On the Expressiveness of Forwarding in Higher-Order Communication (May 2, 2009)

Cinzia Di Giusto, Jorge A. Pérez, and Gianluigi Zavattaro

Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy

Abstract. In higher-order process calculi the values exchanged in communications may contain processes. There are only two capabilities for received processes: execution and forwarding. Here we propose a limited form of forwarding: output actions can only communicate the parallel composition of statically known closed processes and processes received through previously executed input actions. We study the expressiveness of a higher-order process calculus featuring this style of communication. Our main result shows that in this calculus termination is decidable while convergence is undecidable.

1 Introduction

Higher-order process calculi are calculi in which processes can be communicated. They have been put forward in the early 1990s, with CHOCS [1], Plain CHOCS [2], the Higher-Order π-calculus [3], and others. 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. These calculi are inspired by, and 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.

HOCORE is a core calculus for higher-order concurrency, recently introduced in [4]. It 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 also relevant, as known encodings of fixed-point combinators in higher-order process calculi require the restriction operator. The grammar of HOCORE processes is:

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

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 (the output object) on a; parallel composition allows processes to interact.

Despite this minimality, via a termination preserving encoding of Minsky machines [5], HOCORE was shown to be Turing complete. Therefore, in HOCORE, properties such as *termination* (i.e. non existence of divergent computations) and *convergence* (i.e.

existence of a terminating computation)[1] are both undecidable. In contrast, somewhat surprisingly, strong bisimilarity is decidable, and several sensible bisimilarities in the higher-order setting coincide with it.

In this paper, we shall aim at identifying the intrinsic source of expressive power in HOCORE. A substantial part of the expressive power of a concurrent language comes from the ability of accounting for infinite behavior. In higher-order process calculi there is no explicit operator for such a behavior, as both recursion and replication can be encoded. We then find that infinite behavior resides in the interplay of higher-order communication, in particular, in the ability of *forwarding* a received process within an *arbitrary context*. For instance, consider the process $R = a(x).\, \overline{b}\langle P_x \rangle$ (here P_x stands for a process P with free occurrences of a variable x). Intuitively, R receives a process on name a and forwards it on name b. It is easy to see that since objects in output actions are built following the syntax given by $(*)$, the actual structure of P_x can be fairly complex. One could even "wrap" the process to be received in x using an arbitrary number of k "output layers", i.e., by letting $P_x \equiv \overline{b_1}\langle \overline{b_2}\langle \ldots \overline{b_k}\langle x \rangle\rangle \ldots \rangle$. This *nesting capability* embodies a great deal of the expressiveness of HOCORE: as a matter of fact, the encoding of Minsky machines in [4] depends critically on nesting-based counters. Therefore, investigating suitable limitations to the kind of processes that can be communicated in an output action appears as a legitimate approach for assessing the expressive power of higher-order concurrency.

With the above consideration in mind, in this paper we propose $\text{HO}^{-\mathsf{f}}$, a sublanguage of HOCORE in which output actions are limited so as to rule out the nesting capability. In $\text{HO}^{-\mathsf{f}}$, output actions can communicate the parallel composition of two kinds of objects: (i) statically known closed processes (i.e. that do not contain free variables), and (ii) processes received through previously executed input actions. Hence, the context in which the output action resides can only contribute to communication by "appending" pieces of code that admit no inspection, available in the form of a black-box. More formally, the grammar of $\text{HO}^{-\mathsf{f}}$ processes is that in $(*)$, excepting the production for output actions, which is replaced by the following one:

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

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

The limited style of higher-order communication enforced in $\text{HO}^{-\mathsf{f}}$ is relevant from a pragmatic perspective. In fact, communication in $\text{HO}^{-\mathsf{f}}$ is inspired by those cases in which a process P is communicated in a translated format $[\![P]\!]$, and the translation is not compositional. That is, the cases in which, for any process context C, the translation of $C[P]$ cannot be seen as a function of the translation of P, i.e. there exists no context D such that $[\![C[P]]\!] = D[P]$. This setting can be related to several existing programming scenarios. The simplest example is perhaps mobility of already compiled code, on which it is not possible to apply inverse translations (such as reverse engineering). Other examples include *proof-carrying code* [6] and communication of *obfuscated code*

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

[7]. The former features communication of executable code that comes with a certificate: a recipient can only check the certificate and decide whether to execute the code or not. The latter consists in the communication of source code that is made difficult to understand for, e.g., security/copyright reasons, while preserving its functionality.

The main contribution of the paper is the study of the expressiveness of $\text{HO}^{-\mathsf{f}}$ in terms of decidability of termination and convergence. Our main results are:

1. Similarly as HOCORE, $\text{HO}^{-\mathsf{f}}$ is Turing complete. Therefore, despite the limitation of forward capabilities motivated above, the calculus has a significant expressive power. The result is obtained by exhibiting an encoding of Minsky machines.
2. In sharp contrast with HOCORE, termination in $\text{HO}^{-\mathsf{f}}$ is *decidable*. This result is obtained by appealing to the theory of well-structured transition systems [8], following the approach used in [9].

As for (1), it is worth commenting that the encoding is not *faithful* in the sense that, unlike the encoding of Minsky machines in HOCORE, it may introduce computations which do not correspond to the expected behavior of the modeled machine. Such computations are forced to be infinite and thus regarded as non-halting computations which are therefore ignored. Only the finite computations correspond to those of the encoded Minsky machine. This way, we prove that a Minsky machine terminates if and only if its encoding in $\text{HO}^{-\mathsf{f}}$ converges. Consequently, convergence in $\text{HO}^{-\mathsf{f}}$ is *undecidable*.

As for (2), the use of the theory of well-structured transition systems is certainly not a new approach for obtaining expressiveness results. However, to the best of our knowledge, this is the first time it is applied in the higher-order setting. This is significant because the adaptation to the $\text{HO}^{-\mathsf{f}}$ case is far from trivial. Indeed, as we shall discuss, this approach relies on approximating an upper bound on the depth of the (set of) derivatives of a process. By *depth* of a process we mean its maximal nesting of input/output actions. Notice that, even with the limitation on forwarding enforced by $\text{HO}^{-\mathsf{f}}$, because of the "term copying" feature of higher-order calculi, variable instantiation might lead to a potentially larger process. Hence, finding suitable ways of bounding the set of derivatives of a process is rather challenging and needs care.

Structure of the paper. The syntax and semantics of $\text{HO}^{-\mathsf{f}}$ are introduced in Section 2. The encoding of Minsky machines into $\text{HO}^{-\mathsf{f}}$, and the undecidability of convergence are discussed in Section 3. The decidability of termination is addressed in Section 4. Some final remarks, as well as a review of related work, are included in Section 5.

2 The Calculus

We now introduce the syntax and LTS of $\text{HO}^{-\mathsf{f}}$. 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.

$$
\begin{array}{rll}
P,\, Q ::= & \overline{a}\langle x_1 \parallel \cdots \parallel x_k \parallel P\rangle \quad (\text{with } k \geq 0,\ \mathsf{fv}(P) = \emptyset) & \text{output} \\
\mid & a(x).\, P & \text{input prefix} \\
\mid & P \parallel Q & \text{parallel composition} \\
\mid & x & \text{process variable} \\
\mid & \mathbf{0} & \text{nil}
\end{array}
$$

An input $a(x).\, P$ binds the free occurrences of x in P. We write $\mathsf{fv}(P)$ for the set of free variables in P, and $\mathsf{bv}(P)$ for the bound variables. A process is *closed* if it does not have free variables. We abbreviate $a(x).\, P$, with $x \notin \mathsf{fv}(P)$, as $a.\, P$, $\overline{a}\langle \mathbf{0} \rangle$ as $\overline{a}$, and $P_1 \parallel \ldots \parallel P_k$ as $\prod_{i=1}^{k} P_i$. Hence, an output action can be written as $\overline{a}\langle \prod_{k \in K} x_k \parallel P \rangle$. Similarly, we write $\prod_1^n P$ as an abbreviation for the parallel composition of n copies of P. Further, $P\{{}^{Q}\!/x\}$ denotes the substitution of the free occurrences of x with process Q in P.

Now we describe the Labeled Transition System, which is defined on closed processes. There are three forms of transitions: τ transitions $P \xrightarrow{\tau} P'$; input transitions $P \xrightarrow{a(x)} P'$, meaning that P can receive at a a process that will replace x in the continuation P'; and output transitions $P \xrightarrow{\overline{a}\langle P' \rangle} P''$ meaning that P emits P' at a, and in doing so it evolves to P''. We use α to indicate a generic label of a transition.

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

$$\textsc{Act}1\ \ \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \parallel P_2 \xrightarrow{\alpha} P_1' \parallel P_2} \qquad\qquad \textsc{Tau}1\ \ \frac{P_1 \xrightarrow{\overline{a}\langle P \rangle} P_1' \qquad P_2 \xrightarrow{a(x)} P_2'}{P_1 \parallel P_2 \xrightarrow{\tau} P_1' \parallel P_2'\{{}^{P}\!/x\}}$$

(We have omitted $\textsc{Act}2$ and $\textsc{Tau}2$, the symmetric counterparts of the last two rules.)

Remark 1. Since we consider closed processes, in rule $\textsc{Act}1$, P_2 has no free variables and no side conditions are necessary. As a consequence, alpha-conversion is not needed.

Definition 1. *The* structural congruence *relation is the smallest congruence generated by the following laws:*

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

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

Lemma 1. *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 $P \xrightarrow{\alpha} Q$.

