# On the Expressiveness of Forwarding in Higher-Order Communication (December 6, 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. Then, as a way of recovering the expressiveness loss due to limited forwarding, we extend the calculus with a form of process suspension called *passivation*. Somewhat surprisingly, in the calculus extended with passivation both termination and convergence are 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 $\pi$-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 $\pi$-calculus for the description of mobile systems. These calculi are inspired by, and formally close to, the $\lambda$-calculus, whose basic computational step — $\beta$-reduction — involves term instantiation. As in the $\lambda$-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. This way, continuations following output messages have been left out, so communication in HOCORE is asynchronous. 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:

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

* Research partially funded by EU Integrated Projects HATS (contract number 231620) and SENSORIA (contract number 016004).

An input prefixed process $a(x).\,P$ can receive on name (or channel) $a$ a process to 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 its minimality, HOCORE was shown to be Turing complete via a termination preserving encoding of Minsky machines [5]. Therefore, properties such as

- *termination*, i.e. non existence of divergent computations
- *convergence*, i.e. existence of a terminating computation

are both undecidable in HOCORE[1]. In contrast, somewhat surprisingly, strong bisimilarity is decidable, and several sensible bisimilarities coincide with it.

In this chapter, 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 $R = a(x).\,\overline{b}\langle P_x\rangle$, where $P_x$ stands for a process $P$ with free occurrences of a variable $x$. Intuitively, $R$ receives a process on name $a$ and forwards it on name $b$. It is easy to see that since there are no limitations on the structure of objects in output actions, 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 $P_x = \overline{b_1}\langle\overline{b_2}\langle\ldots\overline{b_k}\langle x\rangle\rangle\ldots\rangle$. 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 to assess the expressive power of higher-order concurrency.

With the above consideration in mind, in this chapter we propose $\text{HO}^{-\mathsf{f}}$, a sublanguage of HOCORE in which output actions are limited so as to rule out the nesting capability (Section 2). In $\text{HO}^{-\mathsf{f}}$, output actions can communicate the parallel composition of two kinds of objects:

1. closed processes (i.e. processes that do not contain free variables), and
2. processes received through previously executed input actions.

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

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

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

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

The limited style of higher-order communication enforced in $\text{HO}^{-\text{f}}$ is relevant from a pragmatic perspective. In fact, communication in $\text{HO}^{-\text{f}}$ 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 $[\![C[P]]\!] = D[P]$.

More concretely, communication as in $\text{HO}^{-\text{f}}$ 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* [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 of the communication of source code that is made difficult to understand for, e.g., security/copyright reasons, while preserving its functionality.

In this chapter we study the expressiveness of $\text{HO}^{-\text{f}}$ using decidability of termination and convergence of processes as a yardstick. Our main results are:

**Undecidability of Convergence in $\text{HO}^{-\text{f}}$.** Similarly as HOCORE, $\text{HO}^{-\text{f}}$ is shown to be Turing complete by exhibiting an encoding of Minsky machines into $\text{HO}^{-\text{f}}$. The calculus thus retains a significant expressive power despite of the limited forwarding capability. Unlike the encoding of Minsky machines in HOCORE, however, the encoding in $\text{HO}^{-\text{f}}$ is not *faithful* in that 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 allows us to prove that a Minsky machine terminates if and only if its encoding in $\text{HO}^{-\text{f}}$ converges. Consequently, convergence in $\text{HO}^{-\text{f}}$ is *undecidable*.

**Decidability of Termination in $\text{HO}^{-\text{f}}$.** In sharp contrast with HOCORE, termination in $\text{HO}^{-\text{f}}$ is *decidable*. This result is obtained by appealing to the theory of *well-structured transition systems* [8,9,10], following the approach used in [11]. To the best of our knowledge, this is the first time the theory of well-structured transition systems is applied in a higher-order concurrency setting. This is significant because the adaptation to the $\text{HO}^{-\text{f}}$ case is far from trivial. Indeed, as we shall discuss, this approach relies on defining 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 $\text{HO}^{-\text{f}}$, 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.

**Undecidability of Termination *and* Convergence in $\text{HO}^{-\text{f}}$ with Passivation.** The decidability of termination in $\text{HO}^{-\text{f}}$ provides compelling evidence on the fact that the limited forwarding entails a loss of expressive power for HOCORE. It is therefore legitimate to investigate whether such an expressive power can be recovered while preserving the essence of the limited forwarding in $\text{HO}^{-\text{f}}$. For this purpose, we extend $\text{HO}^{-\text{f}}$ with a *passivation* construct that allows to *suspend* the execution of

a running process. Forms of process suspension (such as passivation) are of both practical and theoretical interest as they are at the heart of mechanisms for *dynamic system reconfiguration*. The extension of $\text{HO}^{-\mathsf{f}}$ with passivation, called $\text{HOP}^{-\mathsf{f}}$, is shown to be Turing complete by exhibiting a *deterministic* encoding of Minsky machines. Therefore, in $\text{HOP}^{-\mathsf{f}}$ *both* convergence and termination are *undecidable*. To the best of our knowledge, ours is the first result on the expressiveness and decidability of constructs for process suspension in the context of higher-order process calculi.

The remainder of the chapter is structured as follows. 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 for $\text{HO}^{-\mathsf{f}}$ is addressed in Section 4. The expressiveness results for $\text{HOP}^{-\mathsf{f}}$ are presented in Section 5. Some final remarks, as well as a review of related work, are included in Section 6.

## 2 The Calculus

We now introduce the syntax and semantics of $\text{HO}^{-\mathsf{f}}$. We use $a, b, c$ to range over names, and $x, y, z$ to range over variables; the sets of names and variables are disjoint.

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

An input $a(x).\, P$ binds the free occurrences of $x$ in $P$. We write $\mathsf{fv}(P)$ and $\mathsf{bv}(P)$ for the set of free and bound variables in $P$, respectively. 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 $\overline{a}\langle\prod_{k\in K} x_k \parallel P\rangle$. 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 $P$.

The Labeled Transition System (LTS) of $\text{HO}^{-\mathsf{f}}$ is defined in Figure 1. It decrees there are three forms of transitions: $\tau$ 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 $\alpha$ to indicate a generic label of a transition. The notions of free and bound variables extend to labels as expected.

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

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

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

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

**Fig. 1.** An LTS for $\text{HO}^{-\mathsf{f}}$. Rules ACT2 and TAU2, the symmetric counterparts of ACT1 and TAU1, have been omitted.

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 internal runs of a process are given by sequences of *reductions*. Given a process $P$, its reductions $P \longrightarrow P'$ are defined as $P \xrightarrow{\tau} P'$. We denote with $\longrightarrow^*$ the reflexive and transitive closure of $\longrightarrow$; notation $\longrightarrow^j$ is to stand for a sequence of $j$ reductions. We use $P \not\rightarrow$ to denote that there is no $P'$ such that $P \longrightarrow P'$. Following [11] we now define process convergence and process termination. Observe that termination implies convergence while the opposite does not hold.

**Definition 2.** *Let $P$ be a $\text{HO}^{-\mathsf{f}}$ process.*

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

## 3 Undecidability of Convergence in $\text{HO}^{-\mathsf{f}}$

In this section we show that $\text{HO}^{-\mathsf{f}}$ is powerful enough to model Minsky machines [5], a Turing complete model. 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 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

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

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

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

**Table 1.** Reduction of Minsky machines

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 $\longrightarrow_{\text{M}}$, is defined in Table 1.

In the encoding of a Minsky machine into $\text{HO}^{-\text{f}}$ we will find it convenient to have a simple form of guarded replication. This construct can be encoded in $\text{HO}^{-\text{f}}$ 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 $\text{HO}^{-\text{f}}$, the encoding is not compositional and replications cannot be nested.

**Definition 3.** *Assume a fresh name $c$. The encoding of* input-guarded replication *is as follows:*

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

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

The above encoding preserves termination.

