Reversibility in the higher-order π-calculus

Ivan Lanese, Claudio Antares Mezzina, Jean-Bernard Stefani

Reversibility in the higher-order π-calculus

Ivan Lanese[a,1], Claudio Antares Mezzina[b], Jean-Bernard Stefani[c,2,1]

[a] *Focus Team, University of Bologna/INRIA, Italy*
[b] *SOA Unit, FBK, Trento, Italy*
[c] *INRIA, France*

Abstract

The notion of reversible computation is attracting increasing interest because of its applications in diverse fields, in particular the study of programming abstractions for reliable systems. In this paper, we continue the study undertaken by Danos and Krivine on reversible CCS by defining a reversible higher-order π-calculus, called rhoπ. We prove that reversibility in our calculus is causally consistent and that the causal information used to support reversibility in rhoπ is consistent with the one used in the causal semantics of the π-calculus developed by Boreale and Sangiorgi. Finally, we show that one can faithfully encode rhoπ into a variant of higher-order π, substantially improving on the result we obtained in the conference version of this paper.

Keywords: reversible computation, process algebra, π-calculus

1. Introduction

Motivation. The notion of reversible computation has already a long history [1]. It has its origin in physics with the observation by Landauer that only irreversible computations need to consume energy [2]. It has since attracted interest in diverse fields, including e.g. hardware design [3], computational biology [4], program debugging [5], and quantum computing [6]. Of particular interest is its application to the study of programming abstractions for reliable systems. For instance, Bishop advocates using reversible computing as a means to achieve *fail-safety* in a sequential setting [7]. Moreover,

[1]Partly funded by the ANR-11-INSE-007 project REVER.
[2]Partly funded by the ANR-2010-BLAN-0305-01 project PiCoq.

most fault-tolerant schemes exploiting system recovery techniques [8], including exception handling [9], checkpoint/rollback [10] and transaction management [11], rely on some form of *undo* or another. All this suggests it can be worthwhile to formally investigate reversibility in concurrent systems, with the hope that reversible computing ideas can lead us to more systematic and more composable abstractions for the design and construction of recoverable systems.

The seminal work of Danos and Krivine on reversible CCS (RCCS) constitutes a first step on this research programme. They develop in [12] a basic reversible concurrent programming model in the form of a reversible variant of CCS, and introduce the notion of *causal consistency* as the least constraining correctness criterion for reversing a concurrent system. Requiring a concurrent system to go back in a computation by undoing actions in the inverse order with respect to the one described by its interleaving semantics for the forward computation is too restrictive, since forward actions could have executed concurrently. Causality constraints should be respected, however: first the consequences have to be undone, then the causes. Causal consistency captures exactly this: when reversing a computation, actions are undone in reverse order up to causal equivalence, i.e. up to swaps of concurrent actions. In [13] Danos and Krivine show how to leverage RCCS for the design of transactional systems. They provide an interpretation of distributed transactions as "ballistic" processes, that freely explore the state space defined by their reversible part until they commit by performing irreversible actions, and they show how reversibility can help in the design and proof of correctness of complex transactional systems. Later on, Phillips and Ulidowski [14] showed how to devise causally consistent reversible extensions to process calculi specified by SOS rules in a subset of the path format.

A reversible CCS, or a process calculus defined by operators in the path format, remains limited as a reversible programming model, however. The same reasons that motivated the introduction of the π-calculus [15] apply to motivate the study of a reversible π-calculus. As a first contribution in that study, we introduced in [16] a causally-consistent reversible extension of an asynchronous variant of the higher-order π-calculus [17], where we showed how to preserve the usual properties of the π-calculus operators (e.g. associativity and commutativity of parallel composition), and that one could faithfully encode (up to weak barbed bisimilarity) our reversible higher-order π, called $\mathsf{rho}\pi$, into a variant of the higher-order π-calculus with abstractions and join patterns. The operational semantics of $\mathsf{rho}\pi$ was given in [16] by way of a reduction semantics. We then showed how to leverage the reversible machinery in rhoπ for a calculus with explicit rollback [18] called rollπ, and further, building on rollπ, how to faithfully encode certain kinds of communicating transactions [19]. Very recently, Cristescu, Krivine and Varacca have proposed in [20] a reversible π-calculus, called Rπ, and defined a labelled transition system semantics for it. In addition, they showed their LTS semantics for Rπ to be as liberal as possible in the sense that the causality relation between labelled transitions is the smallest relation that is consistent with the structural causality between reductions.

Contributions. In this paper, we revisit our work on rhoπ. Apart from providing proofs that where omitted from [16] for lack of space, we discuss in more detail notions of barbed bisimilarity for rhoπ, and we study the relationship between the notion of causality that emerges from our reversibility machinery and that introduced by Boreale and Sangiorgi in their causal π-calculus [21]. Our discussion of barbed bisimilarity shows that the usual notion of weak barbed bisimilarity is very coarse in rhoπ since it identifies processes that have the same weak observables. Weak barbed bisimilarity remains a non-trivial relation since the question of knowing whether two rhoπ processes have the same weak observables is undecidable. However, since it was used to show the faithfulness of our encoding of rhoπ in higher-order π, one can wonder whether the result would still hold with a finer bisimilarity, in particular one that could distinguish between forward and backward reductions. We show in this paper that this is indeed the case, at the cost of minor tweaks in our encoding, and at the expense of a much more complex proof.

Outline. The paper is organized as follows. Section 2 defines the rhoπ calculus. We explain the main constructions of the calculus and we contrast our way of handling reversibility with that of Danos and Krivine. We also define and discuss barbed equivalences in rhoπ. Section 3 is devoted to the proof of our first main result, namely that reversibility in rhoπ proceeds in a causally consistent way. We also show that the notion of causality built in the rhoπ operational semantics agrees with that of Boreale and Sangiorgi. Section 4 presents a compositional encoding of the rhoπ calculus into a variant of HOπ and proves its faithfulness. Section 5 discusses related work. Section 6 concludes the paper. Main proofs and auxiliary results are collected in Appendix.

This paper constitutes a revised and extended version of our conference paper [16]. While the $\mathsf{rho}\pi$ calculus and its reversible machinery are unchanged, the analysis in Section 3 of the relationship between our notion of causality and that provided by Boreale and Sangiorgi for the π-calculus is new. The material in Section 4, which makes up the bulk of the paper, is entirely new.

2. The $\mathsf{rho}\pi$ calculus

2.1. Informal presentation

Building a reversible variant of a process calculus involves devising appropriate syntactic representations for computation histories. As already hinted at in the Introduction, since a process models a concurrent system, asking for a deterministic reverse execution, which traverses the exact same states of the forward execution, is too restrictive. In fact, those states depend on the chosen interleaving of concurrent actions. An approach more suitable for a concurrent system is *causally consistent* reversibility, where states that are reached during a backward computation are states that could have been reached during the computation history by just performing independent actions in a different order. In RCCS, Danos and Krivine achieve this with CCS without recursion by attaching a memory m to each process P, in the monitored process construct $m : P$. A memory in RCCS is a stack of information needed for processes to backtrack. Thus, if two processes P_1 and P_2 can synchronize on a channel a in order to evolve into P_1' and P_2', respectively, (e.g., $P_1 = a.P_1'$ and $P_2 = \overline{a}.P_2'$) then the parallel composition of monitored processes $m_1 : (P_1 + Q_1)$ and $m_2 : (P_2 + Q_2)$ can evolve, according to RCCS semantics, as follows:

$$m_1 : (P_1 + Q_1) \mid m_2 : (P_2 + Q_2) \to \langle m_2, a, Q_1\rangle \cdot m_1 : P_1' \mid \langle m_1, \overline{a}, Q_2\rangle \cdot m_2 : P_2'$$

In the reduction above, a memory of the form $\langle m_2, a, Q_1\rangle \cdot m_1$ represents the fact that its monitored process has performed an input on the channel a ($\overline{a}$ represents an output), interacting with a process monitored by the memory m_2, and discarded the alternative process Q_1. By exploiting all the information stored in the memories, the above synchronization can be reverted as follows:

$$\langle m_2, a, Q_1\rangle \cdot m_1 : P_1' \mid \langle m_1, \overline{a}, Q_2\rangle \cdot m_2 : P_2' \to m_1 : (P_1 + Q_1) \mid m_2 : (P_2 + Q_2)$$

Additionally, Danos and Krivine rely on the following rule:

$$m : (P \mid Q) \equiv \langle 1\rangle \cdot m : P \mid \langle 2\rangle \cdot m : Q$$

so as to ensure that each primitive thread, i.e. some process with no parallel composition at top level, gets its own unique identity. Since this rule stores the exact position of a process in a parallel composition, it is not compatible with the usual structural congruence rules for the parallel operator, namely associativity, commutativity, and $\mathbf{0}$ as neutral element. Danos and Krivine suggest that it could be possible to work up to tree isomorphisms on memories, but this would indeed lead to a more complex syntax, as well as additional difficulties (see Remark 5).

We adopt for $\mathsf{rho}\pi$ a different approach: instead of associating each thread with a stack that records, essentially, past actions and positions in parallel branches, we rely on simple thread tags, which act as unique identifiers but have little structure, and on new process terms, which we call *memories*, which are dedicated to undoing a single (forward) computation step.

More precisely, a *forward* computation step in $\mathsf{rho}\pi$ (denoted by arrow $\twoheadrightarrow$) consists in the receipt of a message ($\mathsf{rho}\pi$ is an asynchronous calculus). The receipt of a message $a\langle P\rangle$ on channel a by a receiver process (or *trigger*) $a(X) \triangleright Q$ takes in $\mathsf{rho}\pi$ the following form:

$$(\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q) \;\twoheadrightarrow\; \nu k.\, k : Q\{^P/_X\} \mid [M; k]$$

Each thread (message and trigger) participating in the above computation step is uniquely identified by a tag: κ_1 identifies the message $a\langle P\rangle$, and κ_2 identifies the trigger $a(X) \triangleright Q$. The result of the message receipt consists in a classical part and two side effects. The classical part is the launch of an instance $Q\{^P/_X\}$ of the body of the trigger Q, with the formal parameter X instantiated by the received value, i.e. the process P ($\mathsf{rho}\pi$ is a higher-order calculus). The two side effects are: (i) the tagging of the newly created process $Q\{^P/_X\}$ by a fresh new key k (ν is the standard restriction operator of the π-calculus), and (ii) the creation of a memory process $[M; k]$. M is simply the configuration on the left hand side of the reduction, namely $M = (\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q)$.

In this setting, a *backward* computation step takes the form of an interaction between a memory and a process tagged with the appropriate key: when a memory $[M; k]$ is put in presence of a process tagged with k, a *backward* reduction (denoted by arrow $\rightsquigarrow$) can take place. Such a reduction kills the process tagged with k and reinstates the configuration M:

$$(k : P) \;\mid\; [M; k] \;\rightsquigarrow\; M$$

We thus have:

$$M \twoheadrightarrow \nu k.\, k : Q\{^{P}/_{X}\} \mid [M; k] \rightsquigarrow \nu k.\, M$$

Since k is fresh, $\nu k.\, M$ is actually structurally equivalent to M. We thus have a perfect reversal of a forward computation: $M \twoheadrightarrow\rightsquigarrow M$.