The *alphabet* of an $\textsc{Ho}^{-\mathsf{f}}$ process is defined as follows:

Definition 2 (Alphabet of a process). *Let P be a* $\textsc{Ho}^{-\mathsf{f}}$ *process. The* alphabet of P*, denoted $\mathcal{A}(P)$, is inductively defined as:*

$$\mathcal{A}(\mathbf{0}) = \emptyset \qquad \mathcal{A}(P \parallel Q) = \mathcal{A}(P) \cup \mathcal{A}(Q) \qquad \mathcal{A}(x) = \{x\}$$
$$\mathcal{A}(a(x).\, P) = \{a, x\} \cup \mathcal{A}(P) \qquad \mathcal{A}(\overline{a}\langle P \rangle) = \{a\} \cup \mathcal{A}(P)$$

Proposition 1. *Let P be a* $\textsc{Ho}^{-\mathsf{f}}$ *process. The set $\mathcal{A}(P)$ is finite.*

Proof. Straightforward from Definition 2, exploiting the fact that in $\textsc{Ho}^{-\mathsf{f}}$ alpha-conversion is not necessary (see Remark 1). □

Proposition 2. *Let P and P' be* $\textsc{Ho}^{-\mathsf{f}}$ *processes. If $P \xrightarrow{\alpha} P'$ then $\mathcal{A}(P') \subseteq \mathcal{A}(P)$.*

Proof. We proceed by a case analysis on the rule used to infer $\xrightarrow{\alpha}$. We thus have six cases:

Case **INP** Then $P \equiv a(x).\, P'$. By Definition 2, $\mathcal{A}(P) = \{a, x\} \cup \mathcal{A}(P')$, and the thesis holds.

Case **OUT** Then $P \equiv \overline{a}\langle Q \rangle$. By Definition 2 $\mathcal{A}(P) = \{a\} \cup \mathcal{A}(Q)$ and $\mathcal{A}(P') = \emptyset$, and the thesis holds.

Case **TAU1** Then $P = P_1 \parallel P_2$, $P' = P_1' \parallel P_2'\{R/x\}$, with $P_1 \xrightarrow{\overline{a}\langle R \rangle} P_1'$ and $P_2 \xrightarrow{a(x)} P_2'\{R/x\}$. By Definition 2 we have that $\mathcal{A}(P_1) = \{a\} \cup \mathcal{A}(P_1') \cup \mathcal{A}(R)$ and hence $\mathcal{A}(P_1') \subseteq \mathcal{A}(P_1)$. Also by Definition 2 we have $\mathcal{A}(P_2) = \{a, x\} \cup \mathcal{A}(P_2')$. Now, the process R is closed: therefore, during substitution, no variable can be captured. Hence, α-conversion is not needed, and we have $\mathcal{A}(P_2'\{R/x\}) \subseteq \mathcal{A}(P_2') \cup \mathcal{A}(R)$. The result then follows.

Case **TAU2** Similarly as for TAU1.

Case **ACT1** Then $P \equiv P_1 \parallel P_2$, $P' \equiv P_1' \parallel P_2$, and $P_1 \xrightarrow{\alpha} P_1'$. We then have $\mathcal{A}(P_1') \subseteq \mathcal{A}(P_1)$ by using one of the above cases. By noting that $\mathcal{A}(P_1') \cup \mathcal{A}(P_2) \subseteq \mathcal{A}(P_1) \cup \mathcal{A}(P_2)$, the thesis holds.

Case **ACT2** Similarly as for ACT1.

□

Remark 2. Had we considered open processes, we would have required α-conversion. In such a case, $\mathcal{A}(P_2'\{R/x\}) \subseteq \mathcal{A}(P_2') \cup \mathcal{A}(R)$ would no longer hold. This is because by using α-conversion during substitution some new variables could be added to the alphabet.

Given a process P, its internal runs $P \longrightarrow P_1 \longrightarrow P_2 \longrightarrow \ldots$ are given by sequences of reductions. We denote with $\longrightarrow^*$ the reflexive and transitive closure of $\longrightarrow$; notation $\longrightarrow^j$ is to stand for a sequence of j reductions. We use $P \not\longrightarrow$ to denote that there is no P' such that $P \longrightarrow P'$. Following [9] we now define process convergence and process termination.

Definition 3. *Let P be a* $\text{HO}^{-\mathsf{f}}$ *process.*

1. P converges *iff there exists P' such that $P \longrightarrow^* P'$ and $P' \not\longrightarrow$.*
2. P terminates *iff there exist no $\{P_i\}_{i \in \mathbb{N}}$ such that $P_0 = P$ and $P_j \longrightarrow P_{j+1}$ for any j.*

Observe that process termination implies process convergence while the opposite does not hold.

3 Convergence is Undecidable

In this section we show that $\text{HO}^{-\mathsf{f}}$ is powerful enough to model Minsky machines [5], a Turing complete formalism. We present an encoding that is not *faithful*: unlike the encoding of Minsky machines in HOCORE, it may introduce computations which do not correspond to the expected behavior of the modeled machine. Such computations are forced to be infinite and thus regarded as non-halting computations which are therefore

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

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

$$\text{M-JMP}\ \frac{i : \texttt{DECJ}(r_j, s) \quad m_j = 0}{(i, m_0, m_1) \longrightarrow_{\mathrm{M}} (s, m_0, m_1)}$$

Table 1. Reduction of Minsky machines

ignored. Only finite computations correspond to those of the encoded Minsky machine. More precisely, given a Minsky machine N, its encoding $[\![N]\!]$ has a terminating computation if and only if N terminates. This allows to prove that convergence is undecidable.

We begin by briefly recalling the definition of Minsky machines; we then present the encoding into $\text{HO}^{-\mathsf{f}}$ and discuss its correctness.

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

In the encoding of a Minsky machine into $\text{HO}^{-\mathsf{f}}$ we will find it convenient to have a simple form of guarded replication. This construct can be encoded in $\text{HO}^{-\mathsf{f}}$ as follows.

Input-guarded replication. We follow the standard encoding 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 $\text{HO}^{-\mathsf{f}}$, the encoding is not compositional and replications cannot be nested.

Definition 4. *Assume a fresh name c. The encoding of* input-guarded replication *is as follows:*

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

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

The above encoding preserves termination.

REGISTER r_j $\quad \llbracket r_j = m \rrbracket_{\mathsf{M}} = \prod_1^m \overline{u_j}$

INSTRUCTIONS $(i : I_i)$

$$\begin{array}{ll}
\llbracket (i : \mathtt{INC}(r_j)) \rrbracket_{\mathsf{M}} & = !p_i.\,(\overline{u_j} \parallel set_j(x).\,\overline{set_j}\langle x \parallel \textsc{Inc}_j \rangle \parallel \overline{p_{i+1}}) \\
\llbracket (i : \mathtt{DECJ}(r_j, s)) \rrbracket_{\mathsf{M}} & = \ \ !p_i.\,\overline{m_i} \\
& \quad \parallel !m_i.\,(\overline{loop} \parallel u_j.\,loop.\,set_j(x).\,\overline{set_j}\langle x \parallel \textsc{Dec}_j \rangle \parallel \overline{p_{i+1}}) \\
& \quad \parallel !m_i.\,set_j(x).\,(x \parallel \overline{set_j}\langle \mathbf{0} \rangle \parallel \overline{p_s}))
\end{array}$$

where

$$\textsc{Inc}_j = \overline{loop} \parallel check_j.\,loop \qquad \textsc{Dec}_j = \overline{check_j}$$

Table 2. Encoding of Minsky machines

Lemma 5 (Correctness of $\llbracket\cdot\rrbracket_{\mathsf{i!}}$) *Let P be a* $\textsc{Ho}^{-\mathsf{f}}$ *process with non-nested input-guarded replications.*

- *If $\llbracket P \rrbracket_{\mathsf{i!}} \longrightarrow Q$ then $\exists P'$ such that $P \longrightarrow P'$ and either $\llbracket P' \rrbracket_{\mathsf{i!}} = Q$ or $Q \longrightarrow \llbracket P' \rrbracket_{\mathsf{i!}}$.*
- *If $P \longrightarrow P'$ then either $\llbracket P \rrbracket_{\mathsf{i!}} \longrightarrow \llbracket P' \rrbracket_{\mathsf{i!}}$ or $\llbracket P \rrbracket_{\mathsf{i!}} \longrightarrow\longrightarrow \llbracket P' \rrbracket_{\mathsf{i!}}$.*
- *$\llbracket P \rrbracket_{\mathsf{i!}} \not\rightarrow$ iff $P \not\rightarrow$.*

Proof. By induction on the transitions. □

Encoding Minsky machines into $\textsc{Ho}^{-\mathsf{f}}$. The encoding of Minsky machines into $\textsc{Ho}^{-\mathsf{f}}$ is denoted by $\llbracket\cdot\rrbracket_{\mathsf{M}}$ and presented in Table 2. The encoding of a register r_j that stores the number m is the parallel composition of m copies of the unit process $\overline{u_j}$. To implement the test for zero it is necessary to record how many increments and decrements have been performed on the register r_j. This is done by using a special process $\textsc{Log}_j$, which is communicated back and forth on channel set_j. More precisely, every time an increment instruction occurs, a new copy of the process $\overline{u_j}$ is created, and the process $\textsc{Log}_j$ is updated by adding the process $\textsc{Inc}_j$ in parallel. Similarly for decrements: a copy of $\overline{u_j}$ is consumed and the process $\textsc{Dec}_j$ is added to $\textsc{Log}_j$. As a result, after k increments and l decrements on register r_j, process $\textsc{Log}_j$ is of the form $\textsc{Log}_j = \prod_k \textsc{Inc}_j \parallel \prod_l \textsc{Dec}_j$, which we abbreviate as $\textsc{Log}_j[k, l]$.

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. We already described the behavior of increments. Let us now focus on decrements, the instructions that can introduce divergent —unfaithful— computations. In this case, the process can internally choose either to actually perform a decrement and proceed with the next instruction, or to jump. This can be seen as a guess the process makes on the actual number stored by the register r_j. Therefore, two situations can occur:

1. *The process chooses to decrement r_j.* In this case instruction p_{i+1} is immediately enabled, and the process launches process $\overline{loop}$ and then tries to consume a copy of $\overline{u_j}$. If this operation succeeds (i.e. the content of r_j is greater than 0) then a synchronization with the input on $loop$ that guards the updating of $\textsc{Log}_j$ (represented as an output on name set_j) takes place. Otherwise, the unit process $\overline{u_j}$ could not be consumed (i.e. the content of r_j is zero and the process made a wrong guess).