**Lemma 4 (Correctness of $[\![\cdot]\!]_{\text{i!}}$)** *Let $P$ be a* $\text{HO}^{-\text{f}}$ *process with non-nested input-guarded replications.*

- *If $[\![P]\!]_{\text{i!}} \longrightarrow Q$ then $\exists P'$ such that $P \longrightarrow P'$ and either $[\![P']\!]_{\text{i!}} = Q$ or $Q \longrightarrow [\![P']\!]_{\text{i!}}$.*
- *If $P \longrightarrow P'$ then either $[\![P]\!]_{\text{i!}} \longrightarrow [\![P']\!]_{\text{i!}}$ or $[\![P]\!]_{\text{i!}} \longrightarrow\!\!\longrightarrow [\![P']\!]_{\text{i!}}$.*
- *$[\![P]\!]_{\text{i!}} \not\rightarrow$ iff $P \not\rightarrow$.*

*Proof.* By induction on the transitions. □

REGISTER $r_j$ $\quad [\![r_j = m]\!]_{\mathsf{M}} = \prod_1^m \overline{u_j}$

INSTRUCTIONS $(i : I_i)$

$$
\begin{aligned}
[\![(i : \texttt{INC}(r_j))]\!]_{\mathsf{M}} &= !p_i.\,(\overline{u_j} \parallel set_j(x).\,\overline{set_j}\langle x \parallel \textsc{Inc}_j\rangle \parallel \overline{p_{i+1}}) \\
[\![(i : \texttt{DECJ}(r_j, s))]\!]_{\mathsf{M}} &= \quad !p_i.\,\overline{m_i} \\
&\quad \parallel !m_i.\,(\overline{loop} \parallel u_j.\,loop.\,set_j(x).\,\overline{set_j}\langle x \parallel \textsc{Dec}_j\rangle \parallel \overline{p_{i+1}}) \\
&\quad \parallel !m_i.\,set_j(x).\,(x \parallel \overline{set_j}\langle x\rangle \parallel \overline{p_s})
\end{aligned}
$$

where

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

**Table 2.** Encoding of Minsky machines

### 3.1 Encoding Minsky machines into HO$^{-\mathsf{f}}$

The encoding of Minsky machines into HO$^{-\mathsf{f}}$ is denoted by $[\![\cdot]\!]_{\mathsf{M}}$ and presented in Table 2. We begin by discussing the encodings of registers and instructions; then we define the encoding of the a configuration of a Minsky machine.

A register $r_j$ that stores the number $m$ is encoded as the parallel composition of $m$ copies of the unit process $\overline{u_j}$. To implement the test for zero 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 name $set_j$. 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 $r_j$, we have that $\textsc{Log}_j = \prod_k \textsc{Inc}_j \parallel \prod_l \textsc{Dec}_j$, 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 guess is right as the content of $r_j$ is greater than 0) then a synchronization with the input on $loop$ that guards the update 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 $loop$. 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 $\text{INC}_j$ process to be matched by a complementary one. Otherwise, then it is 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 $loop$. DIV then takes place, thus spawining a divergent computation.

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 from zero.

We are now ready to define the encoding of a configuration of the Minsky machine. It is given as expected; the encodings of instructions and registers are put in parallel with a process that spawns divergent behavior in case of a wrong guess.

**Definition 5 (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 $r_j$, $p_1, \ldots, p_n$, $set_j$, $loop$, $check_j$. Also, let* DIV *be a divergent process (e.g. $\overline{w} \parallel !w.\overline{w}$). Given the encodings in Table 2, we have:*

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

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

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:*

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

### 3.2 Correctness of the Encoding

We formalize the correctness of our encoding by means of two lemmas ensuring completeness (Lemma 2) and soundness (Lemma 3). Both these lemmas give us Theorem 1. We begin by formalizing the following intuition: removing the program counter from the encoding of configurations leads to a stuck process.

**Proposition 1.** *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:*

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

*Then $P \not\to$.*

*Proof.* Straightforward by the following facts:

1. Processes $[\![r_0 = m_0]\!]_{\mathsf{M}}$, $[\![r_1 = m_1]\!]_{\mathsf{M}}$, $\overline{set_0}\langle \text{LOG}_0[k_0, l_0]\rangle$, and $\overline{set_1}\langle \text{LOG}_1[k_1, l_1]\rangle$ 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. $loop$. 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 $r_j$.*

Because we assume the value of both registers to be initialized with zero before executing the instructions, we following is immediate.

**Fact 1** *Let $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$ be an annotated Minsky configuration. We then have, for $n \in \{0, 1\}$: (i) $k_n = l_n$ if and only if $r_n = 0$; and (ii) $k_n > l_n$ if and only if $r_n > 0$.*

The following proposition formalizes an invariant condition on the number of decrements and increments stored by the logs in the encoding. This invariant will be useful later, when formalizing the conditions under which our encoding leads to divergent computation.

**Proposition 6** *Let $(1, 0, 0)$ be the initial configuration of a Minsky machine $N$. For all $P$ such that*

1. $[\![(1, 0, 0)]\!]_{\mathsf{M}} \longrightarrow^* P$;
2. *for some $S$, $P \equiv \overline{set_0}\langle \text{LOG}_0[k_0, l_0]\rangle \parallel \overline{set_1}\langle \text{LOG}_1[k_1, l_1]\rangle \parallel S$.*

*Then, for $j \in \{0, 1\}$, it holds that $k_j \geq l_j$.*

*Proof.* By contradiction, assuming that $k_j < l_j$. For $k_j < l_j$ to hold there was an execution in which $k_j = l_j$ and then a DEC was added to one of the logs. Using Fact 1 we know that this means that $r_j = 0$. In turn, by the encoding of counters, we know this means that there is no top-level occurrence of $\overline{u_j}$. By inspection of the structure of the encoding of the decrement instruction, we know that a process DEC can only be added to one of the logs if the encoding of decrement takes the branch

$$\overline{loop} \parallel u_j.\, loop.\, set_j(x).\, \overline{set_j}\langle x \parallel \text{DEC}_j\rangle \parallel \overline{p_{i+1}}\,.$$

Observe that a modification to the log can only occur in the event in which there is a top-level occurrence of $\overline{u_j}$. Indeed, the input on $set$ (which modifies the log) is guarded by $u_j$. Therefore, in order to modify the log an output on $u_j$ is indispensable; such an output is not available, so we reach a contradiction. □

With a little abuse of notation, we use notation $Q \not\rightarrow$ 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 $(i, m_0, m_1) \not\rightarrow$ then $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \not\rightarrow$*
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 $(i, m_0, m_1) \not\rightarrow$ 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 1.

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 $(i : \texttt{INC}(r_0))$. Its encoding into $\text{HO}^{-\mathsf{f}}$ is as follows:

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

We begin by noting that the program counter $p_i$ is consumed by the encoding of the instruction $i$. As a result, processes $\overline{u_0}$ and $\overline{p_{i+1}}$ are left unguarded; this represents the increment and the invocation of the next instruction, respectively. We then have:

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

where $S$ stands for the rest of the system, i.e.

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

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

$$T \longrightarrow \overline{p_{i+1}} \parallel [\![r_0 = m_0 + 1]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{h=1}^{n} [\![(h : I_h)]\!]_{\mathsf{M}} \parallel$$
$$loop.\, \textsc{Div} \parallel \overline{set_0}\langle \textsc{Log}_0[k_0 + 1, l_0]\rangle \parallel \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle = T'\,.$$

We notice that $T' \equiv [\![(i + 1, m_0 + 1^{k_0+1,l_0}, m_1^{k_1,l_1})]\!]_{\mathsf{M}}$, as desired.

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

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

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

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

where $S$ stands for the rest of the system, i.e.