Remark 1. Following Danos and Krivine [13], one could consider also taking into account *irreversible* actions. We do not do so in this paper for the sake of simplicity. Adding irreversible actions to $\mathsf{rho}\pi$ would be conceptually straightforward.

Remark 2. Using memories as presented here to enable reversibility simplifies the formal development but leads to a space explosion of computations in $\mathsf{rho}\pi$. We do not consider implementation and related space efficiency issues in this paper. This issue has been analysed in [22] in the context of the Oz language.

2.2. Syntax

Names, keys, and variables. We assume the existence of the following denumerable infinite mutually disjoint sets: the set $\mathcal{N}$ of *names*, the set $\mathcal{K}$ of *keys*, and the set $\mathcal{V}$ of *process variables*. The set $\mathcal{I} = \mathcal{N} \cup \mathcal{K}$ is called the set of *identifiers*. $\mathbb{N}$ denotes the set of natural integers. We let (together with their decorated variants): a, b, c range over $\mathcal{N}$; h, k, l range over $\mathcal{K}$; u, v, w range over $\mathcal{I}$; X, Y, Z range over $\mathcal{V}$. We denote by $\tilde{u}$ a finite set of identifiers $\{u_1, \ldots, u_n\}$.

Syntax. The syntax of the $\mathsf{rho}\pi$ calculus is given in Figure 1 (in writing $\mathsf{rho}\pi$ terms, we freely add balanced parenthesis around terms to disambiguate them). *Processes* of the $\mathsf{rho}\pi$ calculus, given by the P, Q productions in Figure 1, are the standard processes of the asynchronous higher-order π-calculus [23]. A receiver process (or *trigger*) in $\mathsf{rho}\pi$ takes the form $a(X) \triangleright P$, which allows the receipt of a message of the form $a\langle Q \rangle$ on channel a.

Processes in $\mathsf{rho}\pi$ cannot directly execute, only *configurations* can. *Configurations* in $\mathsf{rho}\pi$ are given by the M, N productions in Figure 1. A configuration is built up from *threads* and *memories*.

A *thread* $\kappa : P$ is just a tagged process P, where the tag κ is either a single key k or a pair of the form $\langle h, \tilde{h} \rangle \cdot k$, where $\tilde{h}$ is a set of keys, with $h \in \tilde{h}$. A tag serves as an identifier for a process. As we will see below, together with memories tags help capture the flow of causality in a computation.

$$
\begin{aligned}
P, Q &::= \mathbf{0} \mid X \mid \nu a.\, P \mid (P \mid Q) \mid a\langle P\rangle \mid a(X) \triangleright P \\
M, N &::= \mathbf{0} \mid \nu u.\, M \mid (M \mid N) \mid \kappa : P \mid [\mu; k] \\
\kappa &::= k \mid \langle h, \tilde{h}\rangle \cdot k \\
\mu &::= ((\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q)) \\
& u \in \mathcal{I} \quad a \in \mathcal{N} \quad X \in \mathcal{V} \quad h, k \in \mathcal{K} \quad \kappa \in \mathcal{T}
\end{aligned}
$$

Figure 1: Syntax of $\mathsf{rho}\pi$

A *memory* is a process of the form $[\mu; k]$, which keeps track of the fact that a configuration μ was reached during execution, that triggered the launch of a thread tagged with the fresh tag k. In a memory $[\mu; k]$, we call μ the *configuration part* of the memory, and k the *thread tag* of the memory. Memories are generated by computation steps and are used to reverse them. The configuration part $\mu = (\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q)$ of the memory records the message $a\langle P\rangle$ and the trigger $a(X) \triangleright Q$ involved in the message receipt, together with their respective thread tags κ_1, κ_2.

$\mathcal{P}$ denotes the set of $\mathsf{rho}\pi$ processes, and $\mathcal{C}$ the set of $\mathsf{rho}\pi$ configurations. We call *agent* an element of the set $\mathcal{A} = \mathcal{P} \cup \mathcal{C}$. We let (together with their decorated variants) P, Q, R range over $\mathcal{P}$; L, M, N range over $\mathcal{C}$; and A, B, C range over agents. We call *primitive thread process*, a process that is either a message $a\langle P\rangle$ or a trigger $a(X) \triangleright P$. We let τ and its decorated variants range over primitive thread processes.

Remark 3. We have no construct for replicated processes, output prefixing, or guarded choice in $\mathsf{rho}\pi$: as in the asynchronous HOπ, also in $\mathsf{rho}\pi$ these can be easily encoded.

Free names and free variables. Notions of free identifiers and free (process) variables in $\mathsf{rho}\pi$ are classical. It suffices to note that constructs with binders are of the forms: $\nu a.\, P$, which binds the name a with scope P; $\nu u.\, M$, which binds the identifier u with scope M; and $a(X) \triangleright P$, which binds the variable X with scope P. We denote by $\mathtt{fn}(P)$, $\mathtt{fn}(M)$ and $\mathtt{fn}(\kappa)$ the set of free names, free identifiers, and free keys, respectively, of process P, of configuration M, and of tag κ. Note in particular that $\mathtt{fn}(\kappa : P) = \mathtt{fn}(\kappa) \cup \mathtt{fn}(P)$, $\mathtt{fn}(k) = \{k\}$ and $\mathtt{fn}(\langle h, \tilde{h}\rangle \cdot k) = \tilde{h} \cup \{k\}$. We say that a process P or a configuration M is closed if it has no free (process) variable. $\mathcal{P}^\bullet$ denotes the set of closed processes, $\mathcal{C}^\bullet$ the set of closed configurations, and $\mathcal{A}^\bullet$ the set of closed agents.

Remark 4. In the remainder of the paper, we adopt *Barendregt's Variable Convention*: If terms $t_1, \ldots, t_n$ occur in a certain context (e.g. definition, proof), then in these terms all bound identifiers and variables are chosen to be different from the free ones.

Consistent configurations. Not all configurations allowed by the syntax in Figure 1 are meaningful. In a memory $[\mu; k]$, tags occurring in the configuration part μ must be different from the thread tag k. This is because the key k is freshly generated when a computation step (a message receipt) takes place, and it is used to identify the newly created thread. Tags appearing in the configuration part identify threads (message and trigger) which have participated in the computation step. In a configuration M, we require all the threads to be uniquely identified by their tag, and we require consistency between threads and memories: if M contains a memory $[\mu; k]$ (i.e. $[\mu; k]$ occurs as a subterm of M), we require M to also contain a thread tagged with k: components of this thread, i.e. threads whose tags have k as a suffix, can occur either directly in parallel with $[\mu; k]$ or in the configuration part of another memory contained in M (because they may have interacted with other threads). We call *consistent* a configuration that obeys these constraints. We defer the formal definition of consistent configurations to Section 2.3.

2.3. Operational semantics

The operational semantics of the $\mathsf{rho}\pi$ calculus is defined via a reduction relation $\to$, which is a binary relation over closed configurations $\to \subset \mathcal{C}^\bullet \times \mathcal{C}^\bullet$, and a structural congruence relation $\equiv$, which is a binary relation over processes and configurations $\equiv \subset \mathcal{P}^2 \cup \mathcal{C}^2$. We define evaluation contexts as "configurations with a hole $\cdot$" given by the following grammar:

$$\mathbb{E} ::= \cdot \quad | \quad (M \mid \mathbb{E}) \quad | \quad \nu u.\, \mathbb{E}$$

General contexts $\mathbb{C}$ are just processes or configurations with a hole replacing a $\mathbf{0}$ (process or configuration). A congruence on processes and configurations is an equivalence relation $\mathcal{R}$ that is closed for general contexts: $P \,\mathcal{R}\, Q \implies \mathbb{C}[P] \,\mathcal{R}\, \mathbb{C}[Q]$ and $M \,\mathcal{R}\, N \implies \mathbb{C}[M] \,\mathcal{R}\, \mathbb{C}[N]$.

The relation $\equiv$ is defined as the smallest congruence on processes and configurations that satisfies the rules in Figure 2. We write $t =_\alpha t'$ when terms t, t' are equal modulo α-conversion. If $\tilde{u} = \{u_1, \ldots, u_n\}$, then $\nu\tilde{u}.\, A$ stands for $\nu u_1. \ldots \nu u_n.\, A$ (there is no need to indicate the order of binders

$$(\text{E.ParC})\ A \mid B \equiv B \mid A \qquad (\text{E.ParA})\ A \mid (B \mid C) \equiv (A \mid B) \mid C$$

$$(\text{E.NilM})\ A \mid \mathbf{0} \equiv A \qquad (\text{E.NewN})\ \nu u.\, \mathbf{0} \equiv \mathbf{0} \qquad (\text{E.NewC})\ \nu u.\, \nu v.\, A \equiv \nu v.\, \nu u.\, A$$

$$(\text{E.NewP})\ (\nu u.\, A) \mid B \equiv \nu u.\, (A \mid B) \qquad (\text{E.}\alpha)\ A =_\alpha B \implies A \equiv B$$

$$(\text{E.TagN})\ \kappa : \nu a.\, P \equiv \nu a.\, \kappa : P$$

$$(\text{E.TagP})\ k : \prod_{i=1}^{n} \tau_i \equiv \nu \tilde{h}.\, \prod_{i=1}^{n} (\langle h_i, \tilde{h} \rangle \cdot k : \tau_i) \quad \tilde{h} = \{h_1, \ldots, h_n\} \quad n > 1$$

Figure 2: Structural congruence for rhoπ

thanks to rule E.NewC). We write $\prod_{i=1}^{n} A_i$ for $A_1 \mid \ldots \mid A_n$ (as before, there is no need to indicate how the latter expression is parenthesized because the parallel operator is associative by rule E.ParA). In rule E.TagP, processes τ_i are primitive thread processes (i.e., messages or triggers). Recall the use of the variable convention in these rules: for instance, in the rule $(\nu u.\, A) \mid B \equiv \nu u.\, (A \mid B)$ the variable convention makes implicit the condition $u \notin \mathtt{fn}(B)$. The structural congruence rules are the usual rules for the π-calculus (E.ParC to E.α) without the rule dealing with replication, and with the addition of two new rules dealing with tags: E.TagN and E.TagP. Rule E.TagN is a scope extrusion rule to push restrictions to the top level. Rule E.TagP allows to generate unique tags for each primitive thread process in a configuration. An easy induction on the structure of terms provides us with a kind of normal form for configurations (by convention $\prod_{i \in I} A_i = \mathbf{0}$ if $I = \emptyset$):

Lemma 1 (Thread normal form). *For any closed configuration M, we have:*

$$M \equiv \nu \tilde{u}.\, \prod_{i \in I} (\kappa_i : \rho_i) \mid \prod_{j \in J} [\mu_j; k_j]$$

with $\rho_i = \mathbf{0}$, $\rho_i = a_i \langle P_i \rangle$, or $\rho_i = a_i(X_i) \triangleright P_i$.