Process $\overline{loop}$ then synchronizes with the external process $loop.\,\textsc{Div}$, thus spawning a divergent computation.

2. *The process chooses to jump to instruction p_s.* In this case instruction p_s is immediately enabled, and it is necessary to check if the actual value stored by r_j is zero. To do so, the process receives the process $\textsc{Log}_j$ and launches it. If the number of increments is equal to the number of decrements then complementary signals on the name $check_j$ will match each other. In turn, this allows each signal $\overline{loop}$ executed by an $\textsc{Inc}_j$ process to be matched by a complementary one. Otherwise, it is then the case that at least one of those $\overline{loop}$ signals remains active (i.e. the content of the register is not zero); a synchronization with the process $loop.\,\textsc{Div}$ then takes place, and a divergent computation is spawned.

Before executing the instructions, we require both registers in the Minsky machine to be set to zero. This is to guarantee correctness: starting with values different from zero in the registers (without proper initialization of the logs) can lead to inconsistencies. For instance, the test for zero would succeed (i.e. without spawning a divergent computation) even for a register whose value is different than zero. We now define the precise structure of the encoding of configurations.

Definition 6 (Encoding of Configurations). *Let N be a Minsky machine with registers r_0, r_1 and instructions $(1 : I_1), \ldots, (n : I_n)$. For $j \in \{0, 1\}$, suppose fresh, pairwise different names r_j, $p_1, \ldots, p_n$, set_j, loop, $check_j$. Given the encodings in Table 2, we have:*

1. *The initial configuration $(1, 0, 0)$ of N is encoded as:*

$$[\![(1,0,0)]\!]_{\mathsf{M}} ::= \overline{p_1} \parallel \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathsf{M}} \parallel loop.\,\textsc{Div} \parallel \overline{set_0}\langle \mathbf{0} \rangle \parallel \overline{set_1}\langle \mathbf{0} \rangle\ .$$

2. *A configuration (i, m_0, m_1) of N, after k_j increments and l_j decrements of register r_j, is encoded as:*

$$[\![(i, m_0, m_1)]\!]_{\mathsf{M}} = \overline{p_i} \parallel [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathsf{M}} \parallel \\ loop.\,\textsc{Div} \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0] \rangle \parallel \overline{set_1}\langle \textsc{Log}_1[k_1, l_1] \rangle$$

where $\textsc{Div}$ *is a divergent process (e.g. $\overline{w} \parallel !w.\,\overline{w}$).*

We now formalize the following intuition: removing the program counter from the encoding of configurations leads to a stuck process.

Proposition 3. *Let N be a Minsky machine with registers r_0, r_1 and instructions $(1 : I_1), \ldots, (n : I_n)$. Given the encodings in Table 2, let P be defined as:*

$$P = [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathsf{M}} \parallel \\ loop.\,\textsc{Div} \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0] \rangle \parallel \overline{set_1}\langle \textsc{Log}_1[k_1, l_1] \rangle\ .$$

Then $P \not\longrightarrow$.

Proof. Straightforward by the following facts:

1. Processes $[\![r_0 = m_0]\!]_{\mathsf{M}}$, $[\![r_1 = m_1]\!]_{\mathsf{M}}$, $\overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle$, and $\overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle$ are output actions that cannot evolve on their own.
2. Each $[\![(i : I_i)]\!]_{\mathsf{M}}$ is an input-guarded process, waiting for an activation signal on p_i.
3. $loop.\, \textsc{Div}$ is an input-guarded process, and every output on $loop$ appears guarded inside a decrement instruction.

□

The following notation will be useful in proofs.

Notation 1 *Let N be a Minsky machine. The configuration (i, m_0, m_1) of N is annotated as $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$, where, for $j \in \{0,1\}$, k_j and l_j stand for the number of increments and decrements performed on r_j.*

With a little abuse of notation, we use notation $Q \not\longrightarrow$ also for configurations of Minsky machines. We now state that the encoding is correct.

Lemma 2 (Completeness). *Let (i, m_0, m_1) be a configuration of a Minsky machine N. Then, it holds:*

1. *If $(i, m_0, m_1) \not\longrightarrow$ then $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \not\longrightarrow$*
2. *If $(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (i', m_0', m_1')$ then, for some P, $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \longrightarrow^* P \equiv [\![(i', m_0', m_1')]\!]_{\mathsf{M}}$*

Proof. For (1) we have that if $(i, m_0, m_1) \not\longrightarrow$ then, by definition of Minsky machine, the program counter p is set to a non-existent instruction; i.e., for some $i \notin [1..n]$, $p = i$. Therefore, in process $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$ no instruction is guarded by p_i. The thesis then follows by Proposition 3.

For (2) we proceed by a case analysis on the instruction performed by N. Hence, we distinguish three cases corresponding to the behaviors associated to rules M-Jmp, M-Dec, and M-Inc. Without loss of generality we assume instructions on register r_0.

Case M-Inc We have a Minsky machine configuration $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$ with $(i : \mathtt{INC}(r_0))$. Its encoding into $\textsc{Ho}^{-\mathsf{f}}$ is as follows:

$$\begin{aligned}[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathsf{M}} = \overline{p_i} \parallel [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel &\prod_{h=1..n, i\neq h} [\![(h : I_h)]\!]_{\mathsf{M}} \parallel \\ &!p_i.\,(\overline{u_0} \parallel set_0(x).\, \overline{set_0}\langle x \parallel \textsc{Inc}_0\rangle \parallel \overline{p_{i+1}}) \parallel \\ &loop.\,\textsc{Div} \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \parallel \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle\end{aligned}$$

We begin by noting that the program counter p_i is consumed by the encoding of the instruction i. This leaves process $\overline{u_0}$ unguarded; this represents the actual increment. We then have:

$$[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathsf{M}} \longrightarrow \equiv \overline{p_{i+1}} \parallel [\![r_0 = m_0{+}1]\!]_{\mathsf{M}} \parallel set_0(x).\, \overline{set_0}\langle x \parallel \textsc{Inc}_0\rangle \parallel S = T_1$$

where S stands for the rest of the system:

$$S = [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{h=1}^{n} [\![(h : I_h)]\!]_{\mathsf{M}} \parallel loop.\, \textsc{Div} \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \parallel \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle .$$

Now there is a synchronization on set_0 for updating the log of register r_0:

$$\begin{aligned} T_1 \longrightarrow \overline{p_{i+1}} \parallel [\![r_0 = m_0 + 1]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{h=1}^{n} [\![(h : I_h)]\!]_{\mathsf{M}} \parallel \\ loop.\, \textsc{Div} \parallel \overline{set_0}\langle \textsc{Log}_0[k_0+1, l_0]\rangle \parallel \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle = T_2 \,. \end{aligned}$$

We notice that $T_2 \equiv [\![(i+1, m_0 + 1^{k_0+1,l_0}, m_1^{k_1,l_1})]\!]_{\mathsf{M}}$, as desired.

Case M-Dec We have a Minsky machine configuration $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$ $(r_0 > 0)$ with $(i : \texttt{DECJ}(r_0, s))$. By definition, its encoding into $\textsc{Ho}^{-\mathsf{f}}$ is as follows:

$$\begin{aligned} [\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathsf{M}} = \overline{p_i} \parallel [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{h=1..n, i\neq h} [\![(h : I_h)]\!]_{\mathsf{M}} \parallel \\ !p_i.\, \overline{m_i} \parallel !m_i.\, (\overline{loop} \parallel u_0.\, loop.\, set_0(x).\, \overline{set_0}\langle x \parallel \textsc{Dec}_0\rangle \parallel \overline{p_{i+1}}) \parallel \\ !m_i.\, set_0(x).\, (x \parallel \overline{set_0}\langle \mathbf{0}\rangle \parallel \overline{p_s}) \parallel \\ loop.\, \textsc{Div} \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \parallel \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle \end{aligned}$$

The program counter is consumed by the encoding of the instruction i:

$$\begin{aligned} [\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathsf{M}} \longrightarrow [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel \overline{m_i} \parallel \\ !m_i.\, (\overline{loop} \parallel u_0.\, loop.\, set_0(x).\, \overline{set_0}\langle x \parallel \textsc{Dec}_0\rangle \parallel \overline{p_{i+1}}) \parallel \\ !m_i.\, set_0(x).\, (x \parallel \overline{set_0}\langle \mathbf{0}\rangle \parallel \overline{p_s}) \parallel S = T_1 \end{aligned}$$

where S stands for the rest of the system:

$$S = [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{h=1}^{n} [\![(h : I_h)]\!]_{\mathsf{M}} \parallel loop.\, \textsc{Div} \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \parallel \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle .$$

There is an internal choice on the name m_i. Let us suppose T_1 makes the right guess, i.e., that process $\overline{m_i}$ synchronizes with the first input-guarded process. We then have:

$$T_1 \longrightarrow [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel \overline{loop} \parallel u_0.\, loop.\, set_0(x).\, \overline{set_0}\langle x \parallel \textsc{Dec}_0\rangle \parallel \overline{p_{i+1}} \parallel S' = T_2 \,.$$

where S' is the rest of the system, that now contains the input-guarded process that was not involved in the synchronization. Such a process is, however, stuck:

$$S' = S \parallel !m_i.\, set_0(x).\, (x \parallel \overline{set_0}\langle \mathbf{0}\rangle \parallel \overline{p_s})$$

Since the content of r_0 is greater than 0, a synchronization on u_0 takes place, and the value of r_0 decreases. Immediately after, there is also a synchronization on $loop$. More precisely, we have

$$T_2 \longrightarrow^2 [\![r_0 = m_0 - 1]\!]_{\mathsf{M}} \parallel set_0(x).\, \overline{set_0}\langle x \parallel \textsc{Dec}_0 \rangle \parallel \overline{p_{i+1}} \parallel S' = T_3\,.$$

Now the updating of the log associated to r_0 can take place, and a synchronization on set_0 is performed:

$$\begin{aligned} T_3 \longrightarrow\ & \overline{p_{i+1}} \parallel [\![r_0 = m_0 - 1]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{h=1}^{n} [\![(h : I_h)]\!]_{\mathsf{M}} \parallel \\ & loop.\, \textsc{Div} \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0 + 1] \rangle \parallel \overline{set_1}\langle \textsc{Log}_1[k_1, l_1] \rangle = T_4\,. \end{aligned}$$

Clearly, $T_4 \equiv [\![(i+1, m_0 - 1^{k_0, l_0+1}, m_1^{k_1,l_1})]\!]_{\mathsf{M}}$, as desired.

