to the contents S_P and S_Q of the messages: $S_P\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\} \sim_{HO} S_Q\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}$ for arbitrary $\widetilde{R_j}$, thus $S_P\{\widetilde{m_i}/\widetilde{x_i}\} \sim^{\star}_{HO} S_Q\{\widetilde{m_i}/\widetilde{x_i}\}$, thus S_P and S_Q are in the relation. In the case of an input $a(x)$, P' and Q' may have an additional free variable x. In this case, by definition of $\sim_{HO}$ we have $P'\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}\{R/x\} \sim_{HO} Q'\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}\{R/x\}$ for any closed R, and we also conclude by Lemma 4.13 that $P'\{\widetilde{m_i}/\widetilde{x_i}\} \sim^{\star}_{HO} Q'\{\widetilde{m_i}/\widetilde{x_i}\}$, thus P' and Q' are still in the relation.

The final case is for when $P \equiv x \parallel P'$. By hypothesis, we have $P\{\widetilde{m_i}/\widetilde{x_i}\} \sim^{\star}_{HO} Q\{\widetilde{m_i}/\widetilde{x_i}\}$ and we know that $x = x_i$ for some i. We thus have $P\{\widetilde{m_i}/\widetilde{x_i}\} = m_i \parallel P'\{\widetilde{m_i}/\widetilde{x_i}\} \sim^{\star}_{HO} Q\{\widetilde{m_i}/\widetilde{x_i}\}$. Let $\{y_1, \ldots, y_n\}$ be the free guarded variables of $P\{\widetilde{m_i}/\widetilde{x_i}\}$ and $Q\{\widetilde{m_i}/\widetilde{x_i}\}$, and let $R_1, \ldots, R_n$ be closed processes. By Lemma 4.13, we have $P\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\} \sim_{HO} Q\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}$. As the R_j are closed, by Lemma 2.5(1), we have $P\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\} \xrightarrow{m_i} P'\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}$. We have $Q\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\} \xrightarrow{m_i} Q'\{\widetilde{m_i}/\widetilde{x_i}\}\{\widetilde{R_j}/\widetilde{y_j}\}$ by definition of $\sim_{HO}$, and by Lemma 2.5(2) we have $Q\{\widetilde{m_i}/\widetilde{x_i}\} \xrightarrow{m_i} Q'\{\widetilde{m_i}/\widetilde{x_i}\}$. As m_i is fresh in relation to P and Q, it means that x is free in Q, thus that $Q \equiv x \parallel Q'$. Moreover, as we showed this for arbitrary R_j, by Lemma 4.13, we have $P'\{\widetilde{m_i}/\widetilde{x_i}\} \sim^{\star}_{HO} Q'\{\widetilde{m_i}/\widetilde{x_i}\}$. We thus conclude by noting that we have (P', Q') in the relation.

To conclude, we remark that if $P \sim^{\star}_{HO} Q$ then (P, Q) is in the relation defined above (with no variable substituted), thus $P \sim^{o}_{IO} Q$.

We next show that $\sim^{o}_{IO}$ is a $\sim^{\star}_{HO}$-bisimulation. Suppose $(P_1, Q_1) \in \sim^{o}_{IO}$, and let a be a fresh name, $\mathrm{fv}(P_1) \cup \mathrm{fv}(Q_1) = \{x_1, \ldots, x_n\}$. We consider the pair of closed processes $P = a(x_1). \cdots . a(x_n). P_1$ and $Q = a(x_1). \cdots . a(x_n). Q_1$. By Lemma 4.9, we still have $P \sim^{o}_{IO} Q$. We proceed by case on the transition $P \xrightarrow{\alpha} P'$. The case $\alpha = \overline{b}\langle R\rangle$ is immediate, as it means that $n = 0$ (P_1 and Q_1 were already closed), and we conclude by $Q = Q_1 \xrightarrow{\overline{b}\langle S\rangle} Q'$ with $P' \sim^{o}_{IO} Q'$ and $R \sim^{o}_{IO} S$. Similarly, in the case of a τ transition, we also have $P = P_1$ and $Q = Q_1$ and we conclude by Lemma 4.12. We now turn to the input case, i.e., $P \xrightarrow{b(x)} P'$. By $\sim^{o}_{IO}$, we have $Q \xrightarrow{b(x)} Q'$ and $P' \sim^{o}_{IO} Q'$. We conclude by Lemma 4.10 that $P'\{R/x\} \sim^{o}_{IO} Q'\{R/x\}$ for any closed process R. □

We now move to the relationship between $\sim_{HO}$, $\sim^{o}_{NOR}$, and $\sim_{CON}$. We begin by establishing a few properties of normal bisimulation.

Lemma 4.15. *If $m. P_1 \parallel P \sim^{o}_{NOR} m. Q_1 \parallel Q$, for some fresh m, then we have $P_1 \sim^{o}_{NOR} Q_1$ and $P \sim^{o}_{NOR} Q$.*

Proof. We show that for any countable set of fresh names $m_1, \ldots$, the relation

$$\mathcal{S} = \bigcup_{j=1}^{\infty} \left\{ (P,\ Q)\ :\ P \parallel \prod_{k \in 1,\ldots,j} m_k. P_k \sim^{o}_{NOR} Q \parallel \prod_{k \in 1,\ldots,j} m_k. Q_k \right\}$$

is a $\sim^{o}_{NOR}$-bisimulation.

Suppose $(P, Q) \in \mathcal{S}$ and that $P \xrightarrow{\alpha} P'$; we need to show a matching action from Q. We have different cases depending on the shape of α. We consider only the case $\alpha = \overline{a}\langle P''\rangle$, the others being simpler. By Act1 we have

$$P \parallel \prod_{k\in 1,\dots,j} m_k.\,P_k \xrightarrow{\overline{a}\langle P''\rangle} P' \parallel \prod_{k\in 1,\dots,j} m_k.\,P_k.$$

Since $P \parallel \prod_{k\in 1,\dots,j} m_k.\,P_k \sim^{o}_{NOR} Q \parallel \prod_{k\in 1,\dots,j} m_k.\,Q_k$, there should exist a Q^* such that

$$Q \parallel \prod_{k\in 1,\dots,j} m_k.\,Q_k \xrightarrow{\overline{a}\langle Q''\rangle} Q^*\,,$$

with $P' \parallel \prod_{k\in 1,\dots,j} m_k.\,P_k \parallel m''.\,P'' \sim^{o}_{NOR} Q^* \parallel m''.\,Q''$. As $m_1, \dots, m_j$ are fresh, they do not occur in P, thus $\overline{a}\langle P''\rangle$ does not mention them. For the same reason, there cannot be any communication between $\prod_{k\in 1,\dots,j} m_k.\,Q_k$ and Q; so, we infer that the only possible transition is

$$Q \parallel \prod_{k\in 1,\dots,j} m_k.\,Q_k \xrightarrow{\overline{a}\langle Q''\rangle} Q^* = Q' \parallel \prod_{k\in 1,\dots,j} m_k.\,Q_k,$$

applying rule Act1 to $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$. Since $P' \parallel \prod_{k\in 1,\dots,j} m_k.\,P_k \parallel m''.\,P'' \sim^{o}_{NOR} Q' \parallel \prod_{k\in 1,\dots,j} m_k.\,Q_k \parallel m''.\,Q''$, we have $(P', Q') \in \mathcal{S}$ as needed.

We now show that if $P \parallel \prod_{k\in 1,\dots,j} m_k.\,P_k \sim^{o}_{NOR} Q \parallel \prod_{k\in 1,\dots,j} m_k.\,Q_k$ then $P_1 \sim^{o}_{NOR} Q_1$. To this end, we first detail a procedure for *consuming* $\sim^{o}_{NOR}$-bisimilar processes.

Given a process P, let $o(P)$ denote the number of output actions in P. Let $m(P) = \#(P) + o(P)$ be the measure that considers both the (syntactical) size of P and the number of output actions in it. Consider now two $\sim^{o}_{NOR}$-bisimilar processes P and Q. The procedure consists in consuming one of them by performing its actions completely; the other process can match these actions (as it is $\sim^{o}_{NOR}$-bisimilar) and will be consumed as well. We will show that $m(P)$ decreases at each step of the bisimulation game; at the end, we will obtain processes P_n and Q_n with $m(P_n) = m(Q_n) = 0$.

To illustrate the procedure, suppose, w.l.o.g., process P has the following shape:

$$P \equiv \prod_{h\in 1,\dots,t} x_h \parallel \prod_{i\in 1,\dots,k} a_i(x_i).\,P_i \parallel \prod_{j\in 1,\dots,l} \overline{b_j}\langle P_j\rangle$$

where we have t top-level variables, k input actions, and l output actions. We use a_i and b_j for channels in input and output actions, respectively. The first step is to remove top-level variables; this relies on the fact $\sim^{o}_{NOR}$ is an open bisimilarity. One thus obtains processes P_1 and Q_1 with only input and output actions, and both $m(P_1) < m(P)$ and $m(Q_1) < m(Q)$ hold. As a second step, the procedure exercises every output action in P_1. By definition of $\sim^{o}_{NOR}$, Q_1 should be able to match those actions. Call the resulting processes P_2 and Q_2. Recall that when an output $\overline{a}\langle P_j\rangle$ is consumed in the bisimulation game, process $m_j.\,P_j$ is added in parallel. Thus since $\#(\overline{a}\langle P\rangle_j) = \#(m_j.\,P_j)$ and the number of outputs decreases, measure m decreases as well. More precisely, one has that

$$P_2 \equiv \prod_{i\in 1,\dots,k} a_i(x_i).\,P_i \parallel \prod_{j\in 1,\dots,l} m_j.\,P_j$$

where m_j stands for a fresh name. Then, one has to consider the $k+l$ input actions in each process; their consumption proceeds as expected. One obtains processes P_3 and Q_3 that are bisimilar, with strictly decreasing measures for both processes. The procedure concludes by iterating the above steps on P_3 and Q_3. In fact, we have shown that at each step measure m strictly decreases; this guarantees that eventually one will reach processes P_n and Q_n, with $m(P_n) = m(Q_n) = 0$ as desired.