We say that a binary relation $\mathcal{R}$ on closed configurations is *evaluation-closed* if it satisfies the inference rules:

$$(\text{R.Ctx})\ \frac{M \mathrel{\mathcal{R}} N}{\mathbb{E}[M] \mathrel{\mathcal{R}} \mathbb{E}[N]} \qquad (\text{R.Eqv})\ \frac{M \equiv M' \qquad M' \mathrel{\mathcal{R}} N' \qquad N' \equiv N}{M \mathrel{\mathcal{R}} N}$$

$$(\text{R.Fw}) \quad (\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q) \twoheadrightarrow \nu k.\, (k : Q\{^P/_X\}) \mid [(\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q); k]$$

$$(\text{R.Bw}) \quad (k : P) \mid [M; k] \rightsquigarrow M$$

Figure 3: Reduction rules for $\mathsf{rho}\pi$

The reduction relation $\rightarrow$ is defined as the union of two relations, the *forward* reduction relation $\twoheadrightarrow$ and the *backward* reduction relation $\rightsquigarrow$: $\rightarrow = \twoheadrightarrow \cup \rightsquigarrow$. Relations $\twoheadrightarrow$ and $\rightsquigarrow$ are defined to be the smallest evaluation-closed binary relations on closed configurations satisfying the rules in Figure 3 (note again the use of the variable convention: in rule R.Fw the key k is fresh).

The rule for forward reduction (R.Fw) is the standard communication rule of the higher-order π-calculus with two side effects: (i) the creation of a new memory to record the configuration that gave rise to it, namely the parallel composition of a message and a trigger, properly tagged (tags κ_1 and κ_2 in the rule); (ii) the tagging of the continuation of the message receipt with the fresh key k. The rule for backward reduction (R.Bw) is straightforward: in presence of the thread tagged with key k, memory $[M; k]$ reinstates the configuration M that gave rise to the tagged thread. We use $\twoheadrightarrow^*$, $\rightsquigarrow^*$ and $\Rightarrow$ to denote the reflexive and transitive closure of $\twoheadrightarrow$, $\rightsquigarrow$ and $\rightarrow$, respectively. With the reduction rules and the structural laws in place, we can see how the structural rule E.TagP is used by the reduction. In particular the rule, if used from left to right after a forward step, lets the continuation of a trigger (if it is a parallel composition) continue executing in the forward direction. On the other side, when used from right to left, E.TagP gathers back all the primitive thread processes belonging to the same parallel composition identified by a particular key. An example of execution will make it clear. Let $M = (k_1 : a\langle P\rangle) \mid (k_2 : a(X) \triangleright b\langle X\rangle \mid b(X) \triangleright \mathbf{0})$, we have that:

$$\begin{aligned}
&M \twoheadrightarrow \nu k.\, k : (b\langle P\rangle \mid b(X) \triangleright \mathbf{0}) \mid [M; k] && (1)\\
&\equiv \nu k, h_1, h_2.\, (\langle h_1, \tilde{h}\rangle \cdot k : b\langle P\rangle) \mid (\langle h_2, \tilde{h}\rangle \cdot k : b(X) \triangleright \mathbf{0}) \mid [M; k] && (2)\\
&\twoheadrightarrow \nu k, h_1, h_2, k_3.\, (k_3 : \mathbf{0}) \mid [M; k] \mid [M_1; k_3] && (3)\\
&\rightsquigarrow \nu k, h_1, h_2.\, (\langle h_1, \tilde{h}\rangle \cdot k : b\langle P\rangle) \mid (\langle h_2, \tilde{h}\rangle \cdot k : b(X) \triangleright \mathbf{0}) \mid [M; k] && (4)\\
&\equiv \nu k.\, k : (b\langle P\rangle \mid b(X) \triangleright \mathbf{0}) \mid [M; k] && (5)\\
&\rightsquigarrow \nu k.\, (k_1 : a\langle P\rangle) \mid (k_2 : a(X) \triangleright b\langle X\rangle \mid b(X) \triangleright \mathbf{0}) && (6)
\end{aligned}$$

with $\tilde{h} = \{h_1, h_2\}$, $M_1 = (\langle h_1, \tilde{h}\rangle \cdot k : b\langle P\rangle) \mid (\langle h_2, \tilde{h}\rangle \cdot k : b(X) \triangleright \mathbf{0})$. We can note in (2) the use of the rule E.TagP from left to right, in order to allow the two primitive processes to execute (3). On the other side, we use the rule in the opposite way in (5) in order to build back the parallel composition and enable the backward reduction in (6).

Remark 5. One could have thought of mimicking the structural congruence rule dealing with parallel composition in [12], using a monoid structure for tags:

$$(\text{E.TagP}^*)\ \kappa : (P \mid Q) \equiv \nu h_1, h_2.\, (h_1 \cdot \kappa : P) \mid (h_2 \cdot \kappa : Q)$$

Unfortunately using E.TagP* instead of E.TagP would introduce some undesired non-determinism, which would later complicate our definitions (in relation to causality) and proofs. For instance, let $M = k : a\langle Q\rangle \mid (h : a(X) \triangleright X)$. We have:

$$M \to M' = \nu l.\, (l : Q) \mid [M; l]$$

Now, assuming ETagP*, we would have:

$$\begin{aligned}&M \equiv (k : (a\langle Q\rangle \mid \mathbf{0})) \mid (h : a(X) \triangleright X) \equiv\\ &\nu h_1, h_2.\, ((h_1 \cdot k : a\langle Q\rangle) \mid (h_2 \cdot k : \mathbf{0})) \mid (h : a(X) \triangleright X)\end{aligned}$$

Let $M_1 = (h_1 \cdot k : a\langle Q\rangle) \mid (h : a(X) \triangleright X)$. We would then have: $M \to M''$, where $M'' = \nu h_1, h_2, l.\, (l : Q) \mid [M_1; l] \mid (h_2 \cdot k : \mathbf{0})$. Clearly $M' \not\equiv M''$, which means that a seemingly deterministic configuration, M, would have in fact two (actually, an infinity of) derivations towards non structurally equivalent configurations. By insisting on tagging only primitive thread processes, E.TagP avoids this unfortunate situation.

We can characterize this by proving a kind of *determinacy* lemma for rhoπ, which fails if we replace rule E.TagP with rule E.TagP*. Extend the grammar of rhoπ with marked primitive thread processes of the form τ^*. This extended calculus has exactly the same structural congruence and reduction rules as rhoπ, but with possibly marked primitive thread processes. Now call *primed* a closed configuration M with exactly two marked processes of the form $a\langle P\rangle^*$ and $(a(X) \triangleright Q)^*$. By extending $\equiv$ and $\to$ to marked processes we have:

Lemma 2 (Determinacy). *Let M be a primed configuration such that $M \equiv M_1 = \mathbb{E}_1[\kappa_1 : a\langle P\rangle^* \mid \kappa_2 : (a(X) \triangleright Q)^*]$ and $M \equiv M_2 = \mathbb{E}_2[\kappa'_1 : a\langle P\rangle^* \mid \kappa'_2 : (a(X) \triangleright Q)^*]$. Assume $M_1 \to M'_1$ and $M_2 \to M'_2$ are derived by applying* R.Fw *with configurations $\kappa_1 : a\langle P\rangle^* \mid \kappa_2 : (a(X) \triangleright Q)^*$, and $\kappa'_1 : a\langle P\rangle^* \mid \kappa'_2 : (a(X) \triangleright Q)^*$, respectively, followed by* R.Ctx. *Then $M'_1 \equiv M'_2$.*

Proof. By induction on the form of $\mathbb{E}_1$, and case analysis on the form of κ_1 and κ_2. □

We can now formally define the notion of *consistent configuration.*

Definition 1 (Consistent configuration).
Let $M \equiv \nu\tilde{u}.\ \prod_{i\in I}(\kappa_i : \rho_i) \mid \prod_{j\in J}[\mu_j; k_j]$, with $\rho_i = \mathbf{0}$ or ρ_i a primitive thread process, $\mu_j = \delta_j : R_j \mid \gamma_j : T_j$, $R_j = a_j\langle P_j\rangle$, $T_j = a_j(X_j) \triangleright Q_j$ be a configuration. Let K be the multiset containing all the tags κ_i for $i \in I$, δ_j, γ_j for $j \in J$. M is said to be *consistent* if the following conditions are met:

1. K is a set (i.e. all the elements in K are distinct)
2. For all $j \in J$, $k_j \neq \delta_j$ and $k_j \neq \gamma_j$
3. For all $j_1, j_2 \in J$, $j_1 \neq j_2 \implies k_{j_1} \neq k_{j_2}$
4. If there exists a tag $\kappa \in K$ of the form $\langle h, \tilde{h}\rangle \cdot k$ then:
   - for each $h_l \in \tilde{h}$ there is $\kappa' \in K$ of the form $\langle h_l, \tilde{h}\rangle \cdot k$
   - $k \notin K$
   - there is no $\kappa' \in K$ of the form $\langle h'_l, \tilde{h}'\rangle \cdot k$ with $\tilde{h} \neq \tilde{h}'$
5. For all $j \in J$, there exist $E \subseteq I$, $D \subseteq J \setminus \{j\}$, $G \subseteq J \setminus \{j\}$, such that:

$$\nu\tilde{u}.\ k_j : Q_j\{^{P_j}/_{X_j}\} \equiv \nu\tilde{u}.\ \prod_{e\in E} \kappa_e : \rho_e \mid \prod_{d\in D} \delta_d : R_d \mid \prod_{g\in G} \gamma_g : T_g$$

Roughly, consistent configurations enjoy two properties: (i) uniqueness of tags and (ii) that for each memory $[\mu; k]$ there are processes corresponding to the continuation of μ in the configuration. In more detail, condition 1 ensures that processes (inside or outside memory) have unique tags. Condition 2 ensures that the thread tag of a memory never occurs in its own configuration part. Condition 3 states that all thread tags of memories are distinct. Condition 4 ensures that if a process (inside or outside memory) has a complex tag $\langle h, \tilde{h}\rangle \cdot k$ then there are processes tagged with all the tags $\langle h_l, \tilde{h}\rangle \cdot k$ with $h_l \in \tilde{h}$, and that no process is tagged by k or by a different complex tag with the same suffix key k. Condition 5 is the most tricky one. It requires that for each memory $[\delta_j : a_j\langle P_j\rangle \mid \gamma_j : a_j(X_j) \triangleright Q_j; k_j]$ there are threads in the configuration whose composition gives the continuation $\nu\tilde{u}.\ k_j : Q_j\{^{P_j}/_{X_j}\}$. Note that because of the use of $\equiv$ there are only two possibilities: either there is a unique thread corresponding to the continuation tagged with k_j, or there are many of them, all tagged with complex tags having k_j as a suffix, generated by one application of rule E.TagP. These threads may be at top level or inside the configuration parts of other memories (as participants to other communications).