Case M-Jmp This case is similar to the previous one. We have a Minsky machine configuration $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$ with $(i : \mathtt{DECJ}(r_0, s))$. Since $m_0 = 0$ we have that $k_0 = l_0$, and we proceed exactly as in the previous case. After synchronizing on p_i and spawning a new copy of (the encoding of) the instruction i, the process evolves to T_1. Once in T_1 there is an internal choice on the name m_i. Again, let us suppose T_1 makes the right guess, which in this case corresponds to the synchronization of $\overline{m_i}$ and the second input-guarded process. We then have

$$T_1 \longrightarrow [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel set_0(x).\, (x \parallel \overline{set_0}\langle \mathbf{0} \rangle \parallel \overline{p_s}) \parallel S' = T_2\,.$$

where S' is the rest of the system, that is composed of S (as defined in the previous case) and of the input-guarded process on m_i that was not involved in the synchronization. Such a process is, however, stuck:

$$S' = S \parallel !m_i.\, (\overline{loop} \parallel u_0.\, loop.\, set_0(x).\, \overline{set_0}\langle x \parallel \textsc{Dec}_0 \rangle \parallel \overline{p_{i+1}})$$

Now there is a synchronization on set_0, and the log becomes unguarded:

$$\begin{aligned} T_2 \longrightarrow \equiv\ & \overline{p_s} \parallel [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{h=1}^{n} [\![(h : I_h)]\!]_{\mathsf{M}} \parallel \\ & loop.\, \textsc{Div} \parallel \prod_{k_0} \textsc{Inc}_0 \parallel \prod_{l_0} \textsc{Dec}_0 \parallel \overline{set_0}\langle \mathbf{0} \rangle \parallel \overline{set_1}\langle \textsc{Log}_1[k_1, l_1] \rangle = T_3\,. \end{aligned}$$

Recall that $k_0 = l_0$. Starting in T_3, we have that $2 \cdot k_0$ reductions take place: these are the interactions between an $\textsc{Inc}_0$ and a corresponding $\textsc{Dec}_0$. Half of these interactions correspond to synchronizations on $check_0$, whereas the rest are synchronizations on $loop$. All of these procesess are consumed. We then have that there exists a T_4 such that (i) $T_3 \longrightarrow^{2 \cdot k_0} T_4$ and (ii) $T_4 \equiv [\![(s, m_0^{0,0}, m_1^{k_1,l_1})]\!]_{\mathsf{M}}$, as wanted.

□

Lemma 3 (Soundness). *Let (i, m_0, m_1) be a configuration of a Minsky machine N. Given $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$, for some $n > 0$ and process $P \in \mathrm{HO}^{-\mathsf{f}}$, we have that:*

1. $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \longrightarrow^n P$ *and either:*
   – $P \equiv [\![(i', m'_0, m'_1)]\!]_{\mathsf{M}}$ *and* $(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (i', m'_0, m'_1)$*, or*
   – P *is a divergent process.*

2. For all $0 \leq m < n$*, if* $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \longrightarrow^m P$ *then, for some* P'*,* $P \longrightarrow P'$.

3. If $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \not\rightarrow$ *then* $(i, m_0, m_1) \not\rightarrow$.

Proof. For (1), since $n > 0$, in all cases there is at least one reduction from $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$. An analysis of the structure of process $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$ reveals that, in all cases, the first step corresponds to the consumption of the program counter p_i. 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. We exploit the analysis reported for the proof of Lemma 2(2):

Case $i : \mathtt{INC}(r_0)$ Then the process evolves deterministically to $P \equiv [\![(i + 1, m_0 + 1, m_1)]\!]_{\mathsf{M}}$ in $n = 2$ reductions.

Case $i : \mathtt{DEC}(r_0, s)$ **with** $r_0 > 0$ Then the process evolves non-deterministically to one of the following cases:
1. $P \equiv [\![(i + 1, m_0 - 1, m_1)]\!]_{\mathsf{M}}$, in $n = 5$ reductions;
2. $P = \textsc{Div}$, in $n = 4$ reductions. This is the case when the process makes a wrong guess on the content of the register (i.e. it assumes that $r_0 = 0$.)

Case $i : \mathtt{DEC}(r_0, s)$ **with** $r_0 = 0$ Then the process evolves non-deterministically to one of the following cases (we use k and l to denote the number of increments and decrements on r_0, respectively):
1. $P \equiv [\![(s, m_0, m_1)]\!]_{\mathsf{M}}$, in $n = 3 + k + l$ reductions;
2. $P = \textsc{Div}$, in $n = 3 + k + l + 1$ reductions. This is the case when the process makes a wrong guess on the content of the register (i.e. it assumes that $r_0 > 0$.)

Notice that statement (2) follows easily from the above analysis.

As for (3), using Proposition 3 we know that if $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \not\rightarrow$ then it is because p_i is not enabling any instruction. Hence, $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$ corresponds to the encoding of a halting instruction and we have that $(i, m_0, m_1) \not\rightarrow$, as desired. □

Summarizing Lemmata 2 and 3 we have the following:

Theorem 1. *Let* N *be a Minsky machine with registers* $r_0 = m_0$*,* $r_1 = m_1$*, instructions* $(1 : I_1), \ldots, (n : I_n)$ *and configuration* (i, m_0, m_1)*. Let* P *be the process* $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$ *then* (i, m_0, m_1) *terminates if and only if* P *converges.*

As a consequence of the results above we have that convergence is undecidable.

Corollary 1. *Convergence is undecidable in* $\textsc{Ho}^{-\mathsf{f}}$.

4 Termination is Decidable

In this section we prove that termination is decidable for $\textsc{Ho}^{-\mathsf{f}}$ processes. As hinted at in the introduction, this is in sharp contrast with the analogous result for HOCORE. The proof appeals to the theory of well-structured transition systems, whose main definitions and results we summarize next.

Well-Structured Transition Systems. The following results and definitions are from [8], unless differently specified. Recall that a *quasi-order* (or, equivalently, preorder) is a reflexive and transitive relation.

Definition 7 (Well-quasi-order). *A* well-quasi-order *(wqo) is a quasi-order $\leq$ over a set X such that, for any infinite sequence $x_0, x_1, x_2 \ldots \in X$, there exist indexes $i < j$ such that $x_i \leq x_j$.*

Note that if $\leq$ is a wqo then any infinite sequence $x_0, x_1, x_2, \ldots$ contains an infinite increasing subsequence $x_{i_0}, x_{i_1}, x_{i_2}, \ldots$ (with $i_0 < i_1 < i_2 < \ldots$). Thus well-quasi-orders exclude the possibility of having infinite strictly decreasing sequences.

We also need a definition for (finitely branching) transition systems. This can be given as follows. Here and in the following $\longrightarrow^*$ denotes the reflexive and transitive closure of the relation $\longrightarrow$.

Definition 8 (Transition system). *A* transition system *is a structure $TS = (S, \longrightarrow)$, where S is a set of states and $\longrightarrow \subseteq S \times S$ is a set of transitions. We define $Succ(s)$ as the set $\{s' \in S \mid s \longrightarrow s'\}$ of immediate successors of S. We say that TS is finitely branching if, for each $s \in S$, $Succ(s)$ is finite.*

Fact 1 *The LTS for* $\textrm{Ho}^{-\mathsf{f}}$ *given in Section 2 is finitely branching.*

The function $Succ$ will be used also on sets by assuming that in this case they are defined by the point-wise extension of the above definitions.

The key tool to decide several properties of computations is the notion of well-structured transition system. This is a transition system equipped with a well-quasi-order on states which is (upward) compatible with the transition relation. Here we will use a strong version of compatibility, hence the following definition.

Definition 9 (Well-structured transition system). *A* well-structured transition system with strong compatibility *is a transition system $TS = (S, \longrightarrow)$, equipped with a quasi-order $\leq$ on S, such that the two following conditions hold:*

1. *$\leq$ is a well-quasi-order;*
2. *$\leq$ is strongly (upward) compatible with $\longrightarrow$, that is, for all $s_1 \leq t_1$ and all transitions $s_1 \longrightarrow s_2$, there exists a state t_2 such that $t_1 \longrightarrow t_2$ and $s_2 \leq t_2$ holds.*

The following theorem is a special case of Theorem 4.6 in [8] and will be used to obtain our decidability result.

Theorem 2. *Let $TS = (S, \longrightarrow, \leq)$ be a finitely branching, well-structured transition system with strong compatibility, decidable $\leq$ and computable $Succ$. Then the existence of an infinite computation starting from a state $s \in S$ is decidable.*

We will also need a result due to Higman [10] which allows to extend a well-quasi-order from a set S to the set of the finite sequences on S. To be more precise, given a set S let use denote by S^* the set of finite sequences built by using elements in S. We can define a quasi-order on S^* as follows.