Consider now the following $\sim^{o}_{NOR}$-bisimilar processes: $P \parallel \prod_{k\in 1,\dots,j} m_k.\,P_k$ and $Q \parallel \prod_{k\in 1,\dots,j} m_k.\,Q_k$. Using the procedure above over P until it becomes $\mathbf{0}$, we obtain $\prod_{k\in 1,\dots,j} m_k.\,P_k \sim^{o}_{NOR} \prod_{k\in 1,\dots,j} m_k.\,Q_k$. This is because fresh names $m_1, \dots, m_j$ do not occur in P and Q, and hence they do not intervene in P's consumption, so the bisimilar process must still mention them and cannot mention anything else. Similarly, we can consume $\prod_{k'\in 2,\dots,j} m_{k'}.\,P_{k'}$ (i.e., all the components excepting $m_1.\;P_1$) which is match by the consumption of the corresponding $\prod_{k'\in 2,\dots,j} m_{k'}.\,Q_{k'}$. We thus end up with $m_1.\;P_1 \sim^{o}_{NOR} m_1.\,Q_1$, and we observe that the only possible action on each side is the input on m_1, which can be trivially matched by the other. We then infer that $(P_1, Q_1) \in \mathcal{S}_1$, as desired. $\square$

Lemma 4.16. *$\sim^{\star}_{HO}$ implies $\sim^{\star}_{CON}$.*

Proof. We only need to prove that $\sim_{HO}$ implies $\sim_{CON}$ by definition of open extension. We suppose $(P, Q) \in \sim_{HO}$ and $P \xrightarrow{\alpha} P'$; we need to show a matching action from Q. We proceed by case analysis on the form α can take. The only interesting case is when α is a higher-order output; the remaining clauses are the same in both relations. By definition of

$\sim_{HO}$, if $P \xrightarrow{\overline{a}\langle P''\rangle} P'$ then $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$, with both $P'' \sim_{HO} Q''$ and $P' \sim_{HO} Q'$. We need to show that, for every S such that $\mathsf{fv}(S) = \{x\}$, $S\{^{P''}/x\} \parallel P' \sim_{HO} S\{^{Q''}/x\} \parallel Q'$; this follows from $P'' \sim_{HO} Q''$ and $P' \sim_{HO} Q'$ and the fact that $\sim_{HO}$ is both a congruence and preserved by substitutions. □

Lemma 4.17. *$\sim_{CON}$ implies $\sim_{NOR}$.*

Proof. Straightforward by showing an appropriate bisimulation. The result is immediate by noticing that (i) both relations are τ-bisimulations, and that (ii) the input and output clauses of $\sim_{NOR}$ are instances of those of $\sim_{CON}$. In the output case, by selecting a process $S = m.\,x$ (with m fresh) one obtains the desired form for the clause. The input clause is similar, and follows from the definition of closed bisimulation, which holds for every closed process R; in particular, it also holds for $R = \overline{m}$ (with m fresh) as required by the clause of $\sim_{NOR}$. □

Lemma 4.18. *$\sim^{\bigstar}_{NOR}$ implies $\sim^{o}_{NOR}$.*

Proof. Remember that $\sim^{\bigstar}_{NOR}$ is the extension of $\sim_{NOR}$ to open processes (see Definition 4.6). Notice that since $\sim_{NOR}$ is an input normal bisimulation (Definition 4.2(2)), we can easily show that

$$P \sim^{\bigstar}_{NOR} Q \text{ iff } P\left\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\right\} \sim_{NOR} Q\left\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\right\}$$

where $\{x_1, \ldots, x_n\} = \mathsf{fv}(P) \cup \mathsf{fv}(Q)$ and $m_1, \ldots, m_n$ are fresh names. We now show that $\sim^{\bigstar}_{NOR}$ is an open normal bisimulation.

We first suppose $P \sim^{\bigstar}_{NOR} Q$ and $P \xrightarrow{\alpha} P'$; we need to find a matching transition $Q \xrightarrow{\alpha} Q'$. We perform a case analysis on the shape α can take. The τ and output cases are immediate by using Lemma 2.5(1 and 3). We now detail the case in which $\alpha = a(x)$. If $P \xrightarrow{a(x)} P'$ then by Lemma 2.5(1)

$$P\left\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\right\} \xrightarrow{a(x)} P'\left\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\right\}.$$

In turn, by definition of $\sim_{NOR}$, such an action guarantees that there exists a Q' that matches that input action, i.e.,

$$Q\left\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\right\} \xrightarrow{a(x)} Q'\left\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\right\}$$

with (for some fresh m)

$$P'\left\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\right\}\left\{\overline{m}/x\right\} \sim_{NOR} Q'\left\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\right\}\left\{\overline{m}/x\right\}.$$

By Lemma 2.5(3), we have $Q \xrightarrow{a(x)} Q'$, and we conclude by the equivalence above that $P' \sim^{\bigstar}_{NOR} Q'$.

The last thing to consider are those variables in evaluation context in the open processes. This is straightforward by noting that by definition of $\sim^{\bigstar}_{NOR}$, all such variables have been closed with a trigger. So, suppose $P \sim^{\bigstar}_{NOR} Q$ and

$$P\left\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x\right\} \equiv \overline{m_1} \parallel P'\left\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\right\}$$

where m_1 is fresh. We need to show that Q has a similar structure, i.e., that $Q \equiv x \parallel Q'$, with $P' \sim^{\bigstar}_{NOR} Q'$. P can perform an output action on m_1, thus evolving to $P'\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\}$. By definition of $\sim^{\bigstar}_{NOR}$, Q can match this action, and evolves to some process Q^*, with $m'.\,\mathbf{0} \parallel P'\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\} \sim^{\bigstar}_{NOR} m'.\,\mathbf{0} \parallel Q^*$, where m' is a fresh name (obtained from the definition of $\sim^{\bigstar}_{NOR}$ for output actions). The input on m' can be trivially consumed on both sides, and one has $P'\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\} \sim^{\bigstar}_{NOR} Q^*$. At this point, since m_1 is a fresh name, we know that Q involves a variable in evaluation context. Furthermore, since there is a correspondence between P' and Q^*, they should involve substitutions in the very same fresh names. More precisely, we have that there should be a Q' such that

$$Q \equiv \overline{m_1} \parallel Q'\left\{\overline{m_1}/x_1, \ldots, \overline{m_n}/x_n\right\} = \overline{m_1} \parallel Q^*$$

as desired. □

Lemma 4.19. *$\sim^{o}_{NOR}$ implies $\sim^{o}_{IO}$.*

Proof. The only difference between the bisimilarities is their output clause: they are both open bisimulations. We analyze directly the case for output action. Suppose $P \sim^{o}_{NOR} Q$ and $P \xrightarrow{\overline{a}\langle P''\rangle} P'$; we need to show a matching action from Q. By definition of $\sim^{o}_{NOR}$, if $P \xrightarrow{\overline{a}\langle P''\rangle} P'$ then also $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$, with $m.\,P'' \parallel P' \sim^{o}_{NOR} m.\,Q'' \parallel Q'$. Using this and Lemma 4.15 we conclude that $P'' \sim^{o}_{NOR} Q''$ and $P' \sim^{o}_{NOR} Q'$. □

Lemma 4.20. *In HOcore, relations $\sim_{HO}$, $\sim^{o}_{NOR}$ and $\sim_{CON}$ coincide on closed processes.*

Proof. This is an immediate consequence of previous results. In fact, we have proved (on open processes) the following implications:

1. $\sim^o_{IO}$ implies $\sim^\star_{HO}$ (Lemma 4.14).
2. $\sim^\star_{HO}$ implies $\sim^\star_{CON}$ (Lemma 4.16);
3. $\sim^\star_{CON}$ implies $\sim^\star_{NOR}$ (Lemma 4.17);
4. $\sim^\star_{NOR}$ implies $\sim^o_{NOR}$ (Lemma 4.18);
5. $\sim^o_{NOR}$ implies $\sim^o_{IO}$ (Lemma 4.19). □

We thus infer that $\sim_{HO}$ and $\sim_{CON}$ are congruence relations. Direct proofs of these results proceed by exhibiting an appropriate bisimulation and are usually hard (a sensible aspect being proving congruence for parallel composition). Congruence of higher-order bisimilarity is usually proved by appealing to, and adapting, Howe's method for the λ-calculus [24]. This is the approach followed in, e.g., [25–27]; very recent uses of Howe's method are reported in [28,29] for higher-order process calculi with *passivation* constructs.

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

Theorem 4.21. *All complete combinations of the HOcore bisimulation clauses coincide, and are decidable.*

Proof. In Lemma 4.20, we have proved that the least demanding combination ($\sim^o_{IO}$) coincides with the most demanding ones ($\sim^\star_{HO}$ and $\sim^\star_{CON}$). Decidability then follows from Lemma 4.11. □

We find this "collapsing" of bisimilarities in HOcore significant; the only similar result we are aware of is by Cao [11], who showed that strong context bisimulation and strong normal bisimulation coincide in higher-order π-calculus.

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 [6,30], as this makes some technical details simpler; however the results also hold for ordinary barbed congruence as defined in [31]. It is worth recalling that the main difference between reduction-closed barbed congruence and the barbed congruence of [31] is quantification over contexts (see (2) in Definition 5.1 below). More importantly, we consider the *asynchronous* version of barbed congruence, where barbs are only produced by output messages; we call barbed congruence *synchronous* when inputs contribute too, as in, e.g., [31]. 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 an asynchronous calculus — 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* (see, e.g., [32]) the input clause is weakened so as 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, for some fresh name m,
  1. either $Q \xrightarrow{a(x)} Q'$ and $P'\{\overline{m}/x\} \mathcal{R} Q'\{\overline{m}/x\}$;
  2. or $Q \xrightarrow{\tau} Q'$ and $P'\{\overline{m}/x\} \mathcal{R} Q' \parallel \overline{a}\langle \overline{m}\rangle$.

We now define asynchronous barbed congruence. 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 symmetric relation on closed processes that is

1. a τ-bisimulation (Definition 4.1(1));
2. context-closed (i.e., $P \simeq Q$ implies $C[P] \simeq C[Q]$, for all closed contexts $C[\cdot]$);
3. barb preserving (i.e., if $P \simeq Q$ and $P \downarrow_{\overline{a}}$, then also $Q \downarrow_{\overline{a}}$).

In synchronous barbed congruence, input barbs $P \downarrow_a$ are also observable.

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

Proof. We first show that $\sim_{NOR}$ implies $\simeq$, and then its converse, which is harder. The relation $\sim_{NOR}$ satisfies the conditions in Definition 5.1 as follows. First, both relations are τ-bisimulations so condition (1) above trivially holds. Second, the context-closure condition follows from the fact that $\sim_{NOR}$ is a congruence. Finally, the barb-preserving condition is seen to hold by definition of $\sim_{NOR}$: having $P \sim_{NOR} Q$ implies that an output action of P on a has to be matched by an output action of Q on a; hence, we have that if $P \downarrow_{\overline{a}}$, then also $Q \downarrow_{\overline{a}}$.

Now the converse. We show that relation $\simeq$ satisfies the three conditions for $\sim_{NOR}$ in Definition 4.3. Suppose $P \simeq Q$ and $P \xrightarrow{\alpha} P'$; we have to show a matching transition $Q \xrightarrow{\alpha} Q'$. We proceed by a case analysis on the form α can take.

Case $\boldsymbol{\alpha = \tau}$. Since by definition $\simeq$ is a τ-bisimulation, then there is a Q' such that $Q \xrightarrow{\tau} Q'$ and $P' \simeq Q'$ and we are done.

Case $\boldsymbol{\alpha = \overline{a}\langle P''\rangle}$. We have $P \xrightarrow{\overline{a}\langle P''\rangle} P'$: it can be shown that $\simeq$ is an output normal bisimulation by showing a suitable context. Let $C^a_{\mathrm{o}}[\cdot]$ be the context

$$C^a_{\mathrm{o}}[\cdot] = [\cdot] \parallel a(x).\,(m.\,x \parallel \overline{n} \parallel n.\,\mathbf{0})$$

where m, n are fresh names. We then have $C^a_{\mathrm{o}}[P] \xrightarrow{\tau} P_1$ with $P_1 \downarrow_{\overline{n}}$. Indeed, we have $P_1 \equiv P' \parallel m.\,P'' \parallel \overline{n} \parallel n.\,\mathbf{0}$. By definition of $\simeq$, we have also $C^a_{\mathrm{o}}[Q] \xrightarrow{\tau} Q_1$ and necessarily, $Q_1 \downarrow_{\overline{n}}$. Since n is a fresh name, we infer that Q also has an output on a, such that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$ and hence $Q_1 \equiv Q' \parallel m.\,Q'' \parallel \overline{n} \parallel n.\,\mathbf{0}$. Note that (P_1, Q_1) is in $\simeq$. They can consume the actions on n; since it is a fresh name, only the corresponding τ action of Q_1 can match it. As a result, both processes evolve to processes $P' \parallel m.\,P''$ and $Q' \parallel m.\,Q''$ that are still in $\simeq$. We then conclude that $\simeq$ is an output normal bisimulation.