Consistency is a global property, so the composition of consistent configurations may lead to a non consistent configuration, and subterms of consistent configurations may not be consistent. To better understand consistency let us consider a few examples:

$$k_1 : a\langle \mathbf{0} \rangle \mid [(k_1 : a\langle \mathbf{0} \rangle) \mid (k_2 : a(X) \triangleright b\langle \mathbf{0} \rangle); k] \tag{1}$$
$$k : a\langle \mathbf{0} \rangle \mid [(k_1 : a\langle \mathbf{0} \rangle) \mid (k_2 : a(X) \triangleright b\langle \mathbf{0} \rangle); k] \tag{2}$$
$$k : a\langle \mathbf{0} \rangle \mid [(k_1 : a\langle \mathbf{0} \rangle) \mid (k_2 : a(X) \triangleright a\langle X \rangle); k] \tag{3}$$
$$(\langle h_1, \tilde{h} \rangle \cdot k : a\langle \mathbf{0} \rangle) \mid (k : b\langle \mathbf{0} \rangle) \tag{4}$$

Configuration (1) is not consistent since it violates condition 1 on key k_1 and condition 5 on the memory. Configuration (2) is not consistent because it violates condition 5. Configuration (3) is consistent since all the tags are unique and $k : a\langle X \rangle\{^{\mathbf{0}}/_{X}\} \equiv k : a\langle \mathbf{0} \rangle$. Configuration (4) is not consistent since it violates condition 4.

Consistent configurations are closed under reduction:

Lemma 3 (Consistency preservation). *Let M be a consistent configuration. If $M \to N$ then N is a consistent configuration.*

Proof. See Appendix A.1. □

Remark 6. The presented semantics and machinery for reversing HOπ can be easily adapted to define reversibility in first order π-calculus. In general, the combination of memories and identifiers should be enough to define reversibility in calculi with implicit substitutions. Indeed, the need for memories stems from the fact that substitution is not a bijective, hence reversible, function: the only way to reverse a substitution is to record additional information to recover the exact form of the process before the substitution was applied.

2.4. Basic properties of reduction in rhoπ

In this section we show two main properties of rhoπ: (i) that rhoπ is a conservative extension of HOπ, and (ii) that each rhoπ reduction step can indeed be reversed.

We first recall HOπ syntax and semantics. The syntax of HOπ processes coincides with the syntax of rhoπ processes in Figure 1 (but HOπ has no concept of configuration). HOπ structural congruence, denoted $\equiv_\pi$, is the least congruence generated by rules in Figure 2 (restricted to processes) but E.TagN and E.TagP. Evaluation contexts in HOπ are given by the following grammar:

$$\mathbb{E} ::= \cdot \quad | \quad (P \mid \mathbb{E}) \quad | \quad \nu u.\, \mathbb{E}$$

HOπ reduction relation $\rightarrow_\pi$ is the least binary relation on closed processes closed under HOπ structural congruence and HOπ evaluation contexts defined by the rule:

$$(\text{HO.Red}) \quad a\langle P\rangle \mid a(X) \triangleright Q \rightarrow_\pi Q\{^P/_X\}$$

In order to show (i) we define the *erasing function* $\gamma : \mathcal{C} \rightarrow \mathcal{P}$, which maps a rho$\pi$ configuration to its underlying HOπ process.

Definition 2 (Erasing function). The *erasing function* $\gamma : \mathcal{C} \rightarrow \mathcal{P}$ is defined inductively by the following clauses:

$$\begin{array}{lll} \gamma(\mathbf{0}) = \mathbf{0} & \gamma(\nu a.\, M) = \nu a.\, \gamma(M) & \gamma(\nu k.\, M) = \gamma(M) \\ \gamma(M \mid N) = \gamma(M) \mid \gamma(N) & \gamma(\kappa : P) = P & \gamma([\mu; k]) = \mathbf{0} \end{array}$$

Let us note that γ directly deletes the creation of new keys (νk) since they have no meaning in HOπ (they are not names). Moreover it deletes all the extra machinery (tags and memories) used to reverse HOπ.

Lemma 4 below shows that rhoπ forward computations are indeed decorations on HOπ reductions.

Lemma 4. *For all closed configurations M, N, if $M \twoheadrightarrow N$ then $\gamma(M) \rightarrow_\pi \gamma(N)$*

Proof. See Appendix A.2. □

We can prove a converse of Lemma 4. More precisely, Lemma 5 shows that for each HOπ process R, each HOπ reduction $R \rightarrow_\pi S$ and each rhoπ configuration M such that $\gamma(M) = R$ we have a forward reduction in rhoπ corresponding to $R \rightarrow_\pi S$.

Lemma 5. *For all closed HOπ processes R, S if $R \rightarrow_\pi S$ then for all closed configurations M such that $\gamma(M) = R$ there is N such that $M \twoheadrightarrow N$ and $\gamma(N) = S$.*

PROOF. See Appendix A.2.

Remark 7. A canonical way of lifting a closed HOπ process P to a closed consistent configuration in $\mathsf{rho}\pi$ is by defining $\delta(P) = \nu k.\, k : P$. As a corollary of Lemma 4 we have:

Corollary 1. *For each closed HOπ process P, if $\delta(P) \twoheadrightarrow N$ then $P \rightarrow_\pi \gamma(N)$.*

We now prove the Loop Lemma, which shows that forward and backward reductions in $\mathsf{rho}\pi$ are really the inverse of each other.

Lemma 6 (Loop Lemma). *For all closed consistent configurations M, N if $M \twoheadrightarrow N$ then $N \rightsquigarrow M$, and if $M \rightsquigarrow N$ then $N \twoheadrightarrow M$.*

PROOF. Let us start from the first implication. From Lemma 36 (see Appendix A.1) we have $M \equiv M'$ and $N' \equiv N$ with $M' = \nu \tilde{u}.\, \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \mid \prod_{i \in I} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j]$ and $N' = \nu \tilde{u}, k.\, k : Q\{^P/_X\} \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] \mid \prod_{i \in I} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j]$. Then by applying rules (R.FW), (R.CTX) and (R.EQV) we have $N \rightsquigarrow M$, as desired.

For the other direction from Lemma 37 (see Appendix A.1) we have $M \equiv M'$ with $M' = \nu \tilde{u}, k.\, k : R \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] \mid \prod_{i \in I} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j]$ and $\nu \tilde{u}.\, \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \mid \prod_{i \in I} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j] \equiv N$. Since M is a consistent configuration and consistency is preserved by structural congruence also M' is consistent. From consistency we know that $k : R$ is the only thread tagged by k and that $\nu \tilde{u}, k.\, k : R \equiv \nu \tilde{u}, k.\, k : Q\{^P/_X\}$. Then the result follows by applying rules (R.BW), (R.CTX) and (R.EQV). $\square$

An easy induction on the length n of the sequence of reductions $M \Rightarrow N$ shows that:

Corollary 2. *For all closed consistent configurations M, N if $M \Rightarrow N$ then $N \Rightarrow M$.*

2.5. Contextual equivalence in $\mathsf{rho}\pi$

Barbed congruence. We can classically complement the operational semantics of the $\mathsf{rho}\pi$ calculus with the definition of a contextual equivalence between configurations, which takes the form of a barbed congruence. We first define observables in configurations. We say that name a is *observable in configuration* M, noted $M \downarrow_a$, if $M \equiv \nu \tilde{u}. (\kappa : a\langle P \rangle) \mid N$, with $a \notin \tilde{u}$. Note that keys are not observable: this is because they are just an internal device used to support reversibility. We write $M\mathcal{R} \downarrow_a$, where $\mathcal{R}$ is a binary relation on configurations, if there exists N such that $M\mathcal{R}N$ and $N \downarrow_a$. The following definitions are classical:

Definition 3 (Barbed bisimulation and congruence). A relation $\mathcal{R} \subseteq \mathcal{C}^{\bullet} \times \mathcal{C}^{\bullet}$ on closed configurations is a *strong* (resp. *weak*) *barbed simulation* if whenever $M \,\mathcal{R}\, N$

- $M \downarrow_a$ implies $N \downarrow_a$ (resp. $N \Rightarrow\downarrow_a$)
- $M \to M'$ implies $N \to N'$, with $M'\mathcal{R}N'$ (resp. $N \Rightarrow N'$ with $M'\mathcal{R}N'$)

A relation $\mathcal{R} \subseteq \mathcal{C}^{\bullet} \times \mathcal{C}^{\bullet}$ is a *strong* (resp. *weak*) *barbed bisimulation* if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are strong (resp. weak) barbed simulations. We call *strong* (resp. *weak*) *barbed bisimilarity* and write $\sim$ (resp. $\approx$) the largest strong (resp. weak) barbed bisimulation. The largest congruence included in $\sim$ (resp. $\approx$) is called *strong* (resp. *weak*) *barbed congruence* and is denoted $\sim_c$ (resp. $\approx_c$).

A direct consequence of the Loop Lemma is that each closed consistent configuration M is weakly barbed congruent to any of its descendants or predecessors.

Lemma 7. *For all closed consistent configurations M, N, if $M \Rightarrow N$, then $M \approx_c N$.*

Proof. We show that the relation

$$\mathcal{R} = \{(\mathbb{C}[M], \mathbb{C}[N]) \mid M \Rightarrow N, \mathbb{C} \text{ is a configuration context}\}$$

is a weak barbed bisimulation. Since $\mathcal{R}$ is symmetric by the corollary of the Loop Lemma (Corollary 2), we only need to show that it is a weak barbed simulation. Consider a pair $(\mathbb{C}[M], \mathbb{C}[N]) \in \mathcal{R}$. We have $M \Rightarrow N$, and hence by Corollary 2 $N \Rightarrow M$. Noting that a configuration context $\mathbb{C}$ is an execution context, i.e. if $M \to N$ then $\mathbb{C}[M] \to \mathbb{C}[N]$, then we also have $\mathbb{C}[N] \Rightarrow \mathbb{C}[M]$. We now check easily the two barbed simulation clauses:

- if $\mathbb{C}[M] \downarrow_a$, then $\mathbb{C}[N] \Rightarrow \mathbb{C}[M] \downarrow_a$, and hence $\mathbb{C}[N] \Rightarrow\downarrow_a$ as required.
- if $\mathbb{C}[M] \to M'$, then $\mathbb{C}[N] \Rightarrow \mathbb{C}[M] \to M'$, and hence $\mathbb{C}[N] \Rightarrow M'$, as required.

Since $\mathcal{R}$ is a congruence by construction the thesis follows. □

Lemma 7 shows that barbed congruence is not very discriminative among configurations: if N is derived from M then $M \approx_c N$. Barbed bisimilarity is even less discriminative: a configuration M is weak barbed bisimilar to a dummy configuration that shows directly all the possible barbs of M.

Lemma 8. *For each closed consistent configuration M let S be the set of weak barbs of M. Then $M \approx M_D = \prod_{a \in S}(k_a : a\langle \mathbf{0} \rangle)$.*

Proof. Let $\mathcal{R} = \{(N, M_D) \mid M \Rightarrow N, M_D = \prod_{a \in S}(k_a : a\langle \mathbf{0} \rangle\}$. We show that $\mathcal{R}$ is a weak barbed bisimilarity. All the challenges from N are matched by M_D by staying idle, since it has by construction all the required barbs. M_D performs no actions to be matched. Let a be a barb of M_D. By construction a is a weak barb of M, and also of N since $N \Rightarrow M$ by the Loop Lemma. □

We will leave finding a general notion of observational equivalence for rhoπ, and for reversible calculi in general, for further study. For our purposes, i.e. proving the correctness of our encoding of rhoπ in a variant of higher-order π, it is enough to consider a custom notion of back and forth bisimulation.