Definition 10. *Let S be a set and $\leq$ a quasi-order over S. The relation $\leq_*$ over S^* is defined as follows. Let $t, u \in S^*$, with $t = t_1 t_2 \dots t_m$ and $u = u_1 u_2 \dots u_n$. We have that $t \leq_* u$ iff there exists an injection f from $\{1, 2, \dots m\}$ to $\{1, 2, \dots n\}$ such that $t_i \leq u_{f(i)}$ and $i \leq f(i)$ for $i = 1, \dots, m$.*

The relation $\leq_*$ is clearly a quasi-order over S^*. It is also a wqo, since we have the following result.

Lemma 4 (Higman [10]). *Let S be a set and $\leq$ a wqo over S. Then $\leq_*$ is a wqo over S^*.*

Finally we will use also the following proposition, whose proof is immediate.

Proposition 4. *Let S be a finite set. Then the equality is a wqo over S.*

Termination is Decidable in $\text{HO}^{-\mathsf{f}}$. Here we prove that termination is decidable in $\text{HO}^{-\mathsf{f}}$. The crux of the proof consists in finding an upper bound for a process and its derivatives. This is possible in $\text{HO}^{-\mathsf{f}}$ because of the limited structure allowed in output actions.

We begin by defining a notion of normal form for $\text{HO}^{-\mathsf{f}}$ processes. We then characterize an upper bound for the derivatives of a given process, and we define an ordering over them. This ordering is then shown to be a wqo that is strongly compatible with respect to the LTS of $\text{HO}^{-\mathsf{f}}$ given in Section 2. The decidability result is then obtained by resorting to the results from [8] reported before.

Definition 11 (Normal Form). *Let $P \in \text{HO}^{-\mathsf{f}}$. P is in* normal form *iff*

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

where each P_i and P'_j are in normal form.

Lemma 5. *Every process $P \in \text{HO}^{-\mathsf{f}}$ is structurally congruent to a normal form.*

Proof. By induction on the structure of P. The base cases are when $P = \mathbf{0}$ and when $P = x$, and are immediate. Cases $P = \overline{a}\langle Q \rangle$ and $P = a(x).\, Q$ follow by applying the inductive hypothesis on Q. For the case $P = P_1 \parallel P_2$, we apply the inductive hypothesis twice and we obtain that

$$P_1 \equiv \prod_{k=1}^{l} x_k \parallel \prod_{i=1}^{m} a_i(y_i).\, P_i \parallel \prod_{j=1}^{n} \overline{b_j}\langle P_j \rangle \text{ and } P_2 \equiv \prod_{k=1}^{l'} x_k \parallel \prod_{i=1}^{m'} a'_i(y'_i).\, P'_i \parallel \prod_{j=1}^{n'} \overline{b'_j}\langle P'_j \rangle\,.$$

It is then easy to see that $P_1 \parallel P_2$ is structurally congruent to a normal form, as desired. □

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

Definition 12 (Relation $\preceq$). *Let $P, Q \in \textsc{Ho}^{-\mathsf{f}}$. We write $P \preceq Q$ iff there exist $x_1 \dots x_l$, $P_1 \dots P_m$, $P'_1 \dots P'_n$, $Q_1 \dots Q_m$, $Q'_1 \dots Q'_n$, and R such that*

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

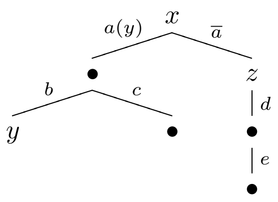
with $P_i \preceq Q_i$ and $P'_j \preceq Q'_j$, for $i \in [1..m]$ and $j \in [1..n]$.

The normal form of a process can be intuitively represented in a tree-like manner. More precisely, given the process in normal form

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

we shall decree its associated tree to have a root node labeled $x_1, \dots, x_k$. This root node has $m + n$ children, corresponding to the the trees associated to processes $P_1, \dots, P_m$ and $P'_1, \dots, P'_m$; the outgoing edges connecting the root node and the children are labeled $a_1(y_1), \dots, a_m(y_m)$ and $\overline{b_1}, \dots, \overline{b_n}$.

Example 1. Process $P = x \parallel a(y).\, (b.\, y \parallel c) \parallel \overline{a}\langle z \parallel d.\, e \rangle$ has the following tree representation:



The intuition given before on the tree representation of a process in normal form will be useful now to reason about the structure of $\textsc{Ho}^{-\mathsf{f}}$ terms. We begin by defining the *depth* of a process. Notice that the depth a process corresponds to the maximum depth of its tree representation.

Definition 13 (Depth). *Let $P = \prod_{k=1}^{l} x_k \parallel \prod_{i=1}^{m} a_i(y_i).\, P_i \parallel \prod_{j=1}^{n} \overline{b_j}\langle P'_j \rangle$ be a $\textsc{Ho}^{-\mathsf{f}}$ process in normal form. The* depth *of P is given by*

$$\mathsf{depth}(P) = \max\{1 + \mathsf{depth}(P_i), 1 + \mathsf{depth}(P'_j) \mid i \in [1..m] \wedge j \in [1..n]\}.$$

Given a natural number n and a process P, the set $\mathcal{P}_{P,n}$ contains all those processes in normal form that (i) can be built using the alphabet of P and (ii) whose depth is at most n.

Definition 14. *Let n be a natural number and $P \in \textsc{Ho}^{-\mathsf{f}}$. We define the set $\mathcal{P}_{P,n}$ as follows:*

$$\begin{array}{rl} \mathcal{P}_{P,n} = \{Q \mid & Q \equiv \prod_{k \in K} x_k \parallel \prod_{i \in I} a_i(y_i).\, Q_i \parallel \prod_{j \in J} \overline{b_j}\langle Q'_j \rangle \\ & \wedge\ \mathcal{A}(Q) \subseteq \mathcal{A}(P) \\ & \wedge\ Q_i, Q'_j \in \mathcal{P}_{P,n-1}\ \forall i \in I, j \in J\} \end{array}$$

where $\mathcal{P}_{P,0}$ contains processes that are built out only of variables in $\mathcal{A}(P)$.

As it will be shown later, the set of all derivatives of P is a subset of $\mathcal{P}_{P,2\cdot\mathsf{depth}(P)}$.

When compared to processes in languages such as Milner's CCS, higher order processes have a more complex structure. This is because, by virtue of reductions, an arbitrary process can take the place of possibly several occurrences of a single variable. As a consequence, the depth of (the syntax tree of) a process cannot be determined (or even approximated) before its execution: it can vary arbitrarily along reductions. Crucially, in $\mathrm{HO}^{-\mathsf{f}}$ it is possible to bound such a depth. Our approach is the following: rather than solely depending on the depth of a process, we define measures on the relative position of variables within a process. Informally speaking, such a position will be determined by the number of prefixes that guard a variable. Notice that since variables are allowed only at the top level of the output objects, the distance of such variables remains invariant during reductions. This allows to obtain a bound on the structure for $\mathrm{HO}^{-\mathsf{f}}$ processes. Finally, it is worth stressing that even if the same notions of normal form, depth, and distance can be defined for HOCORE, a finite upper bound for such a language does not exist.

We now define the maximum distance between a variable and its binder.

Definition 15. *Let $P = \prod_{k\in K} x_k \parallel \prod_{i\in I} a_i(y_i).\,P_i \parallel \prod_{j\in J} \overline{b_j}\langle P'_j\rangle$ be a $\mathrm{HO}^{-\mathsf{f}}$ process in normal form. We define the* maximum distance of P *as:*

$$\mathsf{maxDistance}(P) = \max\{\mathsf{maxDist}_{y_i}(P_i), \mathsf{maxDistance}(P_i), \mathsf{maxDistance}(P'_j) \mid i \in I, j \in J\}$$

where

$$\mathsf{maxDist}_x(P) = \begin{cases} 1 & \text{if } P = x, \\ 1 + \mathsf{maxDist}_x(P_z) & \text{if } P = a(z).\,P_z \wedge x \neq z, \\ 1 + \mathsf{maxDist}_x(P') & \text{if } P = \overline{a}\langle P'\rangle, \\ \max\{\mathsf{maxDist}_x(R), \mathsf{maxDist}_x(Q)\} & \text{if } P = R \parallel Q, \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 6 (Properties of maxDistance). *Let P be a $\mathrm{HO}^{-\mathsf{f}}$ process. It holds that:*

1. $\mathsf{maxDistance}(P) \leq \mathsf{depth}(P)$
2. *For every Q such that $P \xrightarrow{\alpha} Q$,* $\mathsf{maxDistance}(Q) \leq \mathsf{maxDistance}(P)$.

Proof. Part (1) is immediate from Definitions 13 and 15. Part (2) follows by a case analysis on the rule used to infer $\xrightarrow{\alpha}$. We focus in the case TAU1: the other cases are similar or simpler. We then have that $P \equiv \overline{a}\langle S\rangle \parallel a(x).\,R \parallel T$ and $Q \equiv R\{S/x\} \parallel T$. Applying Definition 15 in both processes, we obtain

$$\begin{aligned} \mathsf{maxDistance}(P) &= \max\{\mathsf{maxDistance}(S), \mathsf{maxDist}_x(R), \\ &\qquad \mathsf{maxDistance}(R), \mathsf{maxDistance}(T)\} \\ \mathsf{maxDistance}(Q) &= \max\{\mathsf{maxDistance}(R\{S/x\}), \mathsf{maxDistance}(T)\}\,. \end{aligned}$$

We can thus disregard the contribution of $\mathsf{maxDistance}(T)$, since it does not participate in the synchronization. We then focus on determining $\mathsf{maxDistance}(R\{S/x\})$. We begin by recalling that by the syntax of $\text{Ho}^{-\mathsf{f}}$, $S \equiv x_1 \parallel \cdots \parallel x_k \parallel S'$, where S' is a closed process. The free variables in S can affect $\mathsf{maxDistance}(R\{{}^{S}\!/x\})$ in essentially two ways:

1. *The free variables of S do not get captured by a binder in R.* In this case, they do not contribute to $\mathsf{maxDistance}(R\{{}^{S}\!/x\})$, and we have that

$$\mathsf{maxDistance}(R\{{}^{S}\!/x\}) = \max\{\mathsf{maxDistance}(R), \mathsf{maxDistance}(S)\}$$

   and the thesis holds.
2. *Some of the free variables of S get captured by a binder in R.* In this case we find it convenient to appeal to the tree representations of R and S, denoted T_R and T_S, respectively. Notice that T_S is a tree where the root node is labeled with $x_1, \ldots, x_k$ and whose only descendant is T'_S, the tree representation of S'. Consider now T_Q, the tree representation of $R\{{}^{S}\!/x\}$: it corresponds to the tree T_R in which all occurrences of x in the nodes of T_R have been replaced with T_S. Crucially, the height of $x_1, \ldots, x_k$ is exactly the same height of x, so the distance with respect their binder does not increase. Also, since S' does not contain free variables, the contribution of $\mathsf{maxDistance}(S')$ to P remains invariant in Q. We then conclude that the thesis holds also in this case.