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

In $T_1$ there is an internal choice on the name $m_i$, which represents a guess on the value of $r_0$: $\overline{m_i}$ can either synchronize with the first input-guarded process (thus performing the actual decrement of the register) or with the second one (thus performing a jump). Let us suppose $T_1$ makes the right guess in this case, i.e. $\overline{m_i}$ 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, represented by $S$ (as in the previous case) along with the input-guarded process that was not involved in the synchronization. Notice that $S'$ is stuck:

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

Since we have assumed that $r_0 > 0$, we are sure that a synchronization on $u_0$ can take place, and thus 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 update of the log associated to $r_0$ can take place, and a synchronization on $set_0$ is performed:

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

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 $(i : \texttt{DECJ}(r_0, s))$. In this case, $m_0 = 0$. Hence, using Fact 1 we have that $k_0 = l_0$.

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

$$T_1 \longrightarrow [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel set_0(x).\,(x \parallel \overline{set_0}\langle x\rangle \parallel \overline{p_s}) \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \parallel S' = T_2\,.$$

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. Notice that such a process is 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 $set_0$. As a result, the content of the log is left at the top-level and hence executed. It is not lost, however, as it is still preserved inside an output on $set_0$:

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

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 a process $\textsc{Inc}_0$ and a corresponding process $\textsc{Dec}_0$. Half of these interactions correspond to synchronizations on $check_0$, whereas the rest are synchronizations on $loop$. All of these processes 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) $T_4 \equiv [\![(s, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathsf{M}}$, 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 $P \in \textsc{Ho}^{-\mathsf{f}}$, 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* $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \nrightarrow$ *then* $(i, m_0, m_1) \nrightarrow$*.*

*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 (we use $k_0$ and $l_0$ to denote the number of increments and decrements on $r_0$, respectively):

1. $P \equiv [\![(i+1, m_0-1, m_1)]\!]_{\mathsf{M}}$: in this case, $n = 5$, as illustrated in the analogous case in the proof of Lemma 2(2).
2. $P \equiv \textsc{Div}$: this is the case in which the process makes a wrong guess on the content of the register. Let us elaborate on this possible execution, starting from $T_1$ as defined in the analogous case in the proof of Lemma 2(2). Since $r_0 > 0$, using Fact 1, we know that $k_0 > l_0$. It is sufficient to assume that $k_0 = l_0 + 1$. After the synchronization on $m_i$, we have:

$$T_1 \longrightarrow [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel set_0(x).\,(x \parallel \overline{set_0}\langle x\rangle \parallel \overline{p_s}) \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \parallel loop.\,\textsc{Div} \parallel S' = T_2$$

where $S'$ is the rest of the system, i.e.

$$S' = !m_i.\,(\overline{loop} \parallel u_0.\,loop.\,set_0(x).\,\overline{set_0}\langle x \parallel \textsc{Dec}_0\rangle \parallel \overline{p_{i+1}}) \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{h=1}^{n} [\![(h : I_h)]\!]_{\mathsf{M}} \parallel \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle.$$

In $T_2$ there is a synchronization on $set_0$. Using the definition of LOG, we have:

$$T_2 \longrightarrow [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel \prod^{l_0+1} \textsc{Inc}_0 \parallel \prod^{l_0} \textsc{Dec}_0 \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \parallel \overline{p_s} \parallel loop.\,\textsc{Div} \parallel S' = T_3\,.$$

At this point there are $l_0$ synchronizations between the copies of $\textsc{Dec}_0$ and those of $\textsc{Inc}_0$. There is a copy of $\textsc{Inc}_0$ that remains without synchronizing and hence we have:

$$T_3 \longrightarrow^{l_0} [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel \overline{loop} \parallel check_0.\, loop \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \parallel \overline{p_s} \parallel loop.\, \textsc{Div} \parallel S' = T_4\,.$$

In $T_4$, it suffices a synchronization on $loop$ to produce divergence. Indeed, $T_4 \longrightarrow\equiv$ DIV. Notice that by virtue of Proposition 6 for every derivative of $T_4$ it holds that the number of increments is greater or equal than the number of decrements, which is sufficient to spawn divergent computations. We then conclude that, with $n \geq 4 + l_0$, $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \longrightarrow^n\equiv$ DIV and the thesis holds.

**Case** $i : \texttt{DEC}(r_0, s)$ **with** $r_0 = 0$**:** Then the process evolves non-deterministically to one of the following cases (we use $k_0$ and $l_0$ to denote the number of increments and decrements on $r_0$, respectively):

1. $P \equiv [\![(s, m_0, m_1)]\!]_{\mathsf{M}}$: in this case, $n = 3 + k_0 + l_0$, as illustrated in the analogous case in the proof of Lemma 2(2).
2. $P \equiv$ DIV: this is the case in which the process makes a wrong guess on the content of the register. Again, we carry our analysis starting from process $T_1$ given in the analogous case of the proof of Lemma 2(2). Using Fact 1 we know that $k_0 = l_0$. After the synchronization on $m_i$ we have

$$T_1 \longrightarrow [\![r_0 = m_0]\!]_{\mathsf{M}} \parallel \overline{loop} \parallel u_0.\, loop.\, set_0(x).\, \overline{set_0}\langle x \parallel \textsc{Dec}_0\rangle \parallel \overline{p_{i+1}} \parallel \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \parallel loop.\, \textsc{Div} \parallel S' = T_2$$

where $S'$ is the rest of the system, i.e.

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

It is easy to observe that since $r_0 = 0$ there is no output on $u_j$ that can synchronize with the input in $T_1$. In fact, the only possible synchronization is on $loop$, which leaves the divergent process unguarded. So we have that $T_2 \longrightarrow\equiv$ DIV. Hence, $[\![(s, m_0, m_1)]\!]_{\mathsf{M}} \longrightarrow^n\equiv$ DIV with $n = 2$, and the thesis holds.

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

As for (3), using Proposition 1 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 $(i, m_0, m_1)$. Then $(i, m_0, m_1)$ terminates if and only if process $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$ converges.*

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

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

## 4 Decidability of Termination in $\text{HO}^{-\text{f}}$

In this section we prove that termination is decidable for $\text{HO}^{-\text{f}}$ 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.

### 4.1 Well-Structured Transition Systems

The following results and definitions are from [10], 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 $\rightarrow^*$ denotes the reflexive and transitive closure of the relation $\rightarrow$.

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

The function $Succ$ will also be used on sets by assuming 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 $TS = (S, \rightarrow)$, equipped with a quasi-order $\leq$ on $S$, such that the two following conditions hold:*

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

The following theorem is a special case of Theorem 4.6 in [10] and will be used to obtain our decidability result.

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

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

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

**Fig. 2.** A finitely branching LTS for $\text{HO}^{-\mathsf{f}}$. Rules ACT2 and TAU2, the symmetric counterparts of ACT1 and TAU1, have been omitted.

We will also need a result due to Higman [12] which allows to extend a well-quasi-order from a set $S$ to the set of the finite sequences on $S$. More precisely, given a set $S$ let us 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 $t \leq_* u$ if and only if 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 [12]).** *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 2.** *Let $S$ be a finite set. Then the equality is a wqo over $S$.*

### 4.2 A Finitely Branching LTS for $\text{HO}^{-\mathsf{f}}$

In order to exploit the theory of well-structured transition systems, a finitely branching LTS for $\text{HO}^{-\mathsf{f}}$ is necessary. This is not a significant requirement in our case; the sensible issue here is the treatment of alpha-conversion. To that end, we introduce an alternative LTS *without* alpha-conversion. As we shall see, since we restrict ourselves to closed processes and proofs focus on internal synchronizations, the finitely branching LTS is equivalent to that proposed in Section 2. The alternative LTS is given in Figure 2; its most noticeable is the absence of a side condition on rule ACT1.