Back and Forth Bisimulations. Notions of back and forth bisimulation, requiring essentially that forward moves are matched by forward moves and backward moves by backward moves, have already been studied in the literature (see [24, 25, 14, 26]). However, those works consider only strong equivalences, and use backward actions to better understand calculi with only forward actions, while we use them to study reversible calculi. Indeed, this kind of back and forth bisimulations can distinguish *true concurrency aspects* of forward calculi better than other equivalences such as classical strong bisimulation. For example, the two (CCS) processes:

$$P = a \mid b \qquad Q = a.b + b.a$$

are strongly bisimilar, but they are not back and forth bisimilar. Suppose that P performs the computation ab and then it *undoes* a. This computation cannot be matched by Q: if Q does the computation ab (left part of the choice) then it cannot undo a before b, since b causally depends on a.

We adapt this idea of back and forth bisimulation by defining an ad-hoc notion of weak behavioral equivalence allowing also reductions which are neither forward nor backward. They are needed since the encoding of $\mathsf{rho}\pi$ needs some auxiliary steps for managing bookkeeping information. We call such reductions *administrative* reductions. The equivalence we present below works up to administrative reductions. While we think that such a notion is useful for reasoning on our encoding, we do not claim that it is a good candidate to become a canonical equivalence on reversible calculi. Indeed, if it is used to relate processes of calculi with no administrative reductions, then this relation becomes a strong back and forth bisimulation.

We write $\hookrightarrow$ for the reduction relation composed by administrative steps, and $\hookrightarrow^*$ for its reflexive and transitive closure. Note that in $\mathsf{rho}\pi$ relation $\hookrightarrow$ is empty.

Definition 4 (Bf barbed bisimulation and congruence).
A relation $\mathcal{R} \subseteq \mathcal{C}^\bullet \times \mathcal{C}^\bullet$ on closed configurations is a *strong* (resp. *weak*) *backward and forward barbed simulation* (or *bf barbed simulation* for brevity) if whenever $M \, \mathcal{R} \, N$

- $M \downarrow_a$ implies $N \downarrow_a$ (resp. $N \hookrightarrow^* \downarrow_a$)
- $M \twoheadrightarrow M'$ implies $N \twoheadrightarrow N'$, with $M' \mathcal{R} N'$ (resp. $N \hookrightarrow^* \twoheadrightarrow \hookrightarrow^* N'$ with $M' \mathcal{R} N'$)
- $M \rightsquigarrow M'$ implies $N \rightsquigarrow N'$, with $M' \mathcal{R} N'$ (resp. $N \hookrightarrow^* \rightsquigarrow \hookrightarrow^* N'$ with $M' \mathcal{R} N'$)
- $M \hookrightarrow M'$ implies $N \hookrightarrow N'$, with $M' \mathcal{R} N'$ (resp. $N \hookrightarrow^* N'$ with $M' \mathcal{R} N'$)

A relation $\mathcal{R} \subseteq \mathcal{C}^\bullet \times \mathcal{C}^\bullet$ is a *strong* (resp. *weak*) *bf barbed bisimulation* if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are strong (resp. weak) bf barbed simulations. We call *strong* (resp. *weak*) *bf barbed bisimilarity*, denoted by $\dot{\sim}$ (resp. $\dot{\approx}$), the largest strong (resp. weak) bf barbed bisimulation. The largest congruence included in $\dot{\sim}$ (resp. $\dot{\approx}$) is called *strong* (resp. *weak*) *bf barbed congruence* and is denoted by $\dot{\sim}_c$ (resp. $\dot{\approx}_c$).

Weak bf barbed bisimulation is actually strong with respect to forward and backward reductions, thus showing that the encoding correctly matches the behavior of $\mathsf{rho}\pi$ processes, but weak with respect to administrative steps $\hookrightarrow$, thus allowing the encoding to perform them at any moment. If bf barbed bisimulation is used to compare $\mathsf{rho}\pi$ processes, the strong version and the weak one coincide, that is $\dot{\approx} = \dot{\sim}$. Hence, in $\mathsf{rho}\pi$, weak bf barbed bisimulation becomes stronger than (classical) strong barbed bisimulation, since it is able to distinguish the direction of reductions. In fact, one forward (backward) step has to be matched by one forward (backward) step.

From the definitions, it is clear that $\dot{\sim} \subseteq \sim$, $\dot{\approx} \subseteq \approx$, $\dot{\sim}_c \subseteq \sim_c$, and $\dot{\approx}_c \subseteq \approx_c$, i.e., backward and forward equivalences are more discriminating than the corresponding standard counterparts. We show below that all these inclusions are strict.

Let us start from strong equivalences. Consider:

$$\begin{aligned}
M &= \nu k_1, k_2.\, (k_1 : a\langle \mathbf{0} \rangle) \mid (k_2 : a(X) \triangleright b\langle \mathbf{0} \rangle) \\
N &= \nu k_1, k_2, k.\, [(k_1 : b\langle \mathbf{0} \rangle) \mid (k_2 : b(X) \triangleright a\langle \mathbf{0} \rangle); k] \mid k : a\langle \mathbf{0} \rangle
\end{aligned}$$

Both the configurations have a barb at a, a reduction to a configuration with a barb at b and its inverse, and no other reduction nor barb, thus $M \sim N$. However, the reduction creating the barb at b is forward in M and backward in N, thus $M \not\dot{\sim} N$. The same counterexample and the same reasoning can be used also for weak equivalences.

Let us consider now congruences, starting from the weak ones. Consider the following configurations:

$$\begin{aligned}
M &= \nu k_1, k_2, k_3.\, (k_1 : a\langle \mathbf{0} \rangle) \mid (k_2 : a(X) \triangleright b\langle \mathbf{0} \rangle) \mid (k_3 : a(X) \triangleright c\langle \mathbf{0} \rangle) \\
N &= \nu k_1, k_2, k_3, k_4.\, (k_4 : b\langle \mathbf{0} \rangle) \mid (k_3 : a(X) \triangleright c\langle \mathbf{0} \rangle) \mid \\
&\qquad [(k_1 : a\langle \mathbf{0} \rangle) \mid (k_2 : a(X) \triangleright b\langle \mathbf{0} \rangle); k_4] \\
M' &= \nu k_1, k_2, k_3, k_5.\, (k_5 : c\langle \mathbf{0} \rangle) \mid (k_2 : a(X) \triangleright b\langle \mathbf{0} \rangle) \mid \\
&\qquad [(k_1 : a\langle \mathbf{0} \rangle) \mid (k_3 : a(X) \triangleright c\langle \mathbf{0} \rangle); k_5]
\end{aligned}$$

Since $M \twoheadrightarrow N$ by Lemma 7 $M \approx_c N$, but $M \not\dot{\approx} N$ and hence $M \not\dot{\approx}_c N$. To prove this last part, assume that there exists a weak bf barbed bisimulation $\mathcal{R}$ between M and N, i.e. such that $(M, N) \in \mathcal{R}$. We have $M \twoheadrightarrow M'$, and we have no N' such that $N \twoheadrightarrow N'$ nor $N \hookrightarrow N'$, thus the reduction cannot be matched.

For the strong congruences, consider:

$$\begin{aligned}
M &= \nu k_1, k_2, a, b.\, (k_1 : a\langle \mathbf{0} \rangle) \mid (k_2 : a(X) \triangleright b\langle \mathbf{0} \rangle) \\
N &= \nu k_1, k_2, k, a, b.\, [(k_1 : a\langle \mathbf{0} \rangle) \mid (k_2 : a(X) \triangleright b\langle \mathbf{0} \rangle); k] \mid k : b\langle \mathbf{0} \rangle
\end{aligned}$$

The two configurations have no barbs, and they cannot interact with their contexts. Thus $M \sim_c N$, since they can both do infinite reductions. However $M \not\dot{\sim}_c N$, since M can perform a forward step while N cannot perform forward nor administrative steps.

3. Causality in rhoπ

We now proceed to the analysis of causality in rhoπ. We first show that reversibility in rhoπ is causally consistent. We then show that the causality information in rhoπ is at least as fine as the causality information from [21].

3.1. Causal consistency

In order to prove causal consistency, we mostly adapt the terminology and arguments of [12].

We call *transition* a triplet of the form $M \xrightarrow{m_{\twoheadrightarrow}} M'$, or $M \xrightarrow{m_{\rightsquigarrow}} M'$, where M, M' are closed consistent configurations, $M \to M'$, and m is the memory involved in the reduction $M \to M'$. We say that a memory m is *involved* in a forward reduction $M \twoheadrightarrow M'$ if $M \equiv \mathbb{E}[\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q]$, $M' \equiv \mathbb{E}[\nu k. (k : Q\{^P/_X\}) \mid m]$, and $m = [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k]$. In this case, the transition involving m is denoted by $M \xrightarrow{m_{\twoheadrightarrow}} M'$. Likewise, we say that a memory $m = [N; k]$ is involved in a backward reduction $M \rightsquigarrow M'$ if $M \equiv \mathbb{E}[(k : Q) \mid m]$, $M' \equiv \mathbb{E}[N]$. In this case, the transition involving m is denoted by $M \xrightarrow{m_{\rightsquigarrow}} M'$. We let η and its decorated variants range over labels $m_{\twoheadrightarrow}$ and $m_{\rightsquigarrow}$. If $\eta = m_{\twoheadrightarrow}$, we set $\eta_\bullet = m_{\rightsquigarrow}$ and vice versa. In a transition $M \xrightarrow{\eta} N$, we say that M is the *source* of the transition, N is its *target*, and η is its label (of the form $m_{\twoheadrightarrow}$ or $m_{\rightsquigarrow}$, where m is some memory).

Definition 5 (Name-preserving transitions). We say a transition $t : M \xrightarrow{\eta} M'$ is *name-preserving* if M and M' are in thread normal form and if one of the following assumptions holds:

1. $M = \nu\tilde{u}. \prod_{i\in I}(\kappa_i : \rho_i) \mid \prod_{j\in J}[\mu_j; k_j]$ and $M' = \nu\tilde{u}. \prod_{i\in I'}(\kappa_i : \rho_i) \mid \prod_{j\in J'}[\mu_j; k_j]$, with $J' = J \cup \{j'\}$, $I' \subset I$ and $\eta = m_{\twoheadrightarrow}$ with $m = [\mu_{j'}; k_{j'}]$;
2. $M = \nu\tilde{u}. \prod_{i\in I}(\kappa_i : \rho_i) \mid \prod_{j\in J}[\mu_j; k_j]$ and $M' = \nu\tilde{u}. \prod_{i\in I'}(\kappa_i : \rho_i) \mid \prod_{j\in J'}[\mu_j; k_j]$, with $J = J' \cup \{j'\}$, $I \subset I'$ and $\eta = m_{\rightsquigarrow}$ with $m = [\mu_{j'}; k_{j'}]$.

Intuitively, a name-preserving transition keeps track of the set of restricted identifiers of its configurations (and especially the tag of memory m). In the rest of this section we only consider name-preserving transitions and "transition" used in a definition, lemma or theorem, stands for "name-preserving transition". Note that working with name-preserving transitions only is licit because of the determinacy lemma (Lemma 5).