□

We now define the maximum depth of processes that can be communicated. Notice that the continuations of inputs are considered as they could become communication objects themselves along reductions:

Definition 16. *Let $P = \prod_{k \in K} x_k \parallel \prod_{i \in I} a_i(y_i).\, P_i \parallel \prod_{j \in J} \overline{b_j}\langle P'_j \rangle$ be a $\text{Ho}^{-\mathsf{f}}$ process in normal form. We define the* maximum depth *of a process that can be communicated (*$\mathsf{maxDepCom}(P)$*) in P as:*

$$\mathsf{maxDepCom}(P) = \max\{\mathsf{maxDepCom}(P_i), \mathsf{depth}(P'_j) \mid i \in I, j \in J\}\ .$$

Lemma 7 (Properties of maxDepCom). *Let P be a $\text{Ho}^{-\mathsf{f}}$ process. It holds that:*

1. $\mathsf{maxDepCom}(P) \leq \mathsf{depth}(P)$
2. *For every Q such that $P \xrightarrow{\alpha} Q$,* $\mathsf{maxDepCom}(Q) \leq \mathsf{maxDepCom}(P)$.

Proof. Part (1) is immediate from Definitions 13 and 16. Part (2) follows by a case analysis on the rule used to infer $\xrightarrow{\alpha}$. We focus in the case TAU1: the other cases are similar or simpler. We then have that $P \equiv \overline{a}\langle S \rangle \parallel a(x).\, R \parallel T$ and $Q \equiv R\{{}^{S}\!/x\} \parallel T$. Applying Definition 15 in both processes, we obtain

$$\begin{aligned}\mathsf{maxDepCom}(P) &= \max\{\mathsf{maxDepCom}(T), \mathsf{maxDepCom}(S), \mathsf{depth}(S)\}\\ \mathsf{maxDepCom}(Q) &= \max\{\mathsf{maxDepCom}(T), \mathsf{maxDepCom}(R\{{}^{S}\!/x\})\}\ .\end{aligned}$$

We now focus on analyzing the influence a substitution has on communicated objects. More precisely, since variables can occur in output objects, we analyze whether x occurs free in some communication object in R. We thus have two cases:

1. *There are no output objects with free occurrences of x.* Then S will only occur at the top level in $R\{S/x\}$. Since $\mathsf{depth}(S)$ was already taken into account when determining $\mathsf{maxDepCom}(P)$, we then have that $\mathsf{maxDepCom}(R\{S/x\}) \leq \mathsf{maxDepCom}(P)$, and the thesis holds.
2. *Some output objects have free occurrences of x.* Then, there exists a process $P_x = x \parallel x_1 \parallel \cdots \parallel x_k \parallel S'$ so that an output action $\overline{b}\langle P_x \rangle$ occurs in R. Recall that by definition of $\mathrm{Ho}^{-\mathsf{f}}$, S' is a closed process. Therefore, the process $\overline{b}\langle P_x\{S/x\}\rangle$ occurs in $R\{S/x\}$. Clearly, an eventual increase of $\mathsf{maxDepCom}(Q)$ depends on the depth of $P_x\{S/x\}$. We have that $\mathsf{depth}(P_x\{S/x\}) = \max(\mathsf{depth}(S), \mathsf{depth}(S'))$. Since both $\mathsf{depth}(S)$ and $\mathsf{depth}(S')$ were considered when determining $\mathsf{maxDepCom}(P)$, we conclude that $\mathsf{maxDepCom}(R\{S/x\})$ can be at most equal to $\mathsf{maxDepCom}(P)$, and so the thesis holds.

□

Notation 2 *We use $P \xrightarrow{\widetilde{\alpha}} P'$ if, for some $n \geq 0$, there exist $\alpha_1, \ldots, \alpha_n$ such that $P \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} P'$.*

Generalizing Lemmata 6 and 7 we obtain:

Corollary 2. *Let P be a $\mathrm{Ho}^{-\mathsf{f}}$ process. For every Q such that $P \xrightarrow{\widetilde{\alpha}} Q$, it holds that:*

1. $\mathsf{maxDistance}(Q) \leq \mathsf{depth}(P)$
2. $\mathsf{maxDepCom}(Q) \leq \mathsf{depth}(P)$.

We are interested in characterizing the derivatives of a given process P. We shall show that they are over-approximated by means of the set $\mathcal{P}_{P,2\cdot\mathsf{depth}(P)}$. Over such an approximation we will investigate the properties of the relation $\preceq$; such properties will then hold also for the set of derivatives.

Definition 17. *Let $P \in \mathrm{Ho}^{-\mathsf{f}}$. Then we define* $\mathsf{Deriv}(P) = \{Q \mid P \longrightarrow^{*} Q\}$

The following results hold because of the limitations we have imposed on the output actions for $\mathrm{Ho}^{-\mathsf{f}}$ processes.

Lemma 8. *Let P be an $\mathrm{Ho}^{-\mathsf{f}}$ process. $P \in \mathcal{P}_{P,n}$ if and only if* $\mathsf{depth}(P) \leq n$.

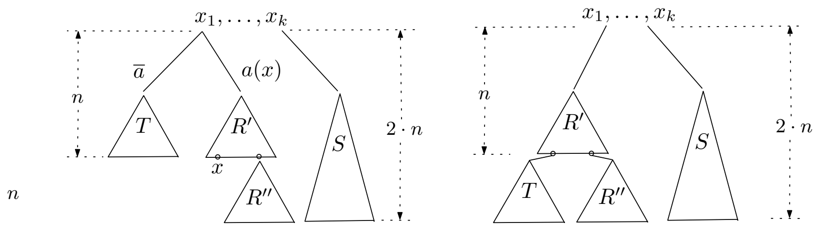
Proof. The "if" direction is straightforward by definition of $\mathcal{P}_{P,n}$ (Definition 14).

For the "only if" direction we proceed by induction on n. If $n = 0$ then $P = \mathbf{0}$ or $P = x_1 \parallel \cdots \parallel x_k$. In both cases, P is easily seen to be in $\mathcal{P}_{P,0}$. If $n > 0$ then

$$P = \prod_{k \in K} x_k \parallel \prod_{i \in I} a_i(y_i).\, P_i \parallel \prod_{j \in J} \overline{b_j}\langle P'_j \rangle$$

where, for every $i \in I$ and $j \in J$, both $\mathsf{depth}(P_i) \leq \mathsf{depth}(P) \leq n - 1$ and $\mathsf{depth}(P'_j) \leq \mathsf{depth}(P) \leq n-1$. By inductive hypothesis, each P_i and P'_j is in $\mathcal{P}_{P,n-1}$. Then, by Definition 14, $P \in \mathcal{P}_{P,n}$ and we are done. □

Proposition 5. *Let P be a $\mathrm{Ho}^{-\mathsf{f}}$ process. Suppose, for some n, that $P \in \mathcal{P}_{P,n}$. For every Q such that $P \xrightarrow{\alpha} Q$, it holds that $Q \in \mathcal{P}_{P,2\cdot n}$.*



(a) Tree representation of R (b) Tree representation of Q

Fig. 1. Tree representation of a reduction $R \longrightarrow Q$, as in the proof of Lemma 9.

Proof. We proceed by case analysis on the rule used to infer $\xrightarrow{\alpha}$. We focus on the case such a rule is TAU1; the remaining cases are similar or simpler. Recall that by Lemmata 6(1) and 7(1) the maximum distance between an occurrence of a variable and its binder is bounded by $\mathsf{depth}(P)$. By Definition 14 any process that can be communicated in P is in $\mathcal{P}_{P,n-1}$ and its maximum depth is also bounded by $\mathsf{depth}(P)$ —which, in turn, by Lemma 8, is bounded by n. The deepest position for a variable is when it is a leaf in the tree associated to the normal form of P. That is, when its depth is exactly $\mathsf{depth}(P)$. If in that position we place a process in $\mathcal{P}_{P,n-1}$ — whose depth is also $\mathsf{depth}(P)$ — then it is easy to see that (the associated tree of) Q has a depth of $2 \cdot \mathsf{depth}(P)$, which is bounded by $2 \cdot n$. Hence, by Lemma 8, Q is in $\mathcal{P}_{P,2 \cdot n}$. □

The lemma below generalizes Proposition 5 to a sequence of reductions.

Lemma 9. *Let P be a* $\text{HO}^{-\mathsf{f}}$ *process. Suppose, for some n, that $P \in \mathcal{P}_{P,n}$. For every Q such that $P \xrightarrow{\widetilde{\alpha}} Q$, it holds that $Q \in \mathcal{P}_{P,2 \cdot n}$.*

Proof. The proof proceeds by induction on k, the number of reductions, exploiting Proposition 5. The base case is when $k = 1$, and it follows by Proposition 5. For the inductive step we assume $k > 1$, so we have that $P \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{k-1}} R \xrightarrow{\alpha_k} Q$. By induction hypothesis we know that $R \in \mathcal{P}_{P,2 \cdot n}$. We then proceed by a case analysis on the rule used to infer $R \xrightarrow{\alpha_k} Q$. We content ourselves with illustrating the case TAU1; the other ones are similar or simpler. We then have that $R \equiv x_1 \parallel \cdots \parallel x_k \parallel \overline{a}\langle T \rangle \parallel a(x).\, R' \parallel S$ and that $Q \equiv x_1 \parallel \cdots \parallel x_k \parallel R'\{{}^{T}\!/x\} \parallel S$.