**Lemma 5.** *Let $P$ be a closed $\text{HO}^{-\mathsf{f}}$ process. For every $P' \in \text{HO}^{-\mathsf{f}}$ if $P \longrightarrow P'$ then $P'$ is a closed process.*

*Proof.* By induction on the height of the inference tree for $P \longrightarrow P'$ considering the possible cases of the last step of the inference. There are two cases; let us consider first the case in which TAU1 be the last rule applied. Then $P \equiv a(x).\,Q \parallel \overline{a}\langle R\rangle$ and $P' \equiv Q\{^R/_x\}$. We know that $P$ is a closed process; hence, $R$ is a closed process and $Q$ is an open process such that $\mathsf{fn}(Q) = \{x\}$. Then the process $P'$ is closed since it is equivalent to the process $Q$ where all the free occurrences of the name $x$ has been replaced with the closed process $R$. Now let ACT1 be the last rule applied. Then $P \equiv P_1 \parallel P_2$ and $P' \equiv P_1' \parallel P_2$. The thesis follows by applying the inductive hypothesis on $P_1$. □

*Remark 1.* Notice that if we consider closed processes and restrict ourselves to reductions then $\mathsf{fv}(\alpha) \cap \mathsf{bv}(P_2)$ —the side condition in rules ACT1 and ACT2 in the LTS in Figure 1— is always equivalent to the empty set. This means that $\alpha$-conversion is not necessary for closed processes.

As before, the internal runs of a process are given by sequences of *reductions*. Given a process $P$, its reductions in the alternative LTS $P \longmapsto P'$ are defined as $P \stackrel{\tau}{\longmapsto} P'$. We denote with $\longmapsto^*$ the reflexive and transitive closure of $\longmapsto$. We use $P \not\longmapsto$ to denote that there is no $P'$ such that $P \longmapsto P'$.

Given a process $P$, we shall use $P_\alpha$ to denote the result of applying the standard alpha-conversion without name captures over $P$.

**Lemma 6.** *Let $P$ be a closed* $\text{HO}^{-\mathsf{f}}$ *process. Then, $P \rightarrow P'$ iff $P \longmapsto P''$ and $P'' \equiv P'_\alpha$, for some $P'$ in* $\text{HO}^{-\mathsf{f}}$.

*Proof.* The "if" direction follows easily from Remark 1. The "only if" direction is straightforward by observing that since $P$ is a closed process, $P''$ is one of the possible evolutions of $P$ in $\rightarrow$. □

**Corollary 2.** *Let $P$ be a closed* $\text{HO}^{-\mathsf{f}}$ *process. If $P \longmapsto P'$ then $P'$ is a closed process in* $\text{HO}^{-\mathsf{f}}$.

*Proof.* Straightforward from Lemma 5 and Lemma 6. □

**Corollary 3.** *Let $P$ be a closed* $\text{HO}^{-\mathsf{f}}$ *process. $P \not\rightarrow$ iff $P \not\longmapsto$.*

*Proof.* Straightforward from Lemma 6. □

*Remark 2.* The encoding of a Minsky machine presented in Section 3 is a closed process. Hence, all the results in that section hold for the LTS in Figure 2 as well.

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

**Definition 11 (Alphabet of a process).** *Let $P$ be a* $\text{HO}^{-\mathsf{f}}$ *process. The* alphabet of $P$, *denoted $\mathcal{A}(P)$, is inductively defined as:*

$$\mathcal{A}(\mathbf{0}) = \emptyset \qquad \mathcal{A}(P \parallel Q) = \mathcal{A}(P) \cup \mathcal{A}(Q) \qquad \mathcal{A}(x) = \{x\}$$

$$\mathcal{A}(a(x).\, P) = \{a, x\} \cup \mathcal{A}(P) \qquad \mathcal{A}(\overline{a}\langle P\rangle) = \{a\} \cup \mathcal{A}(P)$$

The following proposition can be shown for the alternative LTS because it does not consider alpha-conversion. As a matter of fact, had we considered open processes, we would have required $\alpha$-conversion. In such a case, the inclusion $\mathcal{A}(P'_2\{^R/_x\}) \subseteq \mathcal{A}(P'_2) \cup \mathcal{A}(R)$ would no longer hold. This is because by using $\alpha$-conversion during substitution some new variables could be added to the alphabet.

**Proposition 3.** *Let $P$ and $P'$ be closed* $\text{HO}^{-\mathsf{f}}$ *processes. If $P \longmapsto P'$ then $\mathcal{A}(P') \subseteq \mathcal{A}(P)$.*

*Proof.* We proceed by a case analysis on the rule used to infer $\longmapsto$. We thus have four cases:

**Case TAU1** Then $P = P_1 \parallel P_2$, $P' = P_1' \parallel P_2'\{^R/_x\}$, with $P_1 \stackrel{\overline{a}\langle R\rangle}{\longmapsto} P_1'$ and $P_2 \stackrel{a(x)}{\longmapsto} P_2'\{^R/_x\}$. By Definition 11 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 11 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, $\alpha$-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 $P_1 \stackrel{\alpha}{\longmapsto} P_1'$. 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.

□

**Fact 2** *The LTS for* $\text{HO}^{-\mathsf{f}}$ *given in Figure 2 is finitely branching.*

### 4.3 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 proceed as follows. First we define 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 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 4.2. The decidability result is then obtained by resorting to the theory of well-structured transition systems introduced in Section 4.1.

**Definition 12 (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 7.** *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 13 (Relation $\preceq$).** *Let $P, Q \in \text{HO}^{-\mathsf{f}}$. 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*

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

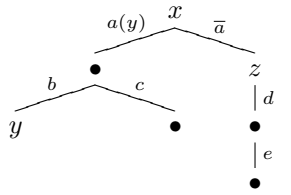
*with $P_i \preceq Q_i$ and $P'_j \preceq Q'_j$, for $i \in [1..m]$ and $j \in [1..n]$.*

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:




This intuitive representation of processes in normal form as trees will be useful to reason about the structure of $\text{HO}^{-\mathsf{f}}$ terms. We begin by defining the *depth* of a process. Notice that such a depth corresponds to the maximum depth of its tree representation.

**Definition 14 (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 $\text{HO}^{-\mathsf{f}}$ 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 can be built using the alphabet of $P$ and whose depth is at most $n$.

**Definition 15.** *Let $n$ be a natural number and $P \in \text{HO}^{-\mathsf{f}}$. We define the set $\mathcal{P}_{P,n}$ as follows:*

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

*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 $\text{HO}^{-\mathsf{f}}$ 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 guarding a variable. Since variables are allowed only at the top level of the output objects, their relative distance will remain invariant during reductions. This allows to obtain a bound on the structure of $\text{HO}^{-\mathsf{f}}$ 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 first define the maximum distance between a variable and its binder.

**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 distance of $P$ *as:*

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

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

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

*Proof.* Part (1) is immediate from Definitions 14 and 16. Part (2) follows by a case analysis on the rule used to infer $\longmapsto$. 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 16 in both processes, we obtain

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

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 17.** *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 9 (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 $P \longmapsto Q$,* $\mathsf{maxDepCom}(Q) \leq \mathsf{maxDepCom}(P)$.

*Proof.* Part (1) is immediate from Definitions 14 and 17. Part (2) follows by a case analysis on the rule used to infer $\longmapsto$. Again, 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 16 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 $\text{HO}^{-\mathsf{f}}$, $S'$ 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.

□

Generalizing Lemmata 8 and 9 we obtain:

**Corollary 4.** *Let $P$ be a $\text{HO}^{-\mathsf{f}}$ process. For every $Q$ such that $P \longmapsto^* Q$, 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 $\mathcal{P}_{P,2\cdot\mathsf{depth}(P)}$. We will investigate the properties of the relation $\preceq$ on such an approximation; such properties will also hold for the set of derivatives.