Case $\boldsymbol{\alpha = a(x)}$. We have $P \xrightarrow{a(x)} P'$. Again, to show that $\simeq$ is an input normal bisimulation, we define a suitable context. Here, the asynchronous nature of HOcore (more precisely, the lack of output prefixes, which prevents the control of output actions by modifying their continuation) and the input clause for $\sim_{NOR}$ it induces (reported above) result in a more involved definition of these contexts. Notice that simply defining a context with an output action on a so as to force synchronization with the input action does not work here: process P itself could contain other output actions on a that could synchronize with the input we are interested in, and as output actions have no continuation, it is not possible to put a fresh barb indicating it has been consumed. We overcome this difficulty by (i) renaming *every output* in P, so as to avoid the possibility of τ actions (including those coming from synchronizations on channels different from a), (ii) consuming the input action on a (by placing the renamed process in a suitable context) and then (iii) restoring the initial outputs.

We define a context for (i) above, i.e., to rename every output in a process so as to prevent τ actions. We start by denoting by $out(P)$ the multiset of names in output subject position in a process P. Further, let σ denote an injective relation between each occurrence of name in $out(P)$ and a fresh name. Let $C[\cdot]$ be the context

$$C[\cdot] = [\cdot] \parallel \prod_{(b_i, c_i) \in \sigma} b_i(x).\,\overline{c_i}\langle x\rangle$$

which uniquely renames every output $\overline{b_i}\langle S_i\rangle$ as $\overline{c_i}\langle S_i\rangle$. (We shall use c_i ($i \in 1, \ldots, n$) to denote the fresh names for the renamed outputs.) Consider now processes $C[P]$ and $C[Q]$: since the renaming is on fresh channels, it can be ensured that the τ action due to the renaming of one output on one process is matched by the other process with a τ action that corresponds to the renaming of the same output. At the end, after a series of n τ actions, $C[P]$ and $C[Q]$ become processes P_1 and Q_1 that have no τ actions arising from their subprocesses and that are in $\simeq$. At this point it is then possible to use a context for (ii), to capture the input action on a in P_1. Let $C^a_{\mathrm{i}}[\cdot]$ be the context

$$C^a_{\mathrm{i}}[\cdot] = [\cdot] \parallel \overline{a}\langle\overline{m}\rangle$$

where m is a fresh name. We then have

$$C^a_{\mathrm{i}}[P_1] \xrightarrow{\tau} \overline{c_1}\langle S_1\rangle \parallel \cdots \parallel \overline{c_n}\langle S_n\rangle \parallel P_a\{\overline{m}/x\} \parallel P'' = P_2$$

which, by definition of $\simeq$, implies that also it must be the case that, for some process Q_2, $C^a_{\mathrm{i}}[Q_1] \xrightarrow{\tau} Q_2$. In fact, since there is a synchronization at a, it implies that Q_1 must have at least one input action on a. More precisely, we have

$$Q_2 \equiv \overline{c_1}\langle S_1\rangle \parallel \cdots \parallel \overline{c_n}\langle S_n\rangle \parallel Q_a\{\overline{m}/x\} \parallel Q''.$$

We notice that P_2 and Q_2 are still in $\simeq$; it remains however to perform (iii), i.e., to revert the renaming made by $C[\cdot]$. To do so, we proceed analogously as before and define the context

$$C'[\cdot] = [\cdot] \parallel \prod_{(c_i, b_i) \in \sigma^{-1}} c_i(x).\,\overline{b_i}\langle x\rangle.$$

We have that each of $C'[P_2]$ and $C'[Q_2]$ produces n τ steps that exactly revert the renaming done by context $C[\cdot]$ above and lead to P_3 and Q_3, respectively. This renaming occur in lockstep (and no other τ action may be performed by Q_2), as each one removes a barb on a fresh name, thus the other process has to remove the same barb by doing the renaming. Hence, P_3

and Q_3 have the same output actions as the initial P and Q, and since $\simeq$ is a τ-bisimulation we have $P_3 \simeq Q_3$. To conclude, we remark that $C^a_{\mathrm{i}}[P] \xrightarrow{\tau} P_3$ in one step. Indeed, we have

$$C^a_{\mathrm{i}}[P] \equiv T \parallel a(x).\, P_a \parallel P'' \parallel \overline{a}\langle\overline{m}\rangle \xrightarrow{\tau} P_3 = P'\{\overline{m}/x\}$$

where T stands for all the output actions in P (on which the renaming took place). By doing and undoing the renaming on every output, we were able to infer that Q has an analogous structure

$$C^a_{\mathrm{i}}[Q] \equiv T' \parallel a(x).\, Q_a \parallel Q'' \parallel \overline{a}\langle\overline{m}\rangle \xrightarrow{\tau} Q_3$$

where T' stands for all the output actions in Q. Let $Q' = T' \parallel Q_a \parallel Q''$, we then have $Q \xrightarrow{a(x)} Q'$ and $Q_3 = Q'\{\overline{m}/x\}$. To summarize, we have $P \xrightarrow{a(x)} P'$, $Q \xrightarrow{a(x)} Q'$, and $P'\{\overline{m}/x\} \simeq Q'\{\overline{m}/x\}$ with m fresh. Hence we conclude that $\simeq$ is an input normal bisimulation. □

Remark 5.3. The proof relies on the fact that HOcore has no operators of choice and restriction. In fact, choice would prevent the renaming to be reversible, and restriction would prevent the renaming using a context as some names may be hidden. The higher-order aspect 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.

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 4.21.*

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 the main forms of bisimilarity for higher-order process calculi coincide in HOcore. We therefore simply call *bisimilarity* such a relation, and write it as $\sim$. Here we present a sound and complete axiomatization of bisimilarity. We do so by adapting to a higher-order setting results by Moller and Milner on unique decomposition of processes [16,33], and by Hirschkoff and Pous on an axiomatization for a fragment of (finite) CCS [17]. We then exploit this axiomatization to derive complexity bounds for bisimilarity checking.

6.1. Axiomatization

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

Proof. Suppose, for a contradiction, that there exist P, Q such that $P \sim Q$ with $\#(P) < \#(Q)$ and choose a P with a minimal size. If Q has no transition enabled, then it must be $\mathbf{0}$, thus $\#(Q) = 0$, which is impossible as $\#(Q) > \#(P) \geq 0$.

We thus have $Q \xrightarrow{\alpha} Q'$, hence there is a P' such that $P \xrightarrow{\alpha} P'$ with $P' \sim Q'$. We consider two cases, depending on the shape of α (we do not consider τ actions, as such an action implies both an input and an output).

If α is an input action, we have $Q \xrightarrow{a(x)} Q'$, and since $P \sim Q$, then also $P \xrightarrow{a(x)} P'$. We then have that $\#(P') = \#(P) - 1$ and $\#(Q') = \#(Q) - 1$, so it follows that $\#(P') < \#(Q')$. Further, one has $\#(P') < \#(P)$, which contradicts the minimality hypothesis.

Now suppose α is an output action: we have $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$, and by definition of $\sim$, also that $P \xrightarrow{\overline{a}\langle P''\rangle} P'$ with both $P' \sim Q'$ and $P'' \sim Q''$. By the definition of size, we have that $\#(P') = \#(P) - (1 + \#(P''))$ and $\#(Q') = \#(Q) - (1 + \#(Q''))$. Notice that P'', Q'' are strict subterms of P and Q, respectively. If their size is not the same, we have a contradiction. Otherwise, we have $\#(P') < \#(Q')$ and also $\#(P') < \#(P)$, which is also a contradiction. □

Following [16,33] we prove a result of unique prime decomposition of processes.

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

Proof. The proof, which proceeds by induction on $\#(P) + \#(Q) + \#(R)$, is a simple adaptation of the one in [16]. We detail it below. We simultaneously prove the following by induction on $\#(P) + \#(Q) + \#(R)$:

1. if $P \parallel R \sim Q \parallel R$ then $P \sim Q$;
2. if $R \xrightarrow{\alpha} R'$ and $P \parallel R \sim Q \parallel R'$, then $Q \xrightarrow{\alpha} Q'$ for some $P \sim Q'$.

Note that every reduction $P \xrightarrow{\alpha} P'$ decreases the size of P if α is not a τ-transition.

1. Assume that $P \parallel R \sim Q \parallel R$ and suppose that $P \xrightarrow{\alpha} P'$, for some input or output action α (we do not consider τ-transitions as we rely on the fact that $\sim^o_{IO}$ characterizes barbed congruence). We then have $P \parallel R \xrightarrow{\alpha} P' \parallel R$, which can be matched by either:
   (a) $Q \xrightarrow{\alpha} Q'$ and $P' \parallel R \sim Q' \parallel R$; or
   (b) $R \xrightarrow{\alpha} R'$ and $P' \parallel R \sim Q \parallel R'$.
   For case (1a), by induction hypothesis (1) we have $P' \sim Q'$. For case (1b), by induction hypothesis (2) we have $Q \xrightarrow{\alpha} Q'$ and $P' \sim Q'$.
   The last case to consider is for when P contains a variable. If R also contains a variable, then it may be removed from both sides from R and we conclude by induction hypothesis (1). If there is no variable in R, then it occurs in both P and Q and we also conclude by induction hypothesis (1).
   The case when starting from Q is symmetric, thus we have $P \sim Q$.
2. Assume that $R \xrightarrow{\alpha} R'$ and $P \parallel R \sim Q \parallel R'$. We thus have $P \parallel R \xrightarrow{\alpha} P \parallel R'$, and there exists some S such that $Q \parallel R' \xrightarrow{\alpha} S$ with $P \parallel R' \sim S$. As before, we distinguish the source of the action. We either have:
   (a) $Q \xrightarrow{\alpha} Q'$ and $P \parallel R' \sim Q' \parallel R'$; or
   (b) $R' \xrightarrow{\alpha} R''$ and $P \parallel R' \sim Q \parallel R''$.
   For case (2a), we have $P \sim Q'$ by induction hypothesis (1).
   For case (2b), we apply induction hypothesis (2) and thus we have $Q \xrightarrow{\alpha} Q'$ with $P \sim Q'$. □

Proposition 6.4 (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 $\prod_{j=1}^m Q_j$, then $n = m$ and there is a permutation σ of $\{1, \ldots, n\}$ such that $P_i \sim Q_{\sigma(i)}$ for each $i \in \{1, \ldots, n\}$).*

Proof. The proof is also similar to the one in [16]. We also detail this proof.

We proceed by induction on $\#(P)$. Assume there are two prime decompositions, $P \sim \prod_{i=1}^n P_i$ and $P \sim Q = \prod_{j=1}^m Q_j$. First, if some $P_k \sim Q_l$ (w.l.o.g assume that $P_1 \sim Q_1$), then we have the following two prime decompositions for P: $P \sim P_1 \parallel \prod_{i=2}^n P_i$ and $P \sim P_1 \parallel \prod_{j=2}^m Q_j$. By Proposition 6.3, we have $\prod_{i=2}^n P_i \sim \prod_{j=2}^m Q_j$. As these are prime decomposition of a smaller process, by induction they are also unique (up to bisimilarity and permutation of indices). Thus the two decompositions $\prod_{i=1}^n P_i$ and $\prod_{j=1}^m Q_j$ are also identical up to bisimilarity and permutation of indices.