Two transitions are said to be *coinitial* if they have the same source, *cofinal* if they have the same target, *composable* if the target of one is the source of the other. A sequence of pairwise composable transitions is called a *trace*. We let t and its decorated variants range over transitions, σ and its decorated variants range over traces. Notions of target, source and composability extend naturally to traces. We note ϵ_M the empty trace with source M, $\sigma_1;\sigma_2$ the composition of two composable traces σ_1 and σ_2. The *stamp* $\lambda(m_{\twoheadrightarrow})$ of a memory $m = [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k]$ is defined to be the set $\{\kappa_1, \kappa_2, k\}$; we set $\lambda(m_{\rightsquigarrow}) = \lambda(m_{\twoheadrightarrow})$.

Definition 6 (Concurrent transitions). Two coinitial transitions $t_1 = M \xrightarrow{\eta_1} M_1$ and $t_2 = M \xrightarrow{\eta_2} M_2$ are said to be *in conflict* if there is a tag $\kappa \in \lambda(\eta_1)$ such that $\kappa \in \lambda(\eta_2)$ or κ is a key k and there is a complex tag with suffix k in $\lambda(\eta_2)$ or there is a tag $\kappa \in \lambda(\eta_2)$ such that $\kappa \in \lambda(\eta_1)$ or κ is a key k and there is a complex tag with suffix k in $\lambda(\eta_1)$. Two coinitial transitions are *concurrent* if they are not in conflict.

Remark 8. Note that the stamp of a memory $[M; k]$ includes its tag k. This is necessary to take into account possible conflicts between a forward action and a backward action, as in the following example. The configuration

$$M = \nu l, k, h.\, (k : a\langle P\rangle) \mid [N; k] \mid (h : a(X) \triangleright Q)$$

has two possible transitions $t = M \xrightarrow{m_{\rightsquigarrow}} \nu l, k, h.\, N \mid (h : a(X) \triangleright Q)$, where $m = [N; k]$, and $t' = M \xrightarrow{m'_{\twoheadrightarrow}} \nu l, k, h.\, [N; k] \mid m' \mid l : Q\{^P/_X\}$, where $m' = [(k : a\langle P\rangle) \mid (h : a(X) \triangleright Q); l]$. The two transitions t and t' are in conflict over the use of the resource $k : a\langle P\rangle$.

Consider now the configuration

$$M = \nu l, k, h, h_1, h_2.\, (\langle h_1, \tilde{h}\rangle \cdot k : a\langle P\rangle) \mid (\langle h_2, \tilde{h}\rangle \cdot k : b\langle \mathbf{0}\rangle) \mid [N; k] \mid (h : a(X) \triangleright Q)$$

where $\tilde{h} = \{h_1, h_2\}$. Again we have a conflict, since the backward action involving memory $[N; k]$ is in conflict with any forward action by descendants of k, even if not all of them are involved.

The Loop Lemma ensures that each transition $t = M \xrightarrow{\eta} N$ has a reverse one $t_\bullet = N \xrightarrow{\eta_\bullet} M$. The above definition of concurrent transitions makes sense:

Lemma 9 (Square lemma). *If $t_1 = M \xrightarrow{\eta_1} M_1$ and $t_2 = M \xrightarrow{\eta_2} M_2$ are two coinitial concurrent transitions, then there exist two cofinal transitions $t_2/t_1 = M_1 \xrightarrow{\eta_2} N$ and $t_1/t_2 = M_2 \xrightarrow{\eta_1} N$.*

Proof. By case analysis on the form of transitions t_1 and t_2. See Appendix B.1 for details. $\square$

We are now in a position to show that reversibility in rhoπ is causally consistent. Following Lévy [27] we define first the notion of causal equivalence between traces that abstracts away from the order of causally independent reductions. We define $\asymp$ as the least equivalence relation between traces closed under composition that obeys the following rules:

$$t_1; t_2/t_1 \asymp t_2; t_1/t_2 \qquad t; t_\bullet \asymp \epsilon_{\mathtt{source}(t)} \qquad t_\bullet; t \asymp \epsilon_{\mathtt{target}(t)}$$

Intuitively $\asymp$ states that if we have two concurrent transitions, then the two traces obtained by swapping the order of their execution are the same, and that a trace composed by a transition followed by its inverse is equivalent to the empty one.

The proof of causal consistency proceeds along the exact same lines as in [12], with simpler arguments because of the simpler form of our memory stamps.

Lemma 10 (Rearranging Lemma). *Let σ be a trace. There exists forward traces σ' and σ'' such that $\sigma \asymp \sigma'_\bullet; \sigma''$.*

Proof. The proof is in Appendix B.1. $\square$

Lemma 11 (Shortening Lemma). *Let σ_1, σ_2 be coinitial and cofinal traces, with σ_2 forward. Then, there exists a forward trace σ_1' of length at most that of σ_1 such that $\sigma_1' \asymp \sigma_1$.*

Proof. The proof is in Appendix B.1. $\square$

Theorem 1 (Causal consistency). *Let σ_1 and σ_2 be coinitial traces, then $\sigma_1 \asymp \sigma_2$ if and only if σ_1 and σ_2 are cofinal.*

Proof. The proof is in Appendix B.1. $\square$

3.2. Causality in rhoπ *and in the causal higher-order π-calculus*

From what precedes, it should be clear that the core of the reversibility machinery in rhoπ is the causality information built during execution. The question therefore arises of the relationship between this notion of causality and the one found in several studies on the causal semantics of the π-calculus, e.g. [21, 28, 29, 30]. In this section we limit ourselves to the study of the relationship between the causality information used in rhoπ and that used by Boreale and Sangiorgi in their analysis of causality for the π-calculus [21].

For a meaningful comparison, we first adapt the causality apparatus from that paper to the asynchronous higher-order π-calculus. We call "causal higher-order π" (choπ for short) the resulting calculus. Processes of choπ are exactly the processes in Figure 1. Following [21], *causal processes* in choπ, ranged over by A, B, are processes decorated with causality information, defined by the following grammar:

$$A ::= K :: A \quad | \quad A \mid A \quad | \quad \nu a.\, A \quad | \quad P$$

where K ranges over finite sets of keys, i.e. $K \subseteq_{\mathrm{fin}} \mathcal{K}$. Informally, a causal process $K :: P$ identifies the set of causes K that gave rise to process P.

We denote by $\mathtt{fk}(A)$ the set of free keys of a causal process A.

We define a reduction semantics for choπ (see Appendix B.2 for a comparison between this reduction semantics and a labelled transition system semantics directly adapted from [21]), via a reduction relation that operates modulo structural congruence. The structural congruence relation $\equiv$ is the least congruence relation on causal processes that obeys the structural congruence rules on HOπ processes given in Figure 2 and the rules in Figure 4. The reduction relation $\rightarrow$ is the least binary relation on closed causal processes that is closed under structural congruence and evaluation contexts defined by the rule:

$$\text{(C-RED)}\ K_1 :: a\langle P\rangle \mid K_2 :: a(X) \triangleright Q \rightarrow K_1 \cup K_2 :: Q\{^P/_X\}$$

Evaluation contexts for causal processes are given by the following grammar:

$$\mathbb{E} ::= \bullet \quad | \quad \mathbb{E} \mid A \quad | \quad A \mid \mathbb{E} \quad | \quad \nu a.\, \mathbb{E}$$

An easy induction on the structure of a causal process gives us a normal form for closed causal processes.

$$(\text{E-PAR})\ K :: A \mid B \equiv K :: A \mid K :: B \qquad (\text{E-CAU})\ K_1 :: K_2 :: A \equiv K_1 \cup K_2 :: A$$

$$(\text{E-RES})\ K :: \nu a.\, A \equiv \nu a.\, K :: A \qquad (\text{E-VOID})\ \emptyset :: A \equiv A \qquad (\text{E-NIL})\ K :: \mathbf{0} \equiv \mathbf{0}$$

Figure 4: Structural congruence for $\mathsf{cho}\pi$

Lemma 12 (Normal form for closed causal processes).
For any closed causal process A we have

$$A \equiv \nu\tilde{c}.\, \prod_{i \in I} K_i :: \tau_i$$

where each τ_i is of the form $a\langle P\rangle$ or the form $a(X) \triangleright P$.

As for $\mathsf{rho}\pi$, we can inductively define a function γ that erases the causality information from a causal process to get a standard HOπ process via the following clauses:

$$\begin{aligned} \gamma(P) &= P & \gamma(\nu a.\, A) &= \nu a.\, \gamma(A) \\ \gamma(A \mid B) &= \gamma(A) \mid \gamma(B) & \gamma(K :: A) &= \gamma(A) \end{aligned}$$

We can now present the relationship between causal information in causal processes and in $\mathsf{rho}\pi$ configurations. This relationship takes the form of a bisimulation which we call *causal correspondence*. Informally, causal correspondence asserts that during execution, a causal process and its “corresponding” $\mathsf{rho}\pi$ process ascribe the same causes to the same sub-processes.

We first define causal barbs on causal processes and $\mathsf{rho}\pi$ configurations.

For causal processes, a causal barb $K :: a$ corresponds to the ability to communicate on a channel a due to causes in K. More precisely:

Definition 7 (Causal barbs for causal processes). A $\mathsf{cho}\pi$ causal process A has a causal barb $K :: \overline{a}$, noted $A \downarrow_{K::\overline{a}}$, if $A \equiv \nu\tilde{c}.\, K :: a\langle P\rangle \mid B$, with $a \notin \tilde{c}$, for some $\tilde{c}, P, B$. A $\mathsf{cho}\pi$ causal process A has a causal barb $K :: a$, noted $A \downarrow_{K::a}$, if $A \downarrow_{K::a}$ implies $A \equiv \nu\tilde{c}.\, K :: a(X) \triangleright P \mid B$, with $a \notin \tilde{c}$, for some $\tilde{c}, P, B$.

For configurations, the notion of causal barb depends on a causality dependence relation between tags.

Definition 8 (Causal dependence). Let M be a configuration and let T_M be the set of tags occurring in M. The binary relation $>_M$ on T_M is defined as the smallest relation satisfying the following clauses:

- $k >_M \langle h_i, \tilde{h} \rangle \cdot k$;
- $\kappa' >_M k$ if κ' occurs in μ for some memory $[\mu; k]$ that occurs in M.

The causal dependence relation $:>_M$ is the reflexive and transitive closure of $>_M$.

$\kappa :>_M \kappa'$ reads "κ is a causal antecedent of κ' according to M". We also write $\kappa <:_M \kappa'$ for $\kappa' :>_M \kappa$. When configuration M is clear from the context, we write $\kappa :> \kappa'$ for $\kappa :>_M \kappa'$, and $\kappa <: \kappa'$ for $\kappa <:_M \kappa'$. When K is a set of keys, we write $\kappa <: K$ if for all $h \in K$, we have $\kappa <: h$. We denote by $\mathtt{fk}(M)$ the set of free keys of a configuration M. Note that $\mathtt{fk}(M) \subseteq \mathtt{fn}(M)$.