**Definition 18.** *Let $P \in \text{HO}^{-\mathsf{f}}$. Then we define* $\mathsf{Deriv}(P) = \{Q \mid P \longmapsto^* Q\}$

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

**Lemma 10.** *Let $P, Q$ be $\text{HO}^{-\mathsf{f}}$ processes. $Q \in \mathcal{P}_{P,n}$ if and only if $\mathsf{depth}(Q) \leq n$.*

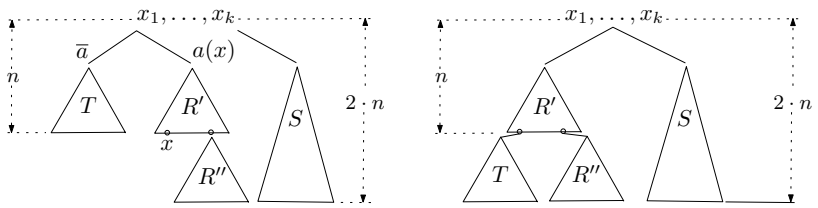
*Proof.* The "if" direction is straightforward by definition of $\mathcal{P}_{P,n}$ (Definition 15).

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

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

where, for every $i \in I$ and $j \in J$, both $\mathsf{depth}(Q_i) \leq \mathsf{depth}(Q) \leq n - 1$ and $\mathsf{depth}(Q'_j) \leq \mathsf{depth}(P) \leq n - 1$. By inductive hypothesis, each $Q_i$ and $Q'_j$ is in $\mathcal{P}_{P,n-1}$. Then, by Definition 15, $Q \in \mathcal{P}_{P,n}$ and we are done. □

**Proposition 4.** *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 $P \longmapsto Q$, it holds that $Q \in \mathcal{P}_{P,2\cdot n}$.*




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

**Fig. 3.** Tree representation of a reduction $R \longmapsto Q$, as in the proof of Lemma 11.

*Proof.* We proceed by case analysis on the rule used to infer $\longmapsto$. We focus on the case such a rule is TAU1; the remaining cases are similar or simpler. Recall that by Lemmata 8(1) and 9(1) the maximum distance between an occurrence of a variable and its binder is bounded by $\mathsf{depth}(P)$. By Definition 15 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 10, 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 10, $Q$ is in $\mathcal{P}_{P,2 \cdot n}$. □

The lemma below generalizes Proposition 4 to a sequence of transitions.

**Lemma 11.** *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 $P \longmapsto^* Q$, it holds that $Q \in \mathcal{P}_{P,2 \cdot n}$.*

*Proof.* The proof proceeds by induction on $k$, the length of $\longmapsto^*$, exploiting Proposition 4. The base case is when $k = 1$, and it follows by Proposition 4. For the inductive step we assume $k > 1$, so we have that $P \longmapsto^* R \longmapsto Q'$ where the sequence from $P$ to $R$ is of lenght $k - 1$. 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 $R \longmapsto Q$. As usual, 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 3 (a). There, $R''$ is used to represent a subprocess of $R'$. By Corollary 4 the maximum distance between $x$ and its binder $a(x)$ is $\mathsf{depth}(P)$, which in turn is bounded by $n$ (Lemma 10). Moreover, the maximum depth of $T$ is bounded by $\mathsf{maxDepCom}(P)$; by Corollary 4, $\mathsf{depth}(P) \leq n$. The tree representation of $Q$ is given in Figure 3 (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 10, $Q \in \mathcal{P}_{P,2 \cdot n}$, as wanted. □

**Corollary 5.** *Let $P \in \mathrm{HO}^{-\mathsf{f}}$. 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 5.** *The relation $\preceq$ is a quasi-order.*

*Proof.* We need to show that $\preceq$ is both reflexive and transitive. From Definition 12, 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 13, the thesis follows. Now suppose $k > 0$. By Definition 12 and by hypothesis we have the following:

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

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

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

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 13, the thesis follows and we are done. □

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

**Theorem 3 (Well-quasi-order).** *Let $P \in \text{HO}^{-\mathsf{f}}$ be a closed process 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 7 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 2 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 LTS in Figure 2. We need the following auxiliary lemma:

**Lemma 12.** *Let $P, P', Q$, and $Q'$ be $\mathrm{HO}^{-\mathsf{f}}$ processes in normal form such that $P \preceq P'$ and $Q \preceq Q'$. Then it holds that $P\{^Q/_x\} \preceq P'\{^{Q'}/_x\}$.*

*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 $P \longmapsto P'$ then there exists $Q'$ such that $Q \longmapsto Q'$ and $P' \preceq Q'$.*

*Proof.* By case analysis on the rule used to infer reduction $P \longmapsto P'$. We content ourselves with illustrating the case derived from the use of rule TAU1; the other ones are similar or simpler. We then have that $P = \overline{a}\langle P_1\rangle \parallel a(y).\,P_2 \parallel N$ 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 $P \longmapsto P' \equiv P_2\{^{P_1}/_y\} \parallel N$ then also $Q \longmapsto Q' \equiv Q_2\{^{Q_1}/_y\} \parallel N'$. By Lemma 12 we have $P_2\{^{P_1}/_y\} \preceq Q_2\{^{Q_1}/_y\}$; using this and the hypothesis the thesis follows. □

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

*Proof.* The transition system of $\text{HO}^{-\mathsf{f}}$ is finitely branching (Fact 2). The fact that $\preceq$ is a well-quasi-order on $\mathsf{Deriv}(P)$ follows from Corollary 5 and Theorem 3. Strong compatibility follows from Theorem 4. □

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

**Corollary 6.** *Let $P \in \text{HO}^{-\mathsf{f}}$ be a closed process. Then, termination of $P$ is decidable.*

*Proof.* This follows from Theorem 2, Theorem 5, and Corollary 3. □

## 5 On the Interplay of Fowarding and Passivation

The decidability of termination in $\text{HO}^{-\mathsf{f}}$ presented in Section 4 provides compelling evidence on the fact that the limited forwarding entails a loss of expressive power for HOCORE. It is therefore worth investigating alternatives for recovering such an expressive power while preserving the essence of limited forwarding.

In this section we examine one such alternatives. We analyze the consequences of extending $\text{HO}^{-\mathsf{f}}$ with a *passivation* construct, an operator that allows to *suspend* the execution of a process at run time. As such, it comes in handy to represent scenarios of (dynamic) system reconfiguration, which are often indispensable in the specification of open, extensible systems such as component-based ones. Passivation has been considered by higher-order calculi such as the Kell calculus [13] and Homer [14], and finds several applications (see, e.g., [15]). Here we shall consider a passivation construct of the form $\widetilde{a}\{P\}$, which represents a *passivation unit* named $a$ that contains a process $P$. The passivation unit is a *transparent locality*, in that there are no restrictions on the interactions between $P$ and processes surrounding $a$. The execution of $P$ can be *passivated* at an arbitrary time; this is represented by the evolution of $\widetilde{a}\{P\}$ into the nil process by means of an output action $\overline{a}\langle P\rangle$. Hence, the passivation of $\widetilde{a}\{P\}$ process might lead to a synchronization with any interacting input action on $a$.

We consider $\text{HOP}^{-\mathsf{f}}$, the extension of $\text{HO}^{-\mathsf{f}}$ with a passivation construct as described above. The syntax extends as expected; for the sake of consistency, we notice that the process $P$ in $\widetilde{a}\{P\}$ respects the limitation on the shape of output objects introduced for $\text{HO}^{-\mathsf{f}}$. The LTS for $\text{HOP}^{-\mathsf{f}}$ is the same as that for $\text{HO}^{-\mathsf{f}}$ in Section 2, extended with the two following rules which formalize the intuitions given before with respect to transparent localities and passivation, respectively:

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

### 5.1 A Deterministic Encoding of Minsky Machines into $\text{HOP}^{-\mathsf{f}}$

Here we investigate the expressiveness of $\text{HOP}^{-\mathsf{f}}$ by exhibiting an encoding of Minsky machines. Interestingly, unlike the encoding presented in Section 3, the encoding into $\text{HOP}^{-\mathsf{f}}$ is *deterministic*[2]. As such, in $\text{HOP}^{-\mathsf{f}}$ *both* termination and convergence are

[2] In fact, the encoding is *nearly-deterministic*, because of the encoding of replication we are considering.