We now assume, for a contraction, that for every i, j we have $P_i \not\sim Q_j$. If either $n = 1$ or $m = 1$, then (by definition of a prime process) $n = m = 1$ and $P_1 \sim Q_1$, a contradiction.

We thus assume that $n, m \geq 2$ and that (w.l.o.g.) that $\#(P_1) \leq \#(P_i)$ and $\#(P_1) \leq \#(Q_j)$ for every i and j. As P_1 is not empty, it contains either a variable or can do an input or output action.

In the case it contains a variable, then we must have $P_1 \sim x$ (the only prime process that contains a variable is one that is equivalent to a variable). As $P \sim \prod_{j=1}^m Q_j$, then one of the Q_j must be equivalent to x, which is a contradiction.

Let us now consider the case where P_1 does an input or output action α: $P_1 \xrightarrow{\alpha} R$. We necessarily have $\#(R) < \#(P_1) \leq \#(P)$, thus R has a unique prime decomposition, which we write $\prod_{k=1}^l R_k$. We thus have $P \xrightarrow{\alpha} P'$ with unique decomposition (as $\#(P') < \#(P)$) $P' \sim \prod_{k=1}^l R_k \parallel \prod_{i=2}^n P_i$. Since $P \sim Q$, we thus have $Q \xrightarrow{\alpha} Q' \sim P'$, thus for some Q_j (w.l.o.g. Q_1), we have $Q_1 \xrightarrow{\alpha} T$ and $Q' = T \parallel \prod_{j=2}^m Q_j$. As the decomposition $P' \sim Q'$ is unique, and as $m \geq 2$, there is some process Q_2 and it must be equivalent to some process in $\prod_{k=1}^l R_k \parallel \prod_{i=2}^n P_i$. It cannot be one of the R_k as $\#(R_k) < \#(P_1) \leq \#(Q_2)$ (and their size have to match by Lemma 6.1), so it must be one of $P_2, \ldots, P_n$, a contradiction. □

Both the key law for the axiomatization and the following results are inspired by similar ones by Hirschkoff and Pous [17] for pure CCS. Using their terminology, we call *distribution law*, briefly (DIS), the axiom schema below (recall that $\prod_1^k Q$ denotes the parallel composition of k copies of Q).

$$a(x).\left(P \parallel \prod_1^{k-1} a(x).P\right) = \prod_1^k a(x).P \qquad (DIS)$$

We then call *extended structural congruence*, written $\equiv_E$, the extension of the structural congruence relation ($\equiv$, Definition 2.2) with the axiom schema (DIS). We write $P \rightsquigarrow Q$ when there are processes P' and Q' such that $P \equiv P'$, $Q' \equiv Q$ and Q' is obtained from P' by rewriting a subterm of P' using law (DIS) from left to right. Below we prove that $\equiv_E$ provides an algebraic characterization of $\sim$ in HOcore. Establishing the soundness of $\equiv_E$ is easy; below we discuss completeness.

Definition 6.5. A process P is in *normal form* if it cannot be further simplified in the system $\equiv_E$ by using $\rightsquigarrow$.

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 not 0.

Lemma 6.6. *If $P \rightsquigarrow Q$, then $P \sim Q$. Also, for any P, $P \sim \mathsf{n}(P)$.*

Proof. The proof proceeds by showing that $(\rightsquigarrow \cup (\rightsquigarrow)^{-1} \cup \equiv)$ is a bisimulation (as $\sim^o_{IO}$, for instance). □

Lemma 6.7. *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.*

Proof. By Lemma 6.6, $a(x).\,P \sim \mathsf{n}(Q \parallel Q')$. Furthermore, by Proposition 6.4, we have that

$$\mathsf{n}(Q \parallel Q') \equiv \prod_{i \le k} a_i(x_i).\,A_i \parallel \prod_{j \le l} \overline{b_j}\langle B_j \rangle \parallel \prod_{h \le m} x_h,$$

where the processes $a_i(x_i).\,A_i$ and $\overline{b_j}\langle B_j \rangle$ and x_h are in normal form and prime. Since the prefix $a(x)$ must be triggered to answer any challenge from the right-hand side, we have $a_i = a$, and $x_i = x$ (this can be obtained via α-conversion, but we can suppose that $a_i(x_i).\,A_i$ was already α-converted to the correct form), and we have $l = 0$ and $m = 0$ (there are no output nor top-level variables in the prime decomposition). As there are at least two processes that are not $\mathbf{0}$, we have $k > 1$. To summarize:

$$a(x).\,P \sim \prod_{i \le k} a(x).\,A_i.$$

After an input action on the right-hand side, we derive

$$P \sim A_i \parallel \prod_{l \ne i} a(x).\,A_l$$

for every $i \le k$. In particular, when $i \ne j$, we have

$$P \sim A_i \parallel a(x).\,A_j \parallel \prod_{l \notin \{i,j\}} a(x).\,A_l \qquad P \sim A_j \parallel a(x).\,A_i \parallel \prod_{l \notin \{i,j\}} a(x).\,A_l$$

and, by Proposition 6.3, $A_i \parallel a(x).\,A_j \sim A_j \parallel a(x).\,A_i$. Since $a(x).\,A_i$ is prime and it has larger size than A_i (and any of its components), it should correspond in the prime decomposition to $a(x).\,A_j$, i.e., $a(x).\,A_i \sim a(x).\,A_j$. As this was shown for every $i \ne j$, we thus have $a(x).\,P \sim \prod_1^k a(x).\,A_1$ with $k > 1$ and $a(x).\,A_1$ in normal form. □

Lemma 6.8. *For A, B in normal form, if $A \sim B$ then $A \equiv B$.*

Proof. We show, simultaneously, the following two properties:

1. if A is a prefixed process in normal form, then A is prime;
2. for any B in normal form, $A \sim B$ implies $A \equiv B$.

We proceed by induction on n, for all A with $\#(A) = n$. The case $n = 0$ is immediate as the only process of this size is $\mathbf{0}$. Suppose that the property holds for all $i < n$, with $n \ge 1$. In the reasoning below, we exploit the characterization of $\sim$ as $\sim^o_{IO}$.

1. Process A is of the form $a(x).\,A'$. Suppose, as a contradiction, that A is not prime. Then we have $A \sim P_1 \parallel P_2$ with P_1 and P_2 non-trivial. By Lemma 6.7, then $A \sim \Pi_1\ ka(x).\,B$ with $k > 1$ and $a(x).\,B$ in normal form. By consuming the prefix on the left-hand side, we have $A' \sim B \parallel \Pi_1^{k-1}\ a(x).\,B$. It follows by induction (using property (2)) that $A' \equiv B \parallel \Pi_1^{k-1}\ a(x).\,B$, and hence also $A \equiv a(x).\,(B \parallel \Pi_1^{k-1}\ a(x).\,B)$. This is impossible, as A is in normal form.
2. Suppose $A \sim B$. We proceed by case analysis on the structure of A.
   - Case $A = x$. We have that B should be the same variable, so $A \equiv B$ trivially.
   - Case $A = \overline{a}\langle P \rangle$. We have that $B = \overline{a}\langle P' \rangle$ with $P \sim P'$ by definition of $\sim^o_{IO}$. By the induction hypothesis, $P \equiv P'$, thus $\overline{a}\langle P \rangle \equiv \overline{a}\langle P' \rangle$.

- Case $A = a(x).\, A'$. We show by contradiction that $B = a(x).\, B'$. Assume $B = Q \parallel Q'$, then by Lemma 6.7, A is a parallel composition of at least two processes. But according to the first property, as A is prefixed, it is prime, a contradiction. We thus have $B = a(x).\, B'$ with $A' \sim^{o}_{IO} B'$. By induction this entails $A' \equiv B'$ and $A \equiv B$.
- Case $A = \prod_{i \le k} P_i$ with $k > 1$ where no P_i has a parallel composition at top-level. We reason on the possible shape of the P_i.
  If there exists j such that $P_j = x$ then also $B \equiv x \parallel B'$. The thesis then follows by induction hypothesis on $\prod_{i \le k, i \neq j} P_i$ and B'.
  If P_i is an output, B must contain an output on the same channel. The thesis then follows by applying the induction hypothesis twice, to the arguments and to the other parallel components.
  The last case is when $A \equiv \prod_{i \le k} a_i(x_i).\, A_i$ with $k > 1$. We know by the induction hypothesis (property (1)) that each component $a_i(x_i).\, A_i$ is prime. Similarly, it must be $B \equiv \prod_{i \le l} b_i(x_i).\, B_i$ with $b_i(x_i).\, B_i$ prime for all $i \le l$. By Proposition 6.4 (unique decomposition), we infer $k = l$ and $a_i(x_i).\, A_i \sim b_i(x_i).\, B_i$ (up to a permutation of indices). Thus $a_i = b_i$ and $A_i \sim B_i$; then by induction $A_i \equiv B_i$ for all i, which finally implies $A \equiv B$. □

The theorem below follows from Lemmas 6.6 and 6.8.

Theorem 6.9. *For any processes P and Q, we have $P \sim Q$ iff $\mathsf{n}(P) \equiv \mathsf{n}(Q)$.*

Corollary 6.10. *$\equiv_E$ is a sound and complete axiomatization of bisimilarity in HOcore.*

6.2. Complexity of bisimilarity checking

To analyze the complexity of deciding whether two processes are bisimilar, one could apply the technique from [15], and derive that bisimilarity is decidable in time which is linear in the size of the LTS for $\sim^{o}_{IO}$ (which avoids τ transitions). This LTS is however exponential in the size of the process. A more efficient solution exploits the axiomatization above: one can first normalize processes and then reduce bisimilarity to syntactic equivalence of the obtained normal forms.

For simplicity, we assume a process P is represented as an ordered tree (but we will transform it into a DAG during normalization). In the following, let us denote with $t[m_1, \ldots, m_k]$ the ordered tree with root labeled t and with (ordered) descendants $m_1, \ldots, m_k$. We write $t[]$ for a tree labeled t and without descendants (i.e., a leaf).

Definition 6.11 (*Tree representation*). Let P be a HOcore process. Its associated ordered tree representation is labeled and defined inductively by

- $\mathsf{Tree}(\mathbf{0}) = \mathbf{0}[]$
- $\mathsf{Tree}(x) = \mathsf{db}(x)[]$
- $\mathsf{Tree}(\bar{a}\langle Q\rangle) = \bar{a}[\mathsf{Tree}(Q)]$
- $\mathsf{Tree}(a(x).\, Q) = a[\mathsf{Tree}(Q)]$
- $\mathsf{Tree}(\prod_{i=1}^{n} P_i) = \prod_{i=1}^{n}[\mathsf{Tree}(P_1), \ldots, \mathsf{Tree}(P_n)]$

where db is a function assigning De Bruijn indices [34] to variables. Parallel composition is n-ary, thus we can assume without loss of generality that children of parallel composition nodes are not parallel composition nodes (i.e., we can always flatten them).

We now describe the normalization steps by characterizing them as reductions as well as pseudocode descriptions. The first step deals with parallel composition nodes: it removes all unnecessary $\mathbf{0}$ nodes, and relabels the nodes when the parallel composition has only one or no descendants.