Definition 9 (Causal barbs for configurations). A $\mathsf{rho}\pi$ configuration M has a causal barb $K :: \overline{a}$ where $K \subseteq \mathtt{fk}(M)$, noted $M \downarrow_{K::\overline{a}}$, if and only if $M \equiv \nu \tilde{u}. (\kappa : a\langle P \rangle) \mid N$, with $a \notin \tilde{u}$ and $\kappa <: K$. Likewise, $M \downarrow_{K::a}$ where $K \in \mathtt{fk}(M)$, if and only if $M \equiv \nu \tilde{u}. (\kappa : a(X) \triangleright P) \mid N$, with $a \notin \tilde{u}$ and $\kappa <: K$.

Definition 10 (Causal correspondence). A relation $\mathcal{R}$ is a causal correspondence between $\mathsf{cho}\pi$ causal processes and $\mathsf{rho}\pi$ configurations if the following condition holds: if $\langle A, M \rangle \in \mathcal{R}$, then

- $\gamma(A) \equiv \gamma(M)$ and $\mathtt{fk}(A) = \mathtt{fk}(M)$
- if $A \downarrow_{K::\alpha}$, then $M \downarrow_{K::\alpha}$
- if $A \to B$, then there exists N such that $M \to N$ and $\langle B, N \rangle \in \mathcal{R}$
- if $M \downarrow_{K::\alpha}$ and $(M \downarrow_{K'::\alpha} \implies K' \subseteq K)$, then $A \downarrow_{K::\alpha}$
- if $M \to N$, then there exists B such that $A \to B$ and $\langle B, N \rangle \in \mathcal{R}$

A causal process A and a configuration M are said to be in causal correspondence if there exists a causal correspondence $\mathcal{R}$ such that $\langle A, M \rangle \in \mathcal{R}$.

Consider now a $\mathsf{rho}\pi$ configuration M with the following constraints: M has no bound key and no memory, and all tags in M are simple keys, i.e. if $M \equiv \nu\tilde{u}.\ \prod_{i\in I} k_i : \tau_i$, where each τ_i is of the form $a\langle P\rangle$ or $a(X) \triangleright P$ for some a, P, with $k_i \in \mathcal{K}$ and $\tilde{u} \cap \mathcal{K} = \emptyset$. We will call *initial* such a configuration. An initial configuration M can easily be seen as a $\mathsf{cho}\pi$ causal process if we apply to it the syntactic transformation χ that replaces configurations $k : P$ in M, with causal processes $\{k\} :: P$, i.e. that is defined inductively as follows:

$$\chi(\mathbf{0}) = \mathbf{0} \qquad \chi(k : P) = \{k\} :: P$$
$$\chi(\nu a.\, M) = \nu a.\, \chi(M) \qquad \chi(M \mid N) = \chi(M) \mid \chi(N)$$

We can now state our causal correspondence theorem:

Theorem 2. *Let M be an initial configuration. Then $\chi(M)$ and M are in causal correspondence.*

Proof. The proof consists in showing that the relation $\mathcal{R}$, defined below, is a causal correspondence. We first define the predicate $\mathtt{corr}$ that relates $\mathsf{cho}\pi$ closed causal processes with $\mathsf{rho}\pi$ closed configurations: we have $\mathtt{corr}(A, M)$ if the following conditions hold:

- $A \equiv \nu\tilde{c}.\ \prod_{i\in I} K_i :: \tau_i$
- $M \equiv \nu\tilde{c}, \tilde{h}.\ (\prod_{i\in I} \kappa_i : \tau_i) \mid N$
- for each $i \in I$, τ_i is of the form $a_i\langle P_i\rangle$ or $a_i(X) \triangleright P_i$ for some a_i, P_i
- for each i$\in I$, $\kappa_i <:_M K_i$ and for all k, $\kappa_i <:_M k \implies k \in K_i$
- $\gamma(N) \equiv \mathbf{0}$ and $\tilde{h} \subset \mathcal{K}$

The relation $\mathcal{R}$ is defined as:

$$\mathcal{R} = \{\langle U, V\rangle \mid A \rightarrow^n U, M \rightarrow^n V, M \text{ initial}, A = \chi(M), \mathtt{corr}(U, V)\}$$

where $A \rightarrow^n U$ (resp. $M \rightarrow^n N$) denotes the fact that A reduces to U (resp. M reduces to N) in n steps.

If M is initial, we have $\langle \chi(M), M\rangle \in \mathcal{R}$ by construction. We now check the different clauses of causal correspondence.

Let $\langle U, V\rangle \in \mathcal{R}$. Since $\mathtt{corr}(U, V)$ we can write

$$U \equiv \nu\tilde{c}.\ \prod_{i\in I} K_i :: \tau_i \qquad V \equiv \nu\tilde{c}, \tilde{h}.\ \prod_{i\in I} \kappa_i : \tau_i \mid N$$

with $\gamma(N) = \mathbf{0}$ and for all $i \in I$, $\kappa_i <:_M K_i$.

- Since $\gamma(N) = \mathbf{0}$, we have $\gamma(U) \equiv \nu\tilde{c}.\, \prod_{i\in I} \tau_i \equiv \gamma(V)$. Now by induction on the length n of the derivations $A \to^n U$ and $M \to^n V$, it is easy to check that $\mathtt{fk}(U) = \mathtt{fk}(A)$ and $\mathtt{fk}(M) = \mathtt{fk}(V)$. Since $A = \chi(M)$, we have $\mathtt{fk}(A) = \mathtt{fk}(M)$ and thus $\mathtt{fk}(U) = \mathtt{fk}(V)$.

- Assume that $U \downarrow_{K::\alpha}$, then we must have $U \downarrow_{K_i::a_i}$ or $U \downarrow_{K_i::\overline{a_i}}$ for some $i \in I$. Since $\mathtt{corr}(U, V)$, we must then have $V \downarrow_{\kappa_i:a_i}$ or $V \downarrow_{\kappa_i:\overline{a_i}}$ with $\kappa_i <:_V K_i$.

- Assume that $U \to U'$. Then, we must have $j, l \in I$ such that $a_j = a_l$ and $\tau_j = a\langle P\rangle$, $\tau_l = a(X) \triangleright Q$ and
$$U' \equiv \nu\tilde{c}.\, (K_j \cup K_l :: Q\{^P/_X\}) \mid \prod_{i\in I\setminus\{j,l\}} K_i :: \tau_i$$
for some P, Q. But then we have
$$V \to \nu\tilde{c}, \tilde{h}, k.\, k : Q\{^P/_X\} \mid (\prod_{i\in I\setminus\{j,l\}} \kappa_i : \tau_i) \mid [m; k] \mid N = V'$$
where $m = \kappa_j : a\langle P\rangle \mid \kappa_l : a(X) \triangleright Q$. We have $k <:_{V'} \kappa_i$ and $k <:_{V'} \kappa_l$, thus $k <:_{V'} K_j \cup K_l$. We check that $\langle U', V'\rangle \in \mathcal{R}$, which amounts to proving that $\mathtt{corr}(U', V')$. Assume, without loss of generality, that $Q \equiv \nu\tilde{e}.\, \prod_{i\in J} \tau_i$, with $J \cap I = \emptyset$. Then
$$U' \equiv \nu\tilde{c}, \tilde{e}.\, \prod_{i\in(I\cup J)\setminus\{j,l\}} K_i :: \tau_i$$
$$V' \equiv \nu\tilde{c}, \tilde{e}, \tilde{h}, k, \tilde{l}.\, \prod_{i\in(I\cup J)\setminus\{j,l\}} \kappa_i : \tau_i \mid [m; k] \mid N$$
where $\tilde{l} = \{l_i \mid i \in J\}$ and, for all $i \in J$, $K_i = K_j \cup K_l$, $\kappa_i = \langle l_i, \tilde{l}\rangle \cdot k$. We have $\gamma([m; k] \mid N) \equiv \mathbf{0}$, and for all $i \in J$ $\kappa_i <:_{V'} k$, and thus $\kappa_i <:_{V'} K_i$. Therefore U', V' have the required form and thus $\mathtt{corr}(U', V')$.

- Assume that $V \downarrow_{K::\alpha}$, with K maximal, then we must have $V \downarrow_{K_i::a_i}$ or $V \downarrow_{K_i::\overline{a_i}}$ for some $i \in I$ and $K = K_i$. Since $\mathtt{corr}(U, V)$, we must then have $U \downarrow_{K_i:a_i}$ or $V \downarrow_{K_i:\overline{a_i}}$.

- Assume that $V \to V'$. Then, we must have $j, l \in I$ such that $a_j = a_l$ and $\tau_j = a\langle P\rangle$, $\tau_l = a(X) \triangleright Q$,
$$V' \equiv \nu\tilde{c}, \tilde{h}, k.\, k : Q\{^P/_X\} \mid (\prod_{i\in I\setminus\{j,l\}} \kappa_i : \tau_i) \mid [m; k] \mid N$$

where $m = \kappa_j : a\langle P \rangle \mid \kappa_j : a(X) \triangleright Q$. Assume that $Q \equiv \nu \tilde{e}. \prod_{i \in J} \tau_i$, with $J \cap I = \emptyset$. Then

$$V' \equiv \nu \tilde{c}, \tilde{e}, \tilde{h}, k, \tilde{l}. \prod_{i \in (I \cup J) \setminus \{j,l\}} \kappa_i : \tau_i \mid [m; k] \mid N$$

where $\tilde{l} = \{l_i \mid i \in J\}$ and for all $i \in J$, $\kappa_i = \langle l_i, \tilde{l} \rangle \cdot k$. Notice that we have for all $i \in J$ $\kappa_i <:_{V'} k$ and thus $\kappa_i <:_{V'} K_j \cup K_l$. Now, we have

$$U \rightarrow \nu \tilde{c}. (K_j \cup K_l :: Q\{^P/_X\}) \mid \prod_{i \in I \setminus \{j,l\}} K_i :: \tau_i = U'$$

Setting for all $i \in J$ $K_i = K_j \cup K_l$, we obtain

$$U' \equiv \nu \tilde{c}, \tilde{e}. \prod_{i \in (I \cup J) \setminus \{j,l\}} K_i :: \tau_i$$

and thus $\texttt{corr}(U', V')$, as required.

$\square$

4. Encoding rhoπ in HOπ

This section shows that rhoπ can be encoded in a variant of HOπ, which we call HOπ^+, that allows the use of bi-adic channels, join patterns [31, 32], sub-addressing, abstractions and applications. This particular variant was chosen for convenience, because it simplifies our encoding. All HOπ^+ constructs are well understood in terms of expressive power with respect to HOπ (see [33, 34] for abstractions in π-calculus).