REGISTER $r_k$ $\quad [\![r_k = m]\!]_{\mathsf{M}} = \widetilde{r_k}\{(\!|\ m\ |\!)_k\}$

where

$(\!|\ 0\ |\!)_k = z_k.\,\overline{a_z} \qquad (\!|\ n\ |\!)_k = u_k.\,(\overline{a_1} \parallel a_2.\,(\!|\ n-1\ |\!)_k)$

INSTRUCTIONS $(i : I_i)$

$[\![(i : \mathtt{INC}(r_k))]\!]_{\mathsf{M}} = !p_i.\,(r_k(x).\,(\overline{c_k}\langle x\rangle \parallel \widetilde{r_k}\{c_k(y).\,(\overline{a_p} \parallel u_k.\,(\overline{a_1} \parallel a_2.\,y))\} \parallel a_p.\,\overline{p_{i+1}}))$

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

**Table 3.** Encoding of Minsky machines into HOP$^{-\mathsf{f}}$.

*undecidable* problems. Hence, it is fair to say that the passivation construct —even with the limitation on the shape of (output) processes— allows to recover the expressive power lost in restricting HOCORE as HO$^{-\mathsf{f}}$.

The encoding is given in Table 3; we now give some intuitions on it. A register $k$ with value $m$ is represented by a passivation unit $r_k$ that contains the encoding of number $m$, denoted $(\!|\ m\ |\!)_k$. In turn, $(\!|\ m\ |\!)_k$ consists of a chain of $m$ nested input prefixes on name $u_k$; it also contains other prefixes on $a_1$ and $a_2$ which are used for synchronization purposes during the execution of instructions. The encoding of zero is given by an input action on $z_k$ that prefixes a trigger $\overline{a_z}$.

As expected, the encoding of an increment operation on the value of register $k$ consists in the enlargement of the chain of nested input prefixes it contains. For that purpose, the content of passivation unit $r_k$ is obtained with an input on $r_k$. We therefore need to recreate the passivation unit $r_k$ with the encoding of the incremented value. Notice that we require an additional synchronization on $c_k$ in order to "inject" such a previous content in a new passivation unit called $r_k$. This way, the the chain of nested inputs in $r_k$ can be enlarged while respecting the limitation on the shape of processes inside passivation units. As a result, the chain is enlarged by putting it behind some prefixes, and the next instruction can be invoked. This is done by a synchronization on name $a_p$.

The encoding of a decrement of the value of register $k$ consists of an internal, exclusive choice implemented as two passivation units that execute in parallel: the first one, named $d$, implements the behavior for decrementing the value of a register, while the second one, named $s$, implements the behavior for performing the jump to some given instruction. Unlike the encoding of Minsky machines in HO$^{-\mathsf{f}}$ presented in Section 3, this internal choice behaves faithfully with respect to the encoding instruction, i.e. the behavior inside $d$ will only execute if the value in $r_k$ is greater than zero, whereas the behavior inside $s$ will only execute if that value is equal to zero. It is indeed a deterministic choice in that it is *not* the case that both an input prefix on $u_k$ (which triggers the "decrement branch" defined in $d$) and one on $z_k$ (which triggers the "jump branch" defined in $s$) are available at the same time; this is because of the way in which we encode numbers, i.e. as a chain of input prefixes. In addition to the passivation units, the encoding of decrement features a "manager" (implemented as a synchronization on $m$) that

enables the behavior of the chosen passivation unit by placing it at the top-level, and consumes both $s$ and $d$ afterwards, thus leaving no residual processes after performing the instruction. In case the value of the register is equal to some $n > 0$, then a decrement is implemented by consuming the input prefixes on $u_k$ and $a_2$ and the output prefix on $a_1$ through suitable synchronizations. As a result, the encoding of $n - 1$ remains inside $r_k$ and the next instruction is invoked. In case the value of the register is equal to zero, the passivation unit $r_k$ is consumed and recreated with the encoding of zero inside. The jump is then performed by invoking the respective instruction.

We are now ready to define the encoding of a configuration of a Minsky machine into HOP$^{-\mathsf{f}}$.

**Definition 19 (Encoding of Configurations)** *Let $N$ be a Minsky machine with registers $r_0 = m_0$, $r_1 = m_1$ and instructions $(1 : I_1), \ldots, (n : I_n)$. The encoding of a configuration $(i, m_0, m_1)$ of $N$ into* HOP$^{-\mathsf{f}}$ *is defined by the encodings in Table 3 as*

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

*assuming fresh, pairwise different names $r_j$, $u_k$, $z_k$, $p_1, \ldots, p_n$, (for $j \in \{0, 1\}$).*

### 5.2 Correctness of the Encoding

We divide the proof of correctness into two properties: completeness (Lemma 13) and soundness (Lemma 14).

**Lemma 13 (Completeness).** *Let $(i, m_0, m_1)$ be a configuration of a Minsky machine $N$. Then, if $(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (i', m_0', m_1')$ then, for some finite $j$ and a process $P$, it holds that $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \longrightarrow^{j} P \equiv [\![(i', m_0', m_1')]\!]_{\mathsf{M}}$.*

*Proof.* We proceed by a case analysis on the instruction performed by the Minsky machine. Hence, we distinguish three cases corresponding to the behaviors associated to rules M-INC, M-DEC, and M-JMP. Without loss of generality, we restrict our analysis to operations on register $r_0$.

**Case M-INC:** We have a Minsky configuration $(i, m_0, m_1)$ with $(i : \mathtt{INC}(r_0))$. By Definition 19, its encoding into HO$^{-\mathsf{f}}$ with passivation is as follows:

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

After consuming the program counter $p_i$ we have the following

$$\begin{aligned}[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \longrightarrow \widetilde{r_0}\{(\!| \, m_0 \, |\!)_0\} \parallel r_0(x).\,(\overline{c_0}\langle x\rangle \parallel \widetilde{r_0}\{c_0(y).\,(\overline{a_p} \parallel u_0.\,(\overline{a_1} \parallel a_2.\,y))\}) \parallel \\ a_p.\,\overline{p_{i+1}} \parallel S = P_1\end{aligned}$$

where $S = [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathsf{M}}$ stands for the rest of the system. The only reduction possible at this point is the synchronization on $r_0$, which allows the content of the passivation unit $r_0$ to be communicated:

$$P_1 \longrightarrow \overline{c_0}\langle (\!| \, m_0 \, |\!)_0 \rangle \parallel \widetilde{r_0}\{c_0(y).\,(\overline{a_p} \parallel u_0.\,(\overline{a_1} \parallel a_2.\,y))\} \parallel a_p.\,\overline{p_{i+1}} \parallel S = P_2\,.$$

Now there is a synchronization on $c_0$, which allows to "inject" the encoding of value $m_0$ inside the passivation unit $r_0$ respecting the limitation of the language:

$$P_2 \longrightarrow \widetilde{r_0}\{(\overline{a_p} \parallel u_0.\,(\overline{a_1} \parallel a_2.\,(\!| \, m_0 \, |\!)_0))\} \parallel a_p.\,\overline{p_{i+1}} \parallel S = P_3\,.$$

The only possible synchronization from $P_3$ is the one on $a_p$, which works as an acknowledgment signal, and allows to release the program counter for instruction $i+1$. By performing such a synchronization, and by the encoding of numbers, we obtain the following

$$P_3 \longrightarrow \widetilde{r_0}\{(\!| \, m_0 + 1 \, |\!)_0\} \parallel \overline{p_{i+1}} \parallel S = P_4$$

It is then easy to see that $P_4 \equiv [\![(i+1, m_0+1, m_1)]\!]_{\mathsf{M}}$, as desired.