The tree representation of process R is depicted in Figure 1 (a). There, R'' is used to represent a subprocess of R'. By Corollary 2 the maximum distance between x and its binder $a(x)$ is $\mathsf{depth}(P)$, which in turn is bounded by n (Lemma 8). Moreover, the maximum depth of T is bounded by $\mathsf{maxDepCom}(P)$; by Corollary 2, $\mathsf{depth}(P) \leq n$. The tree representation of Q is given in Figure 1 (b). We then conclude that because of the limitations on the structure of the communicated objects the overall depth of process Q is $2 \cdot \mathsf{depth}(P)$. Hence, and by using Lemma 8, $Q \in \mathcal{P}_{P,2 \cdot n}$, as wanted. □

Corollary 3. *Let $P \in \text{HO}^{-\mathsf{f}}$. Then* $\mathsf{Deriv}(P) \subseteq \mathcal{P}_{P,2 \cdot \mathsf{depth}(P)}$.

We are now ready to prove that relation $\preceq$ is a wqo. We begin by showing that it is a quasi-order.

Proposition 6. *The relation $\preceq$ is a quasi-order.*

Proof. We need to show that $\preceq$ is both reflexive and transitive. From Definition 11, reflexivity is immediate.

Transitivity implies proving that, given processes P, Q, and R such that $P \preceq Q$ and $Q \preceq R$, $P \preceq R$ holds. We proceed by induction on $k = \mathsf{depth}(P)$. If $k = 0$ then we have that $P = x_1 \parallel \cdots \parallel x_k$. Since $P \preceq Q$, we have that $Q = x_1 \parallel \cdots \parallel x_k \parallel S$. and that $R = x_1 \parallel \cdots \parallel x_k \parallel S'$, for some S, S' such that $S \preceq S'$. By Definition 12, the thesis follows. Now suppose $k > 0$. By Definition 11 and by hypothesis we have the following:

$$
\begin{aligned}
P &= \prod_{k=1}^{l} x_k \parallel \prod_{i=1}^{m} a_i(y_i).\, P_i \parallel \prod_{j=1}^{n} \overline{b_j}\langle P'_j \rangle \\
Q &= \prod_{k=1}^{l} x_k \parallel \prod_{i=1}^{m} a_i(y_i).\, Q_i \parallel \prod_{j=1}^{n} \overline{b_j}\langle Q'_j \rangle \parallel S \\
R &= \prod_{k=1}^{l} x_k \parallel \prod_{i=1}^{m} a_i(y_i).\, R_i \parallel \prod_{j=1}^{n} \overline{b_j}\langle R'_j \rangle \parallel S \parallel T.
\end{aligned}
$$

with $P_i \preceq Q_i$, $P'_j \preceq Q'_j$, $Q_i \preceq R_i$, and $Q'_j \preceq R'_j$ $(i \in I, j \in J)$. Since P_i, P'_j, Q_i, Q'_j, R_i, and R'_j have depth $k-1$, by inductive hypothesis $P_i \preceq R_i$ and $P'_j \preceq R'_j$. By Definition 12, the thesis follows and we are done. $\square$

We are now in place to state that $\preceq$ is a wqo.

Theorem 3 (Well-quasi-order). *Let $P \in \mathrm{HO}^{-\mathrm{f}}$ and $n \geq 0$. The relation $\preceq$ is a well-quasi-order over $\mathcal{P}_{P,n}$.*

Proof. The proof is by induction on n.

– Let $n = 0$. Then $\mathcal{P}_{P,0}$ contains processes containing only variables taken from $\mathcal{A}(P)$. The equality on finite sets is a well-quasi-ordering; by Lemma 4 (Higman's Lemma) also $=_*$ is a well quasi-ordering: it corresponds to the ordering $\preceq$ on processes containing only variables.
– Let $n > 0$. Take an infinite sequence of processes $s = P_1, P_2, \ldots, P_l, \ldots$ with $P_l \in \mathcal{P}_{P,n}$. We shall show that the thesis holds by means of successive filterings of the normal forms of the processes in s. By Lemma 5 there exist K_l, I_l and J_l such that
$$P_l \equiv \prod_{k \in K_l} x_k \parallel \prod_{i \in I_l} a_i(y_i).\, P_i^l \parallel \prod_{j \in J_l} \overline{b_j}\langle P_j'^l \rangle$$
with P_i^l and $P_j'^l \in \mathcal{P}_{P,n-1}$. Hence each P_l can be seen as composed of 3 finite sequences: (i) $x_1 \ldots x_k$, (ii) $a_1(y_1).\, P_1^l \ldots a_i(y_i).\, P_i^l$, and (iii) $\overline{b_1}\langle P_1'^l \rangle \ldots \overline{b_j}\langle P_j'^l \rangle$. We note that the first sequence is composed of variables from the finite set $\mathcal{A}(P)$ whereas the other two sequences are composed by elements in $\mathcal{A}(P)$ and $\mathcal{P}_{P,n-1}$. Since we have an infinite sequence of $\mathcal{A}(P)^*$, as $\mathcal{A}(P)$ is finite, by Proposition 4 and Lemma 4 we have that $=_*$ is a wqo over $\mathcal{A}(P)^*$.

By inductive hypothesis, we have that $\preceq$ is a wqo on $\mathcal{P}_{P,n-1}$, hence by Lemma 4 relation $\preceq_*$ is a wqo on $\mathcal{P}^*_{P,n-1}$. We start filtering out s by making the finite sequences $x_1 \ldots x_k$ increasing with respect to $=_*$; let us call this subsequence t. Then we filter out t, by making the finite sequence $a_1(y_1).\, P_1^l \ldots a_i(y_i).\, P_i^l$ increasing with respect to both $\preceq_*$ and $=_*$. This is done in two steps: first, by considering the relation $=_*$ on the subject of the actions (recalling that $a_i, y_i \in \mathcal{A}(P)$), and then by applying another filtering to the continuation using the inductive hypothesis. For the first step, it is worth remarking that we do not consider symbols of the alphabet but pairs of symbols. Since the set of pairs on a finite set is still finite, we know by Higman's Lemma that $=_*$ is a wqo on the set of sequences of pairs (a_i, y_i).
For the sequence of outputs $\overline{b_1}\langle P_1'^l\rangle \ldots \overline{b_j}\langle P_j'^l\rangle$ this is also done in two steps: the subject of the outputs are ordered with respect to $=_*$ and the objects of the output action are ordered with respect to $\preceq_*$ using the inductive hypothesis.
At the end of the process we obtain an infinite subsequence of s that is ordered with respect to $\preceq$.

□

The last thing to show is that the well-quasi-ordering $\preceq$ is strongly compatible with respect to the (finitely branching) LTS associated to $\mathrm{HO}^{-\mathsf{f}}$. We need the following auxiliary lemma:

Lemma 10. *Let P, P', Q, and Q' be* $\mathrm{HO}^{-\mathsf{f}}$ *processes in normal form such that $P \preceq P'$ and $Q \preceq Q'$. It holds that $P\{Q/x\} \preceq P'\{Q'/x\}$.*

Proof. By induction on the structure of P.

1. Cases $P = \mathbf{0}$ and $P = y$, for some $y \neq x$: Immediate.
2. Case $P = x$. Then $P' = x \parallel N$, for some process N. We have that $P\{Q/x\} = Q$ and that $P'\{Q'/x\} = Q' \parallel N\{Q'/x\}$. Since $Q \preceq Q'$ the thesis follows.
3. Case $P = a(y).\, R$. Then $P' = a(y).\, R' \parallel N$, for some process N. Since by hypothesis $P \preceq P'$, then $R \preceq R'$. We then have that $P\{Q/x\} = a(y).\, R\{Q/x\}$ and that $P'\{Q'/x\} = a(y).\, R'\{Q'/'\}x \parallel N\{Q'/x\}$. By inductive hypothesis we obtain that $R\{Q/x\} \preceq R'\{Q'/x\}$, and the thesis follows.
4. Case $P = \overline{a}\langle R\rangle$: Similar to (3).
5. Case $P = R \parallel S$. Then $P' = R' \parallel S' \parallel N$, for some process N, with $R \preceq R'$ and $S \preceq S'$. We then have that $P\{Q/x\} = R\{Q/x\} \parallel S\{Q/x\}$ and $P'\{Q'/x\} = R'\{Q'/x\} \parallel S'\{Q'/x\} \parallel N\{Q'/x\}$. The thesis then follows by inductive hypothesis.

□

Theorem 4 (Strong Compatibility). *Let $P, Q, P' \in$ $\mathrm{HO}^{-\mathsf{f}}$. If $P \preceq Q$ and $P \xrightarrow{\alpha} P'$ then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \preceq Q'$.*

Proof. By case analysis on the rule used to infer transition $\xrightarrow{\alpha}$. We content ourselves with illustrating the cases INP and TAU1; the other ones are similar or simpler.

Case INP Then $P = a(y).\, P_1$ and $Q = a(y).\, Q_1$, with $P_1 \preceq Q_1$. If $P \xrightarrow{a(y)} P' \equiv P_1$ then also $Q \xrightarrow{a(y)} Q' \equiv Q_1$. We then have that $P' \preceq Q'$, as desired.