The encoding presented here is slightly different from the one presented in [16], in order to obtain a finer correspondence result. Indeed [16] shows that a rhoπ configuration and its translation are weak barbed bisimilar. Such result allows us to encode reversibility in an already existing calculus, without introducing any new ad-hoc primitive. Even if the result is quite surprising, because of the coarseness of weak barbed bisimulation (as emerged in Section 2.5), it does not establish a strong enough correspondence between rhoπ and its encoding. Therefore, here we base our results on a stronger equivalence, weak bf barbed bisimulation, able to distinguish forward reductions from backward ones.

$$
\begin{aligned}
P, Q &::= \mathbf{0} \;\mid\; X \;\mid\; \nu u.\, P \;\mid\; (P \mid Q) \;\mid\; u\langle F, v\rangle \;\mid\; J \triangleright P \;\mid\; (F\ V)\\
F &::= (u)P \;\mid\; (X)P \;\mid\; (u)F \;\mid\; (X)F\\
V &::= u \;\mid\; F\\
J &::= u(X, v) \;\mid\; u(X, \backslash v) \;\mid\; J \mid J\\
u, v &\in \mathcal{I}
\end{aligned}
$$

Figure 5: Syntax of HOπ^+

The remainder of the section is organized as follows: first we introduce the syntax and the semantics of HOπ^+, then we introduce our encoding and show that a rhoπ consistent configuration M and its translation in HOπ^+ are weak bf barbed bisimilar. To ease the reading of the section, some proofs and auxiliary results are reported in Appendix C.

4.1. HOπ^+

The syntax of HOπ^+ is given in Figure 5. Channels in HOπ^+ are bi-abic in the sense that they carry pairs of the form F, v, that is an abstraction and a name; a trigger can receive a message on a given channel, or on a given channel provided that the received message carries some given name (sub-addressing). Actually triggers use Join patterns, i.e. they allow to specify a set of messages that are read atomically. Join patterns are linear. HOπ^+ supports abstractions over names $(u)P$ and over process variables $(X)P$, and applications $(P\ V)$, where a value V can be a name or an abstraction. We take the set of names of HOπ^+ to be the set $\mathcal{I} \cup \{\star\}$ where $\mathcal{I}$ is the set of rhoπ identifiers ($\mathcal{I} = \mathcal{N} \cup \mathcal{K}$). Thus both rho$\pi$ names and rhoπ keys are names in HOπ^+. The set of (process) variables of HOπ^+ is taken to coincide with the set $\mathcal{V}$ of variables of rhoπ. Moreover we let B to range over processes P and functions F.

The structural congruence for HOπ^+, denoted by $\equiv$, obeys the same rules as those of rhoπ processes (see Figure 2), except for the rules E.TagN and E.TagP, which are specific to rhoπ. Evaluation contexts in HOπ^+, as in HOπ, are given by the following grammar:

$$\mathbb{E} ::= \cdot \;\mid\; (P \mid \mathbb{E}) \;\mid\; \nu u.\, \mathbb{E}$$

The reduction relation for HOπ^+, also denoted by $\to$, is defined as the least relation on closed processes closed under HOπ^+ structural congruence and

$$(\textsc{Red})\ \frac{\texttt{match}(F_i, X_i) = \theta'_i \qquad \texttt{match}(v_i, \psi_i) = \theta_i}{\prod_{i=1}^{n} u_i\langle F_i, v_i\rangle \mid (\prod_{i=1}^{n} u_i(X_i, \psi_i) \triangleright P) \to P\theta'_1 \ldots \theta'_n\theta_1 \ldots \theta_n}$$

$$(\textsc{App})\ \frac{\texttt{match}(V, \psi) = \theta}{((\psi)B\ V) \to B\theta}$$

Figure 6: Reduction rules for HOπ^+

HOπ^+ evaluation contexts that satisfies the rules in Figure 6, where ψ is either a name v, a process variable X or an escaped name $\backslash u$. The function `match` in Figure 6 is the partial function which is defined in the cases given by the clauses below, and undefined otherwise:

$$\texttt{match}(u, v) = \{{}^{u}/_{v}\} \quad \texttt{match}(u, \backslash u) = \{{}^{u}/_{u}\} \quad \texttt{match}(F, X) = \{{}^{F}/_{X}\}$$

Note that an escaped name $\backslash u$ will match just with name u. Rule Red is a generalization (in the sense of join patterns) of the usual communication rule for HOπ. If there are enough messages (left-hand side of the reduction in the conclusion) satisfying a certain input process, then they are consumed at once and the continuation of the input process is triggered with the necessary substitutions. Rule App mimics the well known β-reduction of the λ-calculus [35]. We use $\Rightarrow$ to denote the reflexive and transitive closure of $\to$.

Remark 9. Even if HOπ^+ allows using arbitrary join patterns, our encoding will use just binary join patterns.

Conventions. In writing HOπ^+ terms, $u\langle v\rangle$ abbreviates $u\langle (X)\mathbf{0}, v\rangle$, $\overline{u}$ abbreviates $u\langle (X)\mathbf{0}, \star\rangle$ and $u\langle F\rangle$ abbreviates $u\langle F, \star\rangle$. Likewise, $a(u) \triangleright P$ abbreviates $a(X, u) \triangleright P$, where $X \notin \texttt{fv}(P)$, $a \triangleright P$ abbreviates $a(X, \star) \triangleright P$, where $X \notin \texttt{fv}(P)$, and $a(X) \triangleright P$ abbreviates $a(X, \star) \triangleright P$. We adopt the usual conventions for writing applications and abstractions: $(F\ V_1 \ldots V_n)$ stands for $(((F\ V_1) \ldots)\ V_n)$, and $(X_1 \ldots X_n)F$ stands for $(X_1) \ldots (X_n)F$. When there is no potential ambiguity, we often write $F\,V$ for $(F\ V)$. When defining HOπ^+ processes, we freely use recursive definitions for these can be encoded using e.g. the Turing fixed point combinator Θ defined as $\Theta = (\mathtt{A}\ \mathtt{A})$, where $\mathtt{A} = (X\ F)(F\ (X\ X\ F))$ (cf. [35, p. 132]).

In the rest of the paper we denote by $\mathcal{P}_{\mathrm{HO}\pi^+}$ the set of HOπ^+ processes, by $\mathcal{C}_{\mathsf{rho}\pi}$ the set of $\mathsf{rho}\pi$ configurations and by $\mathcal{P}_{\mathsf{rho}\pi}$ the set of $\mathsf{rho}\pi$ processes.

4.2. $\mathsf{rho}\pi$ *encoding*

The encoding $(\!|\cdot|\!) : \mathcal{P}_{\mathsf{rho}\pi} \to \mathcal{P}_{\mathrm{HO}\pi^+}$ of processes of $\mathsf{rho}\pi$ in HOπ^+ is defined inductively in Figure 7. It extends to an encoding $(\!|\cdot|\!) : \mathcal{C}_{\mathsf{rho}\pi} \to \mathcal{P}_{\mathrm{HO}\pi^+}$ of configurations of $\mathsf{rho}\pi$ in HOπ^+ as given in Figure 8 (the encoding for $\mathbf{0}$ in Figure 8 is the encoding for the null configuration). The two main ideas behind the encoding are given now. First, a tagged process $l : P$ is interpreted as a process equipped with a special channel l (sometimes we will refer to it as its *key channel*) on which to report that it has successfully rolled back. This intuition leads to the encoding of a $\mathsf{rho}\pi$ process as an abstraction which takes this reporting channel l (its own key) as a parameter. Second, each $\mathsf{rho}\pi$ process translation generates also the *process killer*, that is a process in charge of rolling it back. We have three kinds of such processes: `KillM`, `KillT` and `KillP` representing respectively the killer of a message, of a trigger and of a parallel composition of processes.

Let us now describe the encoding of configurations given in Figure 8. A null configuration $\mathbf{0}$ is encoded as the null HOπ^+ process $\mathbf{0}$. The parallel and the restriction operator are mapped to the corresponding operators of HOπ^+. There are two ways of encoding a tagged process $\kappa : P$ depending on the kind of the tag. If the tag is a key, then the translation is the application of the encoding of P to the name k, that is $(\!|P|\!)k$. If the tag is complex, of the form $\langle h_i, \tilde{h}\rangle \cdot k$, then the translation is the application of the translation of P to the name h_i, in parallel with the killer of the complex tag. Since we want to generate at once a *tree* of killer processes able to revert an entire parallel composition made of n elements, with n being the size of $\tilde{h}$, we opt for the first element of the sequence to generate it. Said otherwise, a killer of a complex tag $\langle h_i, \tilde{h}\rangle \cdot k$ is the null process if $i \neq 1$, otherwise it is a parallel composition of killer processes able to rollback all the branches in which the key k has been split. Hence, the killer of the complex tag $\langle h_1, \tilde{h}\rangle \cdot k$ is in charge of mimicking the behavior of the $\mathsf{rho}\pi$ structural rule E.TagP (see Figure 2 in Section 2.3) used (from right to left) to build back a tagged parallel composition from a parallel composition of related threads. The encoding of a memory will be explained later.

$$
\begin{array}{ll}
(\!|\mathbf{0}|\!) = \mathtt{Nil} & (\!|X|\!) = X \\
(\!|a\langle P\rangle|\!) = (l)(\mathtt{Msg}\; a\; (\!|P|\!)\; l) & (\!|\nu a.\, P|\!) = (l)\nu a.\, (\!|P|\!)\, l \\
(\!|P \mid Q|\!) = (l)(\mathtt{Par}\; (\!|P|\!)\; (\!|Q|\!)\; l) \text{ if } P, Q \not\equiv \mathbf{0} & (\!|P \mid \mathbf{0}|\!) = (\!|P|\!) \\
(\!|a(X) \triangleright P|\!) = (l)(\mathtt{Trig}\; ((X\; c)c\langle(\!|P|\!)\rangle)\; a\; l) & (\!|\mathbf{0} \mid P|\!) = (\!|P|\!)
\end{array}
$$

$$
\begin{array}{l}
\mathtt{Nil} = (l)(l\langle\mathtt{Nil}\rangle \mid (\mathtt{Rew}\; l)) \\
\mathtt{Msg} = (a\; X\; l)a\langle X, l\rangle \mid (\mathtt{KillM}\; a\; l) \\
\mathtt{KillM} = (a\; l)(a(X, \backslash l) \triangleright l\langle(h)\mathtt{Msg}\, a\, X\, h\rangle \mid \mathtt{Rew}\, l) \\
\mathtt{Par} = (X\; Y\; l)\nu h, k.\, X\, h \mid Y\, k \mid (\mathtt{KillP}\; h\; k\; l) \\
\mathtt{KillP} = (h\; k\; l)(h(W) \mid k(Z) \triangleright l\langle(l)\mathtt{Par}\, W\, Z\, l\rangle \mid \mathtt{Rew}\, l) \\
\mathtt{Trig} = (Y\; a\; l)\nu \mathtt{t}.\, \overline{\mathtt{t}} \mid (a(X, h) \mid \mathtt{t} \triangleright_f \nu k, c.\, (Y\; X\; c) \mid (c(Z) \triangleright (Z\; k)) \mid \\
\qquad\qquad (\mathtt{Mem}\, Y\; a\; X\; h\; k\; l)) \mid (\mathtt{KillT}\; Y\; \mathtt{t}\; l\; a) \\
\mathtt{KillT} = (Y\; \mathtt{t}\; l\; a)(\mathtt{t} \triangleright l\langle(h)\mathtt{Trig}\, Y\, a\, h\rangle \mid \mathtt{Rew}\, l) \\
\mathtt{Mem} = (\; Y a\; X\; h\; k\; l)k