Normalization Step 1. Let $\rightsquigarrow_{\mathrm{N1}}$ be a transformation rule over trees associated to HOcore processes defined by:

1. $\prod_{i=1}^{0}[] \rightsquigarrow_{\mathrm{N1}} \mathbf{0}[]$
2. $\prod_{i=1}^{1}[\mathsf{Tree}(P_1)] \rightsquigarrow_{\mathrm{N1}} T$ if $\mathsf{Tree}(P_1) \rightsquigarrow_{\mathrm{N1}} T$
3. $\prod_{i=1}^{n}[\mathsf{Tree}(P_1), \ldots, \mathsf{Tree}(P_n)] \rightsquigarrow_{\mathrm{N1}} \prod_{i=1}^{m}[T_{\sigma(1)}, \ldots, T_{\sigma(m)}]$, if $\mathsf{Tree}(P_i) \rightsquigarrow_{\mathrm{N1}} T_i$ for each i, where $m < n$ is the number of trees in $T_1, \ldots, T_n$ that are different from $\mathbf{0}[]$, and where σ is a bijective function from $\{1, \ldots, m\}$ to $\{i \mid i \in \{1, \ldots, n\} \wedge T_i \neq \mathbf{0}[]\}$.

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

Normalization Step 2. Let $\rightsquigarrow_{\mathrm{N2}}$ be a transformation rule over trees associated to HOcore processes, defined as follows. 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 1 NS1(n)

```
Require: A tree node $n$
 1: if $n.$type = 'par' then
 2:   if $n.$numChildren = 0 then
 3:     $n.$type = 'zero'
 4:   else if $n.$numChildren = 1 then
 5:     $aux = n.$children[1]
 6:     $n = *(n.$children[1])
 7:     delete($aux$)
 8:     NS1($n$)
 9:   else
10:     $j = 1$
11:     for $i = 1$ to $n.$numChildren do
12:       NS1($*(n.$children[$i$]))
13:       if $*(n.$children[$i$]).type $\neq$ 'zero' then
14:         $n.$children[$j$] = $n.$children[$i$]
15:         $j = j + 1$
16:       else
17:         delete($n.$children[$i$])
18:       end if
19:     end for
20:     $n.$numChildren = $j - 1$
21:   end if
22: else
23:   for $i = 1$ to $n.$numChildren do
24:     NS1($*(n.$children[$i$]))
25:   end for
26: end if
```

The last normalization step applies DIS from left to right if possible:

Normalization Step 3. Let $\rightsquigarrow_{\text{N3}}$ be a transformation rule over trees associated to HOcore processes, defined by:

$$a\left[\prod_{i=1}^{k+1}[\mathsf{Tree}(P),\ \mathsf{Tree}(a(x).\,P),\ \ldots,\ \mathsf{Tree}(a(x).\,P)]\right] \rightsquigarrow_{\text{N3}}$$
$$\prod_{j=1}^{k+1}[\mathsf{Tree}(a(x).\,P),\ \ldots,\ \mathsf{Tree}(a(x).\,P)]$$

where $\mathsf{Tree}(a(x).\,P)$ appears k times in the left-hand side, and $k+1$ times in the right-hand side.

We now present the pseudocode for the three normalization steps. We use a rather standard pointer-based implementation of trees, with the following notational conventions. A node of a tree is a record with four attributes (or fields). Given a node n, we use $n.$`attr` to refer to the attribute `attr` of n. Each node has an attribute `type`, which represents the type of node; possible values are `zero` (for a nil process node); `par` (for a parallel composition node); `inp` and `out` (for nodes for input and output actions, respectively); and `var` (for nodes associated to variables). Each node n has also an attribute `children`, an array of pointers to the children of node n. Hence, $n.$`children`$[i]$ stands for the pointer to the ith children of n. We use $*$ for dereferencing. Furthermore, we assume an operation delete for node deletion (freeing the allocated memory). The length of `children` is stored as attribute `numChildren`. Nodes may also have an attribute `info`, containing the subject name for input and output nodes and the De Bruijn indices for variable nodes. When we use assignment $n_1 = n_2$ on record types we assume that all the fields are copied.

Consider the pseudocode for Normalization Step 1. The step performs a visit of the tree rooted at n applying the normalization rules. Item (1) of the normalization step is represented by lines 2–3, while item (2) is represented by lines 4–7. Item (3) is represented by lines 10–21.

In the pseudocode for Normalization Step 2 we assume auxiliary operations comp for comparing the subtrees rooted at two nodes, sortChildren for lexicographic ordering of children of a node (e.g., using MergeSort) and deleteRec for deleting a subtree.

The pseudocode for Normalization Step 3 exploits an auxiliary operation too. Operation subtree(p_1, p_2, c) takes as arguments two pointers to nodes p_1 and p_2 and a name c. It checks whether p_2 points to an input node with label c and with

Normalization Step 2 NS2(n)

```
Require: A tree node $n$
for $i = 1$ to $n.$numChildren do
  NS2($*(n.$children$[i])$)
end for
if $n.$type $=$ 'par' then
  sortChildren($n$)
  for $i = 1$ to $n.$numChildren $- 1$ do
    if comp($*(n.$children$[i]), *(n.$children$[i+1])$) = true then
      deleteRec($n.$children$[i+1]$)
      $n.$children$[i+1] = n.$children$[i]$
    end if
  end for
end if
```

Normalization Step 3 NS3(n)

```
Require: A tree node $n$
for $i = 1$ to $n.$numChildren do
  NS3($*(n.$children$[i])$)
end for
if $n.$type $=$ 'inp' then
  $s = *(n.$children$[1])$
  if $s.$type $='$ par$'$ then
    $c = n.$name
    $ok$ =false
    if subtree($s.$children$[1], s.$children$[2], c$) then
      $small = 1$, $big = 2$, $ok$ =true
    else if subtree($s.$children$[s.$numChildren$], s.$children$[1], c$) then
      $small = s.$numChildren, $big = 1$, $ok$ =true
    end if
    for $i = 2$ to $n.$numChildren $- 1$ do
      if $s.$children$[big] \neq s.$children$[i]$ then
        $ok$ =false
      end if
    end for
    if $ok$ =true then
      $n = s$
      delete($s$)
      $aux = n.$children$[small]$
      $n.$children$[small] = n.$children$[big]$
      deleteRec($*(n.$children$[big]).$children$[1]$)
      $*(n.$children$[big]).$children$[1] = aux$
    end if
  end if
end if
```

a subtree equal to p_1. The De Bruijn indices in p_1 must differ by 1 w.r.t. the corresponding ones of p_2 for the comparison to succeed. This is required to take into account the additional input prefix in front of p_2.

In Normalization Step 3, lines 9–18 check whether node n is the root of a subtree to which the axiom schema DIS can be applied. There are two possibilities to consider, since the subtree P can be either the first child in the parallel composition or the last one (remember that children of the parallel composition have been sorted by Normalization Step 2). If variable *ok* is true then the axiom schema can be applied, and variables *small* and *big* point to the child containing P and to one of the other children, respectively (all the other children correspond to the same node). Lines 20-25 apply the axiom schema if needed. Line 23 makes all the children point to the same node, the one indexed by *big*, by changing the only different pointer, the one indexed by *small*. Then the child of this node is deleted together with its subtree, and replaced by pointing to the node that was pointed by *small*, which is pointed now by *aux*. This is needed to have the correct De Bruijn indices. In fact, De Bruijn indices of the child indexed by *small* already have the values needed for the final term.

We relate now normalization and bisimilarity.

Lemma 6.12. *Let T_P, T_Q be two tree representations of processes P and Q (as in Definition 6.11), normalized according to normalization Steps 1 and 2. Then $P \equiv Q$ iff $T_P = T_Q$.*

Proof. Immediate from Definitions 2.2 and 6.11, and from Normalization Steps 1 and 2. In particular, $\rightsquigarrow_{\mathtt{N1}}$ corresponds to the elimination of all occurrences of **0** in parallel, and $\rightsquigarrow_{\mathtt{N2}}$ corresponds to the choice of a representative process, up to associativity and commutativity. $\Box$

Lemma 6.13. *Let P, Q be processes and T_P, T_Q their tree representations normalized according to Normalization Steps 1, 2 and 3. Then $P \sim Q$ iff $T_P = T_Q$.*

Proof. Immediate using Lemmas 6.12 and 6.6 ($P \rightsquigarrow Q$ implies $P \sim Q$). $\Box$

We now give a lemma on the cost of sorting the tree representation of a process. Given a process P, we define the size of its tree representation T_P to be the number of nodes of the tree, and denote it as $\mathbf{size}(P)$.

Lemma 6.14. *Consider n HOcore processes $P_1, \ldots, P_n$ and their tree representations $T_{P_1}, \ldots, T_{P_n}$. Their sorting has complexity $O(t \; \log n)$, where $t = \sum_{i \in 1,\ldots,n} \mathbf{size}(P_i)$.*

Proof. Let us assume MergeSort as sorting algorithm. MergeSort sorts a list of elements by (i) splitting the list to be sorted in two; (ii) recursing on both sublists; and (iii) merging the sorted sublists. A merge function starts by comparing the first element of each list and then copies the smallest one to the new list. Comparing two elements P_i, P_j costs $\min(\mathbf{size}(P_i), \mathbf{size}(P_j))$. As each T_{P_i} is considered once (when it is copied to the new list) the cost of merging two lists is the sum of the size of their elements (the actual copying of an element has constant cost since it is just a pointer operation). Let us call a *slice* of MergeSort the juxtaposition of every recursive call at the same depth. In this way, e.g., the first slice considers the lists when recursion depth is equal to 1: the first two recursive calls, each one having half of the original list. In general, at every slice one finds a partition of the list in 2^d sublists, where d is the recursion depth. Each recursive call in every slice is going to merge two sublists, with a complexity of the sum of the sizes of these sublists. Summing everything, we get a cost of $t = \sum_{i \in 1,\ldots,n} \mathbf{size}(P_i)$ at each recursion depth. Therefore, as there are $\log n$ different depths, the total complexity is $O(t \log n)$. $\Box$

Theorem 6.15. *Consider two HOcore processes P and Q. $P \sim Q$ can be decided in time $O(n^2 \log m)$ where $n = \max(\mathbf{size}(P), \mathbf{size}(Q))$ (i.e., the maximum number of nodes in the tree representations of P and Q) and m is the maximum branching factor in them (i.e., the maximum number of components in a parallel composition).*

Proof. Bisimilarity check proceeds as follows: first normalize the tree representations of the two processes, then check them for syntactic equality.

Normalization Step 1 can be performed in time $O(n)$, as can be seen from the corresponding pseudocode. Normalization Step 2 performs a visit of the tree, sorting and removing duplicates from the children of parallel composition nodes. By Lemma 6.14 sorting can be done in $O(n \log m)$ for each parallel composition node. Removing duplicates requires just $O(n)$ time. Normalization Step 3 visits the tree too, possibly reconfiguring input nodes. The check for applicability requires one comparison ($O(n)$) and the check that all the other components coincide (simply check that the subtrees have been merged by Normalization Step 2: $O(n)$). Applying $\rightsquigarrow_{\mathtt{N3}}$ simply entails collapsing the trees ($O(n)$). Other nodes require no operations.

Thus the normalization for a single node can be done in $O(n \log m)$, and the whole normalization can be done in $O(n^2 \log m)$. $\Box$

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 [4,6] and most of the results in Sections 4–6 would break. In particular, higher-order and context bisimilarities are different and both undecidable [3,20].

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 (or similar laws allowing to swap input and restriction), 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. For instance, one would have bounded outputs as actions, as well as rules

$$\textsc{StRes} \quad \frac{P \xrightarrow{\alpha} P' \qquad z \notin \mathsf{fn}(\alpha)}{\nu z P \xrightarrow{\alpha} \nu z P'}$$

Table 3
Encoding of PCP.