**Case M-DEC:** We have a Minsky configuration $(i, c, m_1)$ with $c > 0$ and $(i : \mathtt{DEC}(r_0, s))$. By Definition 19, its encoding into HO$^{-\mathsf{f}}$ with passivation is as follows:

$$[\![(i, c, m_1)]\!]_{\mathsf{M}} = \overline{p_i} \parallel [\![r_0 = c]\!]_{\mathsf{M}} \parallel [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel [\![(i : \mathtt{DEC}(r_0, s))]\!]_{\mathsf{M}} \parallel \prod_{l=1..n, l\neq i} [\![(l : I_l)]\!]_{\mathsf{M}}$$

We begin by consuming the program counter $p_i$, which leaves the content of $[\![(i : \mathtt{DEC}(r_0, s))]\!]_{\mathsf{M}}$ exposed. Using the encoding of numbers we have the following:

$$\begin{aligned}[\![(i, c, m_1)]\!]_{\mathsf{M}} \longrightarrow\ & \widetilde{r_0}\{u_0.\,(\overline{a_1} \parallel a_2.\,(\!| \, c-1 \, |\!)_0)\} \parallel m(x).\,x \parallel \\ & \widetilde{d}\{\overline{u_0} \parallel a_1.\,\overline{m}\langle s(x).\,d(x).\,(\overline{a_2} \parallel \overline{p_{i+1}})\rangle\} \parallel \\ & \widetilde{s}\{\overline{z_0} \parallel a_z.\,\overline{m}\langle d(x).\,s(x).\,r_0(t).\,(\widetilde{r_0}\{z_0.\,\overline{a_z}\} \parallel \overline{p_s})\rangle\} \parallel S = P_1\end{aligned}$$

where $S = [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathsf{M}}$ stands for the rest of the system. Notice that only reduction possible at this point is the synchronization on $u_0$, which signals the fact we are performing a decrement instruction. Such a synchronization enables one on $a_1$. After these two synchronizations we have

$$\begin{aligned}P_1 \longrightarrow^2\ & \widetilde{r_0}\{a_2.\,(\!| \, c-1 \, |\!)_0\} \parallel m(x).\,x \parallel \widetilde{d}\{\overline{m}\langle s(x).\,d(x).\,(\overline{a_2} \parallel \overline{p_{i+1}})\rangle\} \parallel \\ & \widetilde{s}\{\overline{z_0} \parallel a_z.\,\overline{m}\langle d(x).\,s(x).\,r_0(t).\,(\widetilde{r_0}\{z_0.\,\overline{a_z}\} \parallel \overline{p_s})\rangle\} \parallel S = P_2\,.\end{aligned}$$

Starting in $P_2$ the only reduction possible is due to the synchronization on $m$, which gives us the following:

$$\begin{aligned}P_2 \longrightarrow\ & \widetilde{r_0}\{a_2.\,(\!| \, c-1 \, |\!)_0\} \parallel s(x).\,d(x).\,(\overline{a_2} \parallel \overline{p_{i+1}}) \parallel \widetilde{d}\{\mathbf{0}\} \parallel \\ & \widetilde{s}\{\overline{z_0} \parallel a_z.\,\overline{m}\langle d(x).\,s(x).\,r_0(t).\,(\widetilde{r_0}\{z_0.\,\overline{a_z}\} \parallel \overline{p_s})\rangle\} \parallel S = P_3\,.\end{aligned}$$

In $P_3$ we have that passivation units $s$ and $d$ are consumed, thus we have:

$$P_3 \longrightarrow \widetilde{r_0}\{a_2.\, (\!|\, c-1 \,|\!)_0\} \parallel \overline{a_2} \parallel \overline{p_{i+1}} \parallel S = P_4\,.$$

At this point it is easy to see that, after a synchronization on $a_2$, we obtain

$$P_4 \longrightarrow\!\equiv [\![(i+1, c-1, m_1)]\!]_{\mathsf{M}}$$

as desired.

**Case M-JMP:** We have a Minsky configuration $(i, 0, m_1)$ and $(i : \mathtt{DEC}(r_0, s))$. By Definition 19, its encoding into $\mathrm{HO}^{-\mathsf{f}}$ with passivation is as follows:

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

We begin by consuming the program counter $p_i$, which leaves the content of $[\![(i : \mathtt{DEC}(r_0, s))]\!]_{\mathsf{M}}$ exposed. Using the encoding of numbers we have the following:

$$\begin{aligned}[\![(i,0,m_1)]\!]_{\mathsf{M}} \longrightarrow\ & \widetilde{r_0}\{z_0.\, a_z\} \parallel m(x).\, x \parallel \\ & \widetilde{d}\{\overline{u_0} \parallel a_1.\, \overline{m}\langle s(x).\, d(x).\, (\overline{a_2} \parallel \overline{p_{i+1}})\rangle\} \parallel \\ & \widetilde{s}\{\overline{z_0} \parallel a_z.\, \overline{m}\langle d(x).\, s(x).\, r_0(t).\, (\widetilde{r_0}\{z_0.\, \overline{a_z}\} \parallel \overline{p_s})\rangle\} \parallel S = P_1\end{aligned}$$

where $S = [\![r_1 = m_1]\!]_{\mathsf{M}} \parallel \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathsf{M}}$ stands for the rest of the system. In $P_1$, the only reduction possible is through a synchronization on $z_0$, which signals the fact we are performing a jump. Such a synchronization, in turn, enables one on $a_z$. We then have:

$$\begin{aligned}P_1 \longrightarrow^2\ & \widetilde{r_0}\{\mathbf{0}\} \parallel m(x).\, x \parallel \\ & \widetilde{d}\{\overline{u_0} \parallel a_1.\, \overline{m}\langle s(x).\, d(x).\, (\overline{a_2} \parallel \overline{p_{i+1}})\rangle\} \parallel \\ & \widetilde{s}\{\overline{m}\langle d(x).\, s(x).\, r_0(t).\, (\widetilde{r_0}\{z_0.\, \overline{a_z}\} \parallel \overline{p_s})\rangle\} \parallel S = P_2\,.\end{aligned}$$

The only possible reduction from $P_2$ is by means of a synchronization on $m$. This gives us:

$$\begin{aligned}P_2 \longrightarrow\ & \widetilde{r_0}\{\mathbf{0}\} \parallel d(x).\, s(x).\, r_0(t).\, (\widetilde{r_0}\{z_0.\, \overline{a_z}\} \parallel \overline{p_s}) \parallel \\ & \widetilde{d}\{\overline{u_0} \parallel a_1.\, \overline{m}\langle s(x).\, d(x).\, (\overline{a_2} \parallel \overline{p_{i+1}})\rangle\} \parallel \widetilde{s}\{\mathbf{0}\} \parallel S = P_3\,.\end{aligned}$$

In $P_3$ the two passivation units on $d$ and $s$ are consumed, which gives us:

$$P_3 \longrightarrow \widetilde{r_0}\{\mathbf{0}\} \parallel r_0(t).\, (\widetilde{r_0}\{z_0.\, \overline{a_z}\} \parallel \overline{p_s}) \parallel S = P_4\,.$$

At this point, it is easy to see that after a synchronization on $r_0$ we obtain:

$$P_4 \longrightarrow\!\equiv [\![(s, 0, m_1)]\!]_{\mathsf{M}}$$

as desired.

**Lemma 14 (Soundness).** *Let $(i, m_0, m_1)$ be a configuration of a Minsky machine $N$. If $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \longrightarrow P_1$ then for every computation of $P_1$ there exists a $P_j$ such that $P_j = [\![(i', m'_0, m'_1)]\!]_{\mathsf{M}}$ and $(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (i', m'_0, m'_1)$.*

*Proof.* Consider the reduction $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \longrightarrow P_1$. An analysis of the structure of process $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$ reveals that, in all cases, the only possibility for 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.