Case Tau1 Then $P = \overline{a}\langle P_1\rangle \parallel a(y).\, P_2 \parallel N$ and $Q = \overline{a}\langle Q_1\rangle \parallel a(y).\, Q_2 \parallel N'$, with $P_1 \preceq P_2$, $Q_1 \preceq Q_2$, and $N \preceq N'$. If $P \xrightarrow{\tau} P' \equiv P_2\{P_1/y\} \parallel N$ then also $Q \xrightarrow{\tau} Q' \equiv Q_2\{Q_1/y\} \parallel N'$. By Lemma 10 we have $P_2\{P_1/y\} \preceq Q_2\{Q_1/y\}$; using this and the hypothesis the thesis follows.

□

Theorem 5. *Let $P \in \mathrm{Ho}^{-\mathsf{f}}$. The transition system $(\mathsf{Deriv}(P), \longrightarrow, \preceq)$ is a finitely branching well-structured transition system with strong compatibility, decidable $\preceq$, and computable $Succ$.*

Proof. The transition system of $\mathrm{Ho}^{-\mathsf{f}}$ is finitely branching (Fact 1). The fact that $\preceq$ is a well-quasi-order on $\mathsf{Deriv}(P)$ follows from Corollary 3 and Theorem 3. Strong compatibility follows from Theorem 4. □

We can now state the main technical result of the section:

Corollary 4. *Let $P \in \mathrm{Ho}^{-\mathsf{f}}$. Termination of P is decidable.*

Proof. This follows from Theorem 2 and Theorem 5. □

5 Concluding Remarks

We have studied $\mathrm{Ho}^{-\mathsf{f}}$, a higher-order process calculi featuring a limited form of higher-order communication. In $\mathrm{Ho}^{-\mathsf{f}}$, output actions can only include previously received processes in composition with closed ones. This is reminiscent of programming abstractions/paradigms with forms of code mobility in which the recipient is not authorized or capable to access/modify the structure of the received code. The limitation in output actions represents a weakening of the forward capabilities of higher-order processes. Here we have shown it also has consequences both on the expressiveness of the language and on the decidability of termination.

As for the expressiveness issues, by exhibiting an encoding of Minsky machines into $\mathrm{Ho}^{-\mathsf{f}}$, we have shown that convergence is undecidable. Hence, from an *absolute expressiveness* standpoint, $\mathrm{Ho}^{-\mathsf{f}}$ is Turing complete. Now, given the analogous result for HOCORE [4], a *relative expressiveness* issue also arises. Indeed, our encoding of Minsky machines into $\mathrm{Ho}^{-\mathsf{f}}$ is not faithful, which reveals a difference on the criteria each encoding satisfies. This reminds us of the situation in [11], for encodings of Turing complete formalisms into calculi with interruption and compensation. That work offers a detailed comparison of the criteria faithful and unfaithful encodings satisfy. For the sake of space, we do not elaborate further on their exact definition; using the terminology in [11], it suffices to say that the presented encoding satisfies a *weakly Turing completeness* criterion, as opposed to the (stronger) *Turing completeness* criterion that is satisfied by the encoding of Minsky machines into HOCORE in [4]. The discrepancy on the criteria satisfied by each encoding might be interpreted as an expressiveness gap between $\mathrm{Ho}^{-\mathsf{f}}$ and HOCORE; nevertheless, it seems clear that the loss of expressiveness resulting from limiting the forwarding capabilities in HOCORE is much less dramatic than what one would have expected.

The communication style of $\text{HO}^{-\text{f}}$ causes a separation result with respect to HOCORE. In fact, because of the limitation on output actions, it was possible to prove that termination in $\text{HO}^{-\text{f}}$ is decidable. This is in sharp contrast with the situation in HOCORE, where termination is undecidable. In $\text{HO}^{-\text{f}}$, it is possible to provide an upper bound on the depth (i.e. the level of nesting of actions) of the (set of) derivatives of a process. In HOCORE such an upper bound does not exist. This was essential for obtaining the decidability result; for this, we appealed to the approach developed in [9], which relies on the theory of well-structured transition systems [8]. As far as we are aware, this approach for expressiveness has not previously been used in the higher-order setting. The decidability of termination is significant, as it might shed light on the development of verification techniques for higher-order processes.

The $\text{HO}^{-\text{f}}$ calculus is a sublanguage of HOCORE. As such, $\text{HO}^{-\text{f}}$ inherits the many results and properties of HOCORE [4]; most notably, a notion of (strong) bisimilarity which is decidable and coincides with a number of sensible equivalences in the higher-order context. Our results thus complement those in [4] and deepen our understanding of the expressiveness of core higher-order calculi as a whole. Furthermore, by recalling that CCS without restriction is not Turing complete and has decidable convergence, the present results have shaped an interesting expressiveness hierarchy, namely one in which HOCORE is strictly more expressive than $\text{HO}^{-\text{f}}$ (because of the discussion above), and in which $\text{HO}^{-\text{f}}$ is strictly more expressive than CCS without restriction.

Remarkably, we can easily extend our undecidability result proving that (weak) barbed bisimilarity is undecidable in a higher-order calculus obtained by extending $\text{HO}^{-\text{f}}$ with restriction. Consider the encoding of Minsky machines used in Section 3 to prove the undecidability of convergence in $\text{HO}^{-\text{f}}$. Consider now the restriction operator $(\nu\widetilde{x})$ used as a binder for the names in the tuple $\widetilde{x}$. Consider now a Minsky machine N (it is not restrictive to assume that it executes at least one increment instruction) and its encoding P, as defined in Definition 6. We have that N terminates if and only if $(\nu\widetilde{x})P$ is (weakly) barbed equivalent to the process $(\nu d)(\overline{d} \mid d \mid d.\,(\overline{w} \mid !w.\,\overline{w}))$, where $\widetilde{x}$ is the tuple of the names used by the encoding P excluding the name w.

Related Work. The most closely related work is [4], which was already discussed along the paper. We do not know of other works that study the expressiveness of higher-order calculi by restricting higher-order outputs. The recent work [12] studies finite-control fragments of Homer (a higher-order process calculus with locations). While we have focused on decidability of termination and convergence, the interest in [12] in on the decidability of barbed bisimilarity. One of the approaches explored in [12] is based on a type system that bounds the size of processes in terms of their syntactic components (e.g. the number of parallel components, location nesting). Although the restrictions such a type system imposes might be considered as similar in spirit to the limitation on outputs in $\text{HO}^{-\text{f}}$ (in particular, location nesting resembles the output nesting $\text{HO}^{-\text{f}}$ forbids), the fact that the synchronization discipline in Homer depends heavily on the structure of locations makes it difficult to establish a more detailed comparison between the works.

Also similar in spirit to our work, but in a slight different context, are some studies on the expressiveness (of fragments) of Ambient calculus [13]. Ambient calculi are related to higher-order calculi in that both allow the communication of objects with complex structure. Some works on the expressiveness of fragments of Ambient calculi are similar to ours. In particular, in [14] it is determined that, for calculi without restriction (as HO^{-f} and HOCORE), when the capability of ambient movement is ruled out from the language process termination is decidable for the fragment with recursion while it turns out to be decidable for the fragment with recursion. Hence, in [14] the separation between fragments comes from the source of infinite behavior considered, and not from the structures allowed in output action, as in our case. We find, however, that the connections between Ambient-like and higher-order calculi are rather loose, so a proper comparison is difficult also in this case.

Future Work. As already mentioned, a great deal of the expressive power in higher-order calculi resides in constructs for input and output. Here we have studied an alternative for limiting the output capabilities of the calculus; it would be interesting to understand if suitable limitations on input actions are possible, and whether they have influence on the expressiveness and decidability of higher-order calculi. Another interesting direction would be to compare higher-order and Ambient calculi from the expressiveness point of view.

References

1. Thomsen, B.: A calculus of higher order communicating systems. In: Proc. of POPL'89, ACM Press (1989) 143–154
2. Thomsen, B.: Plain CHOCS: A second generation calculus for higher order processes. Acta Inf. **30**(1) (1993) 1–59
3. Sangiorgi, D.: Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. PhD thesis CST–99–93, University of Edinburgh, Dept. of Comp. Sci. (1992)
4. Lanese, I., Pérez, J.A., Sangiorgi, D., Schmitt, A.: On the expressiveness and decidability of higher-order process calculi. In: Proc. of LICS'08, IEEE Computer Society (2008) 145–155 An extended version is available at `http://www.cs.unibo.it/~perez/hocore`.
5. Minsky, M.: Computation: Finite and Infinite Machines. Prentice-Hall (1967)
6. Necula, G.C., Lee, P.: Safe, untrusted agents using proof-carrying code. In: Mobile Agents and Security. Volume 1419 of Lecture Notes in Computer Science., Springer (1998) 61–91
7. Collberg, C.S., Thomborson, C.D.: Watermarking, tamper-proofing, and obfuscation-tools for software protection. IEEE Trans. Software Eng. **28**(8) (2002) 735–746
8. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comput. Sci. **256**(1-2) (2001) 63–92
9. Busi, N., Gabbrielli, M., Zavattaro, G.: On the expressive power of recursion, replication, and iteration in process calculi. Mathematical Structures in Computer Science (2009) To appear. A preliminary version appeared in the Proc. of ICALP'04.
10. Higman, G.: Ordering by divisibility in abstract algebras. Proceedings of the London Mathematical Society (3) **2**(7) (1952) 326–336
11. Bravetti, M., Zavattaro, G.: On the expressive power of process interruption and compensation. Mathematical Structures in Computer Science (2009) To appear.
12. Bundgaard, M., Godskesen, J.C., Haagensen, B., Huttel, H.: Decidable fragments of a higher order calculus with locations. In: Proc. of EXPRESS'08. Electronic Notes in Theoretical Computer Science, Elsevier. To appear.
13. Cardelli, L., Gordon, A.D.: Mobile ambients. Theor. Comput. Sci. **240**(1) (2000) 177–213
14. Busi, N., Zavattaro, G.: On the expressive power of movement and restriction in pure mobile ambients. Theor. Comput. Sci. **322**(3) (2004) 477–515