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	$C_k = up(x).\,low(y).\,(\overline{up}\langle[\![u_k, x]\!]_u\rangle \parallel \overline{low}\langle[\![l_k, y]\!]_l\rangle)$
STARTERS	$S_k = \overline{up}\langle[\![u_k, \overline{b}]\!]_u\rangle \parallel \overline{low}\langle[\![l_k, b.\,\overline{success}]\!]_l\rangle$
EXECUTOR	$E = up(x).\,low(y).\,(x \parallel y)$
SYSTEM	$P_j = \boldsymbol{\nu} up\ \boldsymbol{\nu} low\ \boldsymbol{\nu} a\ \boldsymbol{\nu} b\ (S_j \parallel !\prod_k C_k \parallel E)$

$$\text{StOpen}\quad \frac{P \xrightarrow{\nu\tilde{v}\,\overline{a}\langle R\rangle} P' \qquad z \in \mathrm{fn}(R) \setminus \tilde{v}}{\nu z P \xrightarrow{\nu z\tilde{v}\,\overline{a}\langle R\rangle} P'}$$

defining static restriction and extrusion of restricted names, respectively. Note that there is no need to define how a bounded output interacts with input as every τ transition takes place under the restrictions. Also, structural congruence (Definition 2.2) would be extended with the axioms for restriction $\nu z\, \nu w P \equiv \nu w\, \nu z P$ and $\nu z \mathbf{0} \equiv \mathbf{0}$. (In contrast, notice that we do not require the axiom: $\nu z(P \parallel Q) \equiv P \parallel (\nu z Q)$, where z does not occur in P.) We sometimes write $\nu a_1, \ldots, a_n$ to stand for $\nu a_1, \ldots, \nu a_n$.

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) [18,19] to the bisimilarity of some processes. 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 PCP 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 $i_1, \ldots, i_k$, $1 \leq i_j \leq n$ $(j \in 1, \ldots, k)$, such that $u_{i_1}, \ldots, u_{i_k} = l_{i_1}, \cdots, l_{i_k}$. The decision problem is then to determine whether such a solution exists or not.

Having (static) restrictions, we can refine the notation for non-nested replications (Definition 3.2) and define it in the unguarded case:

$$!P \triangleq \boldsymbol{\nu} c\ (Q_c \parallel \overline{c}\langle Q_c\rangle)$$

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). It is easy to see that $!P \xrightarrow{\tau} !P \parallel P$.

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 they non-deterministically 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 3, 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 $\overline{a}\langle P\rangle$ and a_2 as $a(x).\,(x \parallel P)$, where P is the continuation and x does not occur in P; for the lower word the encodings are switched. In Table 3, $[\![a_i, P]\!]_w$ 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.

The whole system P_j is composed by a (replicated) *creator* C_k for each tile T_k, a *starter* S_j that launches the building of a tile composition ending with (u_j, l_j), and an *executor* E. The starter makes the computation begin; creators non-deterministically add their tile to the beginning of the composition. Also non-deterministically, the executor 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 be performed, which in turn, makes action $\overline{success}$ visible. Notice that without synchronizing on b, action $\overline{success}$ could be visible even in the case in which one of the strings is a prefix of the other one.

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 enough.

Theorem 7.2. *Given an instance of PCP and one of its tiles T_j, there is a solution of the instance of PCP ending with T_j iff P_j is not bisimilar to* $!\mathbf{0}$ *according to any complete τ-bisimilarity.*

Proof. We start by proving the left to right implication. Note that $!\mathbf{0}$ has a unique possible computation, that is infinite and includes only τ actions. Let $T_{i_1}, \ldots, T_{i_m}$ be a solution of the instance of the PCP problem such that $T_{i_m} = T_j$. Then P_j can perform the computation described below, which contains the action $\overline{success}$, thus it is not bisimilar to $!\mathbf{0}$. The computation is as follows:

1. $P_j \xrightarrow{\tau}{}^{*} \nu up, a, b.\ S_j \parallel \prod_{h=1,\ldots,m-1} C_{i_h} \parallel \prod C \parallel\ !\prod_k C_k \parallel E = P_1'$, by replication unfolding (the $\prod C$ is the parallel composition of the creators that have been replicated and will not be used);
2. $P_1' \xrightarrow{\tau}{}^{*} \nu up, a, b.\ \overline{up}\langle [\![u, \overline{b}]\!]_u\rangle \parallel \overline{a}\langle [\![l, b.\overline{success}]\!]_l\rangle \parallel \prod C \parallel\ !\prod_k C_k \parallel E = P_2'$, where (u, l) is the solution of the instance of the PCP problem, by making the starter S_j interact with the creators $C_{i_{m-1}}, \ldots, C_{i_1}$;
3. $P_2' \xrightarrow{\tau}\xrightarrow{\tau} \nu up, a, b.\ \prod C \parallel\ !\prod_k C_k \parallel [\![u, \overline{b}]\!]_u \parallel [\![l, b.\overline{success}]\!]_l = P_3'$, by making the starter interact with the executor (note that as every creator starts by an input on up, none of them my be triggered by messages on $\overline{a}$);
4. $P_3' \xrightarrow{\tau}{}^{*} \nu up, a, b.\ \prod C \parallel\ !\prod_k C_k \parallel \overline{b} \parallel b.\overline{success} = P_4'$, by executing the encodings of the two strings, exploiting the fact that they are equal;
5. $P_4' \xrightarrow{\tau} \nu up, a, b.\ \prod C \parallel\ !\prod_k C_k \parallel \overline{success} = P_5'$, by synchronizing on b;
6. $P_5' \xrightarrow{\overline{success}} \nu up, a, b.\ \prod C \parallel\ !\prod_k C_k$.

For the other implication, first notice that all the computations of P_j are infinite since one can always unfold recursion, and action $\overline{success}$ is the only possible visible action. Thus the only possibility for having P_j not bisimilar to $!\mathbf{0}$ is that P_j has a computation executing $\overline{success}$. The only computations that may produce $\overline{success}$ are structured as follows: they build two strings by concatenating the tiles, and then they execute them. One can prove by induction on the minimum length of the strings that if the two strings are different then either their execution gets stuck, or synchronization at b is not possible (this last case occurs if one of the strings is a prefix of the other). Thus, the two strings must be equal and they are the solution of the instance of the PCP problem. □

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 an expression of the form $(x)P$; it is a parametrized process. An abstraction has a functional type. Applying an abstraction $(x)P$ of type $T \to \diamondsuit$ (where $\diamondsuit$ is the type of all processes) to an argument W of type T yields the process $P\{^W\!/x\}$. The argument W can itself be an abstraction; therefore the order of an abstraction, that is, the level of arrow nesting in its type, can be arbitrarily high. The order can also be ω, if there are recursive types. By setting bounds on the order of the types of abstractions, one can define a hierarchy of subcalculi of the higher-order π-calculus [6]; and when this bound is ω, one obtains a calculus capable of representing the π-calculus (for this all operators of the higher-order π-calculus are needed, including full restriction).

Allowing the communication of abstractions, as in the Higher-Order π-calculus, one then also needs to add in the grammar for processes an *application* construct of the form $P_1\langle P_2\rangle$, as a destructor for abstractions. Extensions in the LTS would be as follows. Suppose, as in [21], that *beta-conversion* $\succ$ is the least precongruence on HOcore processes generated by the rule

$$(x)P_1\langle P_2\rangle \succ P_1\{^{P_2}\!/x\}.$$

The LTS could be then extended with a rule

$$\text{Beta}\quad \frac{P \succ P_1' \qquad P_1' \xrightarrow{\alpha} Q}{P \xrightarrow{\alpha} Q}$$

Notice that with these additions, the characterization of bisimilarity as IO bisimilarity still holds. For a HOcore extended with abstractions and applications, $\sim_{IO}^{o}$ is still a congruence and is preserved by substitutions (by straightforward extensions of the proofs of Lemmas 4.9 and 4.10). Note that, however, abstraction application may increase the size of processes. If abstractions are of finite type (i.e., their order is smaller than ω) then only a finite number of such applications is possible, and decidability of bisimilarity is preserved. Decidability fails if the order is ω, intuitively because in this case it is possible to simulate the λ-calculus.

Output prefix. If we add an output prefix construct $\overline{a}\langle P\rangle.\,Q$ to HOcore, then the proof of the characterization as IO bisimilarity breaks and, with it, the proof of decidability. Decidability proofs can however be adjusted by appealing to results on unique decomposition of processes and axiomatization (along the lines of Section 6).

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., [35]) and CCS without restriction and relabeling [36]. 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 bisimilarities); 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 properties (1) or (2) hold (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 analyze 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) worse 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 [35].

Related work. Recent progresses to the behavioral theory of higher-order processes are related to the results of this paper. Some of them, such as environmental bisimulations [22] and the bisimilarities for calculi with passivation constructs [28, 29], have been already mentioned. Koutavas and Hennessy [37] have proposed a first-order behavioral theory for higher-order processes, based on the combination of the principles of environmental bisimulations and the improvements to normal bisimilarity proposed by Jeffrey and Rathke [10]. At the heart of the proposed theory is a novel treatment of name extrusions, which is formalized as an LTS in which configurations not only contain a process and the current knowledge of its environment, but also information on the names that have been extruded by it. As a consequence, the labels of such an LTS have a very simple structure. This is the simplest presentation of an LTS for a higher-order process calculus we are aware of. The weak bisimilarity derived from the proposed LTS is shown to be a congruence, fully abstract with respect to contextual equivalence, and to have a logic characterization using a propositional Hennessy–Milner logic. These results are shown to scale up to (higher-order) languages with distribution, such as the one introduced in [38], for which practical translations into first-order calculi do not exist.

As mentioned in Section 1, in expressiveness studies for higher-order process calculi the usual yardstick for comparison is given by first-order languages such as the π-calculus. A representative work is [21], where a hierarchy of HOπ fragments (obtained by varying the degree of the abstractions allowed) is shown to match the expressiveness of a hierarchy of first-order calculi with only internal mobility. In the same vein is [39], which, inspired by the encoding given in [2], presents an encoding of the π-calculus into Homer, a higher-order process calculus with locations [26]. Such an encoding is exploited in [40] as a way of characterizing finite-control fragments of Homer in which barbed bisimilarity is decidable.

The encoding of Turing complete models (such as Minsky and Random Access Machines, RAMs [41]) is a common proof technique for carrying out expressiveness studies. Our encoding of Minsky machines into HOcore resembles in structure those in [42,43], where RAMs are used to investigate the expressive power of restriction and replication in name-passing calculi, and those in [44], where the impact of restriction and movement on the expressiveness of Ambient calculi is studied. All these encodings share the same guiding principle: representing counting as the nesting of suitable components. Those components are restricted names in CCS [42,43], recursive definitions in π-calculus [42], ambients themselves in Ambient calculus [44], and higher-order messages in our case. Note that by combining our encoding with the one of higher-order π into π-calculus in [6], we obtain an encoding very similar to the one in [42]. Reductions from the PCP to prove undecidability results have been used in other settings. For instance, suitable reductions are used in [45] to show the undecidability of equivalences in timed concurrent constraint languages, and in [46] to show undecidability of the model checking problem for the Ambient calculus without restriction but with replication.