In all cases, it can be shown that computation evolves (nearly) deterministically, until reaching a process in which a new program counter (that is, some $\overline{p_{i'}}$) appears. To be precise, there is some non-determinism associated with the unfoldings of the encoding of recursive definitions (see Lemma 4). Since recursion is guarded, each recursive definition can be unfolded at most once, thus introducing at most a finite number of additional reductions. The program counter $\overline{p_{i'}}$ is inside a process that corresponds to $[\![(i', m'_0, m'_1)]\!]_{\mathsf{M}}$, 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 Lemma 13, and we omit it.

**Corollary 7.** *Let $N$ be a Minsky machine. We have that $N \nrightarrow_M$ if and only if $[\![N]\!]_{\mathsf{M}} \nrightarrow$.*

*Proof.* Straightforward from Lemmas 13 and 14.

**Lemma 15.** *Termination and convergence are undecidable in* $\text{HOP}^{-\mathsf{f}}$.

*Proof.* This is an immediate consequence of previous results (Lemmas 13 and 14, Corollary 7).

## 6 Concluding Remarks

In this chaper we have studied the expressiveness and decidability of higher-order process calculi featuring *limited forwarding*. Our study has been centered around $\text{HO}^{-\mathsf{f}}$, the fragment of HOCORE in which output actions can only include previously received processes in composition with closed ones. This communication style is reminiscent of programming scenarios with forms of code mobility in which the recipient is not authorized or capable of accessing/modifying the structure of the received code. We have shown that such a weakening of the forward capabilities of higher-order processes has consequences both on the expressiveness of the language and on the decidability of termination. Furthermore, we analyzed the extension of $\text{HO}^{-\mathsf{f}}$ with a *passivation* operator as a way of recovering the expressive power lost when moving from HOCORE to $\text{HO}^{-\mathsf{f}}$.

By exhibiting an encoding of Minsky machines into $\text{HO}^{-\mathsf{f}}$, we have shown that convergence is undecidable. Hence, from an *absolute expressiveness* standpoint, $\text{HO}^{-\mathsf{f}}$ is Turing complete. Now, given the analogous result for HOCORE [4], a *relative expressiveness* issue also arises. Indeed, our encoding of Minsky machines into $\text{HO}^{-\mathsf{f}}$ is not faithful, which reveals a difference on the criteria each encoding satisfies. This reminds us of the situation in [16], 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 conciseness, we do not elaborate further on their exact definition; using the terminology in [16], here 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 $\text{HO}^{-\text{f}}$ 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.

We have shown that 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, for which 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 shown to be essential for obtaining the decidability result; for this, we appealed to the approach developed in [11], which relies on the theory of well-structured transition systems [8,9,10]. As far as we are aware, this approach to studying expressiveness issues 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.

We have also studied the expressiveness and decidability of $\text{HOP}^{-\text{f}}$, the extension of $\text{HO}^{-\text{f}}$ with a passivation operator. To the best of our knowledge, this is the first expressiveness study involving passivation operators in the context of higher-order process calculi. In $\text{HOP}^{-\text{f}}$ it is possible to encode Minsky machines in a *faithful* manner. Hence, similarly as in HOCORE, in $\text{HOP}^{-\text{f}}$ both termination and convergence are undecidable. This certainly does not imply that both languages have the same expressive power; in fact, an interesting direction for future work consists in assessing the exact expressive power that passivation brings into the picture. This would include not only a comparison between $\text{HOP}^{-\text{f}}$ and HOCORE, but also a comparison between HOCORE and HOCORE extended with passivation. All the languages involved are Turing complete, hence such comparisons should employ techniques different from the ones used here. Related to this, it is worth remarking that we have considered a very simple form of passivation, one in which process suspension takes place with a considerable degree of non-determinism. Studying other forms of passivation, possibly with more explicit control mechanisms, could be interesting from several points of view, including expressiveness.

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 shape 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, our undecidability result can be used to prove that (weak) barbed bisimilarity is undecidable in the 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 $\widetilde{x}$. Take a Minsky machine $N$ (it is not restrictive to assume that it executes at least one increment instruction) and its encoding $P$, as given by Definition 5. Let $\widetilde{x}$ be the tuple of the names used by $P$, excluding the name $w$. We have that $N$ terminates if and only if $(\nu\widetilde{x})P$ is (weakly) barbed equivalent to the process $(\nu d)(\overline{d} \mid d \mid d.\,(\overline{w} \mid !w.\,\overline{w}))$.

**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 [17] studies finite-control fragments of Homer [18], a higher-order process calculus with locations. While we have focused on decidability of termination and convergence, in [17] the interest is in decidability of barbed bisimilarity. One of the approaches explored in [17] is based on a type system that bounds the size of processes in terms of their syntactic components (e.g. 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 with $\text{HO}^{-\text{f}}$.

Also similar in spirit to our work, but in a slightly different context, are some studies on the expressiveness (of fragments) of the Ambient calculus [19]. Ambient and higher-order calculi are related 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, [20] shows that termination is decidable for the fragment without both restriction (as $\text{HO}^{-\text{f}}$ and HOCORE) and movement capabilities, and featuring replication; in contrast, the same property turns out to be undecidable for the fragment with recursion. Hence, the separation between fragments comes from the source of infinite behavior, and not from the structures allowed in output action, as in our case. However, we find that the connections between Ambient-like and higher-order calculi are rather loose, so a proper comparison is difficult also in this case.

**Acknowledgments.** We are grateful to Julian Gutierrez and Roland Meyer for their useful remarks on a previous version of this paper.

## 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
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., Low, D.: Manufacturing cheap, resilient, and stealthy opaque constructs. In: Proc. of POPL'98, ACM Press (1998) 184–196
8. Finkel, A.: Reduction and covering of infinite reachability trees. Inf. Comput. **89**(2) (1990) 144–179
9. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.K.: Algorithmic analysis of programs with well quasi-ordered domains. Inf. Comput. **160**(1-2) (2000) 109–127
10. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comput. Sci. **256**(1-2) (2001) 63–92
11. Busi, N., Gabbrielli, M., Zavattaro, G.: On the expressive power of recursion, replication, and iteration in process calculi. Math. Struct. in Comp. Sci. (2009) To appear.
12. Higman, G.: Ordering by divisibility in abstract algebras. Proceedings of the London Mathematical Society (3) **2**(7) (1952) 326–336
13. Schmitt, A., Stefani, J.B.: The kell calculus: A family of higher-order distributed process calculi. In: Proc. of Global Computing. Volume 3267 of Lecture Notes in Computer Science., Springer (2004) 146–178
14. Hildebrandt, T., Godskesen, J.C., Bundgaard, M.: Bisimulation congruences for homer — a calculus of higher order mobile embedded resources. Technical Report TR-2004-52, IT University of Copenhagen (2004)
15. Bundgaard, M., Glenstrup, A.J., Hildebrandt, T.T., Højsgaard, E., Niss, H.: Formalizing higher-order mobile embedded business processes with binding bigraphs. In: Proc. of COORDINATION. Volume 5052 of Lecture Notes in Computer Science., Springer (2008) 83–99
16. Bravetti, M., Zavattaro, G.: On the expressive power of process interruption and compensation. Math. Struct. in Comp. Sci. **19**(3) (2009) 565–599
17. Bundgaard, M., Godskesen, J.C., Haagensen, B., Hüttel, H.: Decidable fragments of a higher order calculus with locations. Electr. Notes Theor. Comput. Sci. **242**(1) (2009) 113–138
18. Bundgaard, M., Godskesen, J.C., Hildebrandt, T.: Bisimulation congruences for homer — a calculus of higher order mobile embedded resources. Technical Report TR-2004-52, IT University of Copenhagen (2004)
19. Cardelli, L., Gordon, A.D.: Mobile ambients. Theor. Comput. Sci. **240**(1) (2000) 177–213
20. Busi, N., Zavattaro, G.: On the expressive power of movement and restriction in pure mobile ambients. Theor. Comput. Sci. **322**(3) (2004) 477–515