A number of expressiveness and decidability results for variants/extensions of HOcore that complement the results here presented have been reported in [47–49]. In an attempt to understand the source of expressive power in HOcore, [48] studies the fragment of HOcore in which nested higher-order output actions are disallowed. Hence, the encoding of Minsky machines in Section 3 is not expressible in such a fragment. The focus of [48] is in the (un)decidability of *termination* (i.e., the absence of divergent computation) and *convergence* (i.e., the existence of a non-diverging computation) in the mentioned fragment. The main result in [48] is that, in contrast to HOcore, in the fragment without nested higher-order outputs termination becomes decidable, while convergence remains undecidable (as in HOcore). While the latter result is shown by means of a non-deterministic encoding of Minsky machines, the former result is obtained by appealing to the theory of well-structured transition systems, following the approach in [43]. This result is then strengthened in [47, Chapter 5] where it is shown that when the fragment of HOcore considered in [48] is extended with a passivation operator (as in Homer and the Kell calculus [50]) full Turing completeness (as in HOcore) is recovered, and hence termination is undecidable. Finally, [47,49, Chapter 6] studies the expressiveness of (a)synchronous and polyadic communication in the context of *strictly* higher-order process calculi (roughly, variants of HOcore with restriction). Three main results are obtained. First, similarly to first-order calculi, synchronous process passing is shown to be encodable into asynchronous process passing. Then, it is shown that the absence of name passing leads to a hierarchy of higher-order process calculi based on the arity allowed in polyadic communication, thus revealing a striking point of contrast with respect to first-order calculi. Finally, the passing of abstractions is shown to be more expressive than process passing alone.

Acknowledgments

This research was initiated by some remarks and email exchange with Naoki Kobayashi. The authors also benefited from exchanges with Jesús Aranda, Cinzia Di Giusto, Maurizio Gabbrielli, Antonín Kučera, Sergueï Lenglet, Nobuko Yoshida, and Gianluigi Zavattaro, from feedback of the readers of the Moca and Concurrency mailing lists, and from the comments and remarks of the anonymous referees.

Appendix A. Proof of Lemma 3.4

This appendix is devoted to the proof of correctness of the encoding of Minsky machines into HOcore. In what follows we assume a Minsky machine N with instructions $(1 : I_1), \ldots, (n : I_n)$ and with registers $r_0 = m_0$ and $r_1 = m_1$. The encoding of a configuration (i, m_0, m_1) of N is denoted $[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}}$. We use $\longrightarrow^j$ to stand for a sequence of j reductions.

Lemma (Soundness)**.** *Let (i, m_0, m_1) be a configuration of a Minsky machine N. If $(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (i', m_0', m_1')$ then there exist a finite j and a process P such that $[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}} \longrightarrow^j P$ and $P = [\![(i', m_0', m_1')_N]\!]_{\mathsf{M}}$.*

Proof. We proceed by case analysis on the instruction performed by the Minsky machine. Hence, we distinguish three cases corresponding to the behaviors associated to rules M-Jmp, M-Dec, and M-Inc.

Case M-Jmp. We have a Minsky configuration (i, m_0, m_1) with $m_0 = 0$ and $(i : \mathtt{DECJ}(r_0, k))$. By Definition 3.3, its encoding in HOcore is as follows:

$$\begin{aligned}[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}} = \overline{p_i} \parallel [\![r_0 = 0]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \\ [\![(i : \mathtt{DECJ}(r_0, k))]\!]_{\mathsf{M}} \parallel \prod_{l=1,\ldots,n, l\neq i} [\![(l : I_l)]\!]_{\mathsf{M}}\end{aligned}$$

We begin by noting that the program counter p_i is consumed by the encoding of the instruction i. The content of the instruction is thus exposed, and we then have

$$[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}} \longrightarrow [\![r_0 = 0]\!]_{\mathsf{M}} \parallel \widehat{dec_0} \parallel ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \parallel S = P_1$$

where $S = [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{l=1}^{n} [\![(l : I_l)]\!]_{\mathsf{M}}$ stands for the rest of the system. The only transition possible at this point is the behavior selection on dec_0, which yields the following:

$$P_1 \longrightarrow \overline{r_0^0} \parallel \widehat{z_0} \parallel \mathsf{REG}_0 \parallel ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \parallel S = P_2$$

Now there is a synchronization between $\overline{r_0^0}$ and REG_0 for reconstructing the register (with an extra step to unfold the recursion again)

$$P_2 \longrightarrow\longrightarrow \widehat{z_0} \parallel \overline{ack} \parallel \left(inc_0.\,\overline{r_0^{\mathsf{S}}}\langle (\!|\ 0\ |\!)_0 \rangle \,+\, dec_0.\left(\overline{r_0^0} \parallel \widehat{z_0}\right)\right) \parallel \mathsf{REG}_0 \parallel$$
$$ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \parallel S = P_3$$

Once the register has been re-created, register and instruction can now synchronize on *ack*:

$$P_3 \longrightarrow \widehat{z_0} \parallel \left(inc_0.\,\overline{r_0^{\mathsf{S}}}\langle (\!|\ 0\ |\!)_0 \rangle \,+\, dec_0.\left(\overline{r_0^0} \parallel \widehat{z_0}\right)\right) \parallel \mathsf{REG}_0 \parallel$$
$$z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}} \parallel S = P_4$$

At this point, the only possible transition is the behavior selection on z_0, which indicates that the content of r_0 was indeed zero:

$$P_4 \longrightarrow \left(inc_0.\,\overline{r_0^{\mathsf{S}}}\langle (\!|\ 0\ |\!)_0 \rangle \,+\, dec_0.\left(\overline{r_0^0} \parallel \widehat{z_0}\right)\right) \parallel \mathsf{REG}_0 \parallel \overline{p_k} \parallel S = P_5$$

Using the definitions of $[\![\cdot]\!]_{\mathsf{M}}$ and S, and some reordering, we note that P_5 can be equivalently written as

$$P_5 = \overline{p_k} \parallel [\![r_0 = 0]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{l=1}^{n} [\![(l : I_l)]\!]_{\mathsf{M}}$$

which, in turn, corresponds to the encoding of $[\![(k, 0, m_1)_N]\!]_{\mathsf{M}}$, as desired.

Case M-Dec. We have a Minsky configuration (i, m_0, m_1) with $m_0 = c$ (for some $c > 0$) and $(i : \mathtt{DECJ}(r_0, k))$. By Definition 3.3, its encoding in HOcore is as follows:

$$[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}} = \overline{p_i} \parallel [\![r_0 = c]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel$$
$$[\![(i : \mathtt{DECJ}(r_0, k))]\!]_{\mathsf{M}} \parallel \prod_{l=1,\ldots,n,\,l\neq i} [\![(l : I_l)]\!]_{\mathsf{M}}$$

We begin by noting that the program counter p_i is consumed by the encoding of the instruction i. The content of the instruction is thus exposed, and we then have

$$[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}} \longrightarrow [\![r_0 = c]\!]_{\mathsf{M}} \parallel \widehat{\overline{dec_0}} \parallel ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \parallel S = P_1$$

where $S = [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{l=1}^{n} [\![(l : I_l)]\!]_{\mathsf{M}}$ stands for the rest of the system. The only transition possible at this point is the behavior selection on dec_0, which yields the following:

$$P_1 \longrightarrow (\!|\ c-1\ |\!)_0 \parallel \mathsf{REG}_0 \parallel ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \parallel S = P_2$$

It is worth recalling that $(\!|\ c-1\ |\!)_0 = \overline{r_0^{\mathsf{S}}}\langle (\!|\ c-2\ |\!)_0 \rangle \parallel \widehat{n_0}$. Considering this, now there is a synchronization between $\overline{r_0^{\mathsf{S}}}$ and REG_0 for decrementing the value of the register (with an extra step to unfold the recursion again)

$$P_2 \longrightarrow\longrightarrow \widehat{n_0} \parallel \overline{ack} \parallel \left(inc_0.\,\overline{r_0^{\mathsf{S}}}\langle (\!|\ c-1\ |\!)_0 \rangle \,+\, dec_0.\,(\!|\ c-2\ |\!)_0\right) \parallel \mathsf{REG}_0 \parallel$$
$$ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \parallel S = P_3$$

Once the register has been re-created, register and instruction can now synchronize on *ack*:

$$P_3 \longrightarrow \widehat{n_0} \parallel \left(inc_0.\, \overline{r_0^{\mathsf{S}}}\langle (\!| \; c-1 \; |\!)_0\rangle \; + \; dec_0.\, (\!| \; c-2 \; |\!)_0\right) \parallel \mathsf{REG}_0 \parallel$$
$$z_0.\, \overline{p_k} + n_0.\, \overline{p_{i+1}} \parallel S = P_4$$

At this point, the only possible transition is the behavior selection on n_0, which indicates that the content of r_0 was greater than zero:

$$P_4 \longrightarrow \left(inc_0.\, \overline{r_0^{\mathsf{S}}}\langle (\!| \; c-1 \; |\!)_0\rangle \; + \; dec_0.\, (\!| \; c-2 \; |\!)_0\right) \parallel \mathsf{REG}_0 \parallel \overline{p_{i+1}} \parallel S = P_5$$

Using the definitions of $[\![\cdot]\!]_{\mathsf{M}}$ and S, and some reordering, we note that P_5 can be equivalently written as

$$P_5 = \overline{p_{i+1}} \parallel [\![r_0 = c-1]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{l=1}^{n} [\![(l : I_l)]\!]_{\mathsf{M}}$$

which, in turn, corresponds to the encoding of $[\![(i+1, c-1, m_1)_N]\!]_{\mathsf{M}}$, as desired.

Case M-INC. We have a Minsky configuration (i, m_0, m_1) with $(i : \texttt{INC}(r_0))$. Its encoding in HOcore is as follows:

$$[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}} = \overline{p_i} \parallel [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel$$
$$[\![(i : \texttt{INC}(r_0))]\!]_{\mathsf{M}} \parallel \prod_{l=1,\ldots,n, l\neq i} [\![(l : I_l)]\!]_{\mathsf{M}}$$

We begin by noting that the program counter p_i is consumed by the encoding of the instruction i:

$$[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}} \longrightarrow [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel \widehat{inc_0} \parallel ack.\, \overline{p_{i+1}} \parallel S = P_1$$

where $S = [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{l=1}^{n} [\![(l : I_l)]\!]_{\mathsf{M}}$ stands for the rest of the system. The only transition possible at this point is the behavior selection on inc_0. After such a selection we have

$$P_1 \longrightarrow \overline{r_0^{\mathsf{S}}}\langle (\!| \; m_0 \; |\!)_0\rangle \parallel \mathsf{REG}_0 \parallel ack.\, \overline{p_{i+1}} \parallel S = P_2$$

Now there is a synchronization between $\overline{r_0^{\mathsf{S}}}$ and REG_0 for incrementing the value of the register

$$P_2 \longrightarrow\longrightarrow \overline{ack} \parallel \left(inc_0.\, \left(\overline{r_0^{\mathsf{S}}}\langle \overline{r_0^{\mathsf{S}}}\langle (\!| \; m_0 \; |\!)_0\rangle \parallel \widehat{n_0}\rangle\right) \; + \; dec_0.\, ((\!| \; m_0 \; |\!)_0)\right) \parallel \mathsf{REG}_0 \parallel$$
$$ack.\, \overline{p_{i+1}} \parallel S = P_3$$

Once the register has been re-created, a synchronization on ack is possible

$$P_3 \longrightarrow \left(inc_0.\, (\overline{r_0^{\mathsf{S}}}\langle \overline{r_0^{\mathsf{S}}}\langle (\!| \; m_0 \; |\!)_0\rangle \parallel \widehat{n_0}\rangle\right) \; + \; dec_0.\, ((\!| \; m_0 \; |\!)_0)) \parallel \mathsf{REG}_0 \parallel$$
$$\overline{p_{i+1}} \parallel S = P_4$$

Using the definition of $(\!| \; \cdot \; |\!)_j$ we note that P_4 actually corresponds to

$$P_4 = \left(inc_0.\, (\overline{r_0^{\mathsf{S}}}\langle (\!| \; m_0 + 1 \; |\!)_0\rangle \; + \; dec_0.\, ((\!| \; m_0 \; |\!)_0))\right) \parallel \mathsf{REG}_0 \parallel \overline{p_{i+1}} \parallel S$$

which in turn can be written as

$$P_4 = \overline{p_{i+1}} \parallel [\![r_0 = m_0 + 1]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{l=1}^{n} [\![(l : I_l)]\!]_{\mathsf{M}}$$

which corresponds to the encoding of $[\![(i+1, m_0+1, m_1)_N]\!]_{\mathsf{M}}$, as desired. □

Lemma (Completeness). *Let (i, m_0, m_1) be a configuration of a Minsky machine N.*
If $[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}} \longrightarrow P_1$ then

- *if P_1 contain $\widehat{inc_j}$ ($j \in \{0, 1\}$) then the only computation possible is $P_1 \longrightarrow^4 P_J$;*
- *if P_1 contain $\widehat{dec_j}$ ($j \in \{0, 1\}$) then the only computation possible is $P_1 \longrightarrow^5 P_J$.*

In both cases, we have $P_j = [\![(i', m'_0, m'_1)_N]\!]_{\mathsf{M}}$ and $(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (i', m'_0, m'_1)$.

Proof. Consider the reduction $[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}} \longrightarrow P_1$. An analysis of the structure of process $[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}}$ reveals that, in all cases, the only possibility for the first step corresponds to the consumption of the program counter p_i, releasing either $\widehat{inc_j}$ or $\widehat{dec_j}$ ($j \in \{0, 1\}$). This implies that there exists an instruction labeled with i, that can be executed from the configuration (i, m_0, m_1). We proceed by a case analysis on the possible instruction, considering also the fact that the register on which the instruction acts can hold a value equal or greater than zero. In all cases, it can be shown that computation evolves deterministically, until reaching a process in which a new program counter (that is, some $\overline{p_{i'}}$) appears. The program counter $\overline{p_{i'}}$ is inside a process that corresponds to $[\![(i', m'_0, m'_1)_N]\!]_{\mathsf{M}}$, where $(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (i', m'_0, m'_1)$. The analysis follows the same lines as the one reported for the proof of the soundness lemma above, and we omit it. □

Lemma. *Let N be a Minsky machine. We have that $N \not\rightarrow_M$ if and only if $[\![N]\!]_{\mathsf{M}} \not\rightarrow$.*

Proof. Straightforward from the soundness and completeness lemmas given above. □

References

[1] B. Thomsen, A calculus of higher order communicating systems, in: Proceedings of POPL'89, ACM Press, 1989, pp. 143–154.
[2] B. Thomsen, Plain CHOCS: a second generation calculus for higher order processes, Acta Inform. 30 (1) (1993) 1–59.
[3] D. Sangiorgi, Expressing Mobility in Process Algebras: First-order and Higher-order Paradigms, Ph.D. thesis CST-99-93, University of Edinburgh, Department of Computer Science, 1992.
[4] B. Thomsen, Calculi for Higher Order Communicating Systems, Ph.D. thesis, Imperial College, 1990.
[5] X. Xu, On the Bisimulation Theory and Axiomatization of Higher-order Process Calculi, Ph.D. thesis, Shanghai Jiao Tong University, 2007.
[6] D. Sangiorgi, D. Walker, The π-Calculus: A Theory of Mobile Processes, Cambridge University Press, 2001.
[7] I. Phillips, M.G. Vigliotti, Symmetric electoral systems for ambient calculi, Inform. Comput. 206 (1) (2008) 34–72.
[8] D. Gorla, On the relative expressive power of calculi for mobility, Electr. Notes Theor. Comput. Sci. 249 (2009) 269–286.
[9] M. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, 1967.
[10] A. Jeffrey, J. Rathke, Contextual equivalence for higher-order pi-calculus revisited, Log. Methods Comput. Sci. 1 (1) (2005) 1–22.
[11] Z. Cao, More on bisimulations for higher order *pi*-calculus, in: Proceedings of FoSSaCS'06, LNCS, vol. 3921, Springer, 2006, pp. 63–78.
[12] D. Sangiorgi, The lazy lambda calculus in a concurrency scenario, Inform. Comput. 111 (1) (1994) 120–153.
[13] P. Jančar, Undecidability of bisimilarity for Petri nets and some related problems, Theor. Comput. Sci. 148 (2) (1995) 281–301.
[14] P. Schnoebelen, Bisimulation and other undecidable equivalences for lossy channel systems, in: Proceedings of TACS'01, LNCS, vol. 2215, Springer, 2001, pp. 385–399.
[15] A. Dovier, C. Piazza, A. Policriti, An efficient algorithm for computing bisimulation equivalence, Theor. Comput. Sci. 311 (1–3) (2004) 221–256.
[16] R. Milner, F. Moller, Unique decomposition of processes, Theor. Comput. Sci. 107 (2) (1993) 357–363.
[17] D. Hirschkoff, D. Pous, A distribution law for CCS and a new congruence result for the π-calculus, Log. Methods Comput. Sci. 4 (2) (2008).
[18] E.L. Post, A variant of a recursively unsolvable problem, Bull. Am. Math. Soc. 52 (1946) 264–268.
[19] M. Sipser, Introduction to the Theory of Computation, PWS Publishing Company, 2005.
[20] D. Sangiorgi, Bisimulation for higher-order process calculi, Inform. Comput. 131 (2) (1996) 141–178.
[21] D. Sangiorgi, π-Calculus, internal mobility and agent-passing calculi, Theor. Comput. Sci. 167 (2) (1996) 235–274.
[22] D. Sangiorgi, N. Kobayashi, E. Sumii, Environmental bisimulations for higher-order languages, in: Proceedings of LICS'07, IEEE Computer Society, 2007, pp. 293–302.
[23] R. Milner, Communication and Concurrency, International Series in Computer Science, Prentice-Hall, 1989.
[24] D.J. Howe, Proving congruence of bisimulation in functional programming languages, Inform. Comput. 124 (2) (1996) 103–112.
[25] M. Baldamus, T. Frauenstein, Congruence Proofs for Weak Bisimulation Equivalences on Higher Order Process Calculi, Technical Report, Berlin University of Technology, 1995.
[26] M. Bundgaard, J.C. Godskesen, T. Hildebrandt, Bisimulation Congruences for Homer – A Calculus of Higher Order Mobile Embedded Resources, Technical Report TR-2004-52, IT University of Copenhagen, 2004.
[27] J.C. Godskesen, T.T. Hildebrandt, Extending howe's method to early bisimulations for typed mobile embedded resources with local names, in: Proceedings of FSTTCS, Lecture Notes in Computer Science, vol. 3821, Springer, 2005, pp. 140–151.
[28] S. Lenglet, A. Schmitt, J.B. Stefani, Normal bisimulations in calculi with passivation, in: Proceedings of FOSSACS, Lecture Notes in Computer Science, vol. 5504, Springer, 2009, pp. 257–271.
[29] S. Lenglet, A. Schmitt, J.B. Stefani, Howe's method for calculi with passivation, in: Proceedings of CONCUR, Lecture Notes in Computer Science, vol. 5710, Springer, 2009, pp. 448–462.
[30] K. Honda, N. Yoshida, On reduction-based process semantics, Theor. Comput. Sci. 151 (2) (1995) 437–486.
[31] R. Milner, D. Sangiorgi, Barbed bisimulation, in: Proceedings of 19th ICALP, LNCS, vol. 623, Springer-Verlag, 1992, pp. 685–695.
[32] R.M. Amadio, I. Castellani, D. Sangiorgi, On bisimulations for the asynchronous pi-calculus, Theor. Comput. Sci. 195 (2) (1998) 291–324.
[33] F. Moller, Axioms for Concurrency, Ph.D. thesis CST-59-89, University of Edinburgh, Department of Computer Science, 1989.
[34] N.G. De Bruijn, Lambda calculus notation with nameless dummies: A tool for automatic formula manipulation, with application to the Church-Rosser theorem, Indagat. Math. 34 (1972) 381–392.
[35] A. Kučera, P. Jančar, Equivalence-checking on infinite-state systems: techniques and results, TPLP 6 (3) (2006) 227–264.
[36] S. Christensen, Y. Hirshfeld, F. Moller, Decidable subsets of CCS, Comput. J. 37 (4) (1994) 233–242.
[37] V. Koutavas, M. Hennessy, First-order Reasoning for Higher-order Concurrency, Technical Report, Trinity College Dublin, February 2010.
[38] J.L. Vivas, M. Dam, From higher-order pi-calculus to pi-calculus in the presence of static operators, in: Proceedings of CONCUR, Lecture Notes in Computer Science, vol. 1466, Springer, 1998, pp. 115–130.
[39] M. Bundgaard, J.C. Godskesen, T. Hildebrandt, On Encoding the Pi-calculus in Higher-order Calculi, Technical Report TR-2008-106, IT University of Copenhagen, 2008.
[40] M. Bundgaard, J.C. Godskesen, B. Haagensen, H. Hüttel, Decidable fragments of a higher order calculus with locations, Electr. Notes Theor. Comput. Sci. 242 (1) (2009) 113–138.

[41] J.C. Shepherdson, H.E. Sturgis, Computability of recursive functions, J. ACM 10 (2) (1963) 217–255.
[42] N. Busi, M. Gabbrielli, G. Zavattaro, Replication vs. recursive definitions in channel based calculi, in: Proceedings of ICALP, LNCS, vol. 2719, Springer, 2003, pp. 133–144.
[43] N. Busi, M. Gabbrielli, G. Zavattaro, On the expressive power of recursion, replication and iteration in process calculi, Math. Struct. Comput. Sci. 19 (6) (2009) 1191–1222.
[44] N. Busi, G. Zavattaro, On the expressive power of movement and restriction in pure mobile ambients, Theor. Comput. Sci. 322 (3) (2004) 477–515.
[45] M. Nielsen, C. Palamidessi, F.D. Valencia, On the expressive power of temporal concurrent constraint programming languages, in: Proceedings of PPDP, ACM, 2002, pp. 156–167.
[46] W. Charatonik, J.M. Talbot, The decidability of model checking mobile ambients, in: Proceedings of CSL, LNCS, vol. 2142, Springer, 2001, pp. 339–354.
[47] J.A. Pérez, Higher-Order Concurrency: Expressiveness and Decidability Results, Ph.D. thesis, University of Bologna, 2010.
[48] C. Di Giusto, J.A. Pérez, G. Zavattaro, On the expressiveness of forwarding in higher-order communication, in: Proceedings of ICTAC, Lecture Notes in Computer Science, vol. 5684, Springer, 2009, pp. 155–169.
[49] I. Lanese, J.A. Pérez, D. Sangiorgi, A. Schmitt, On the expressiveness of polyadic and synchronous communication in higher-order process calculi, in: Proceedings of ICALP'10, Lecture Notes in Computer Science, vol. 6199, Springer, 2010, pp. 442–453.
[50] A. Schmitt, J.B. Stefani, The kell calculus: a family of higher-order distributed process calculi, in: Global Computing, Lecture Notes in Computer Science, vol. 3267, Springer, 2005, pp. 146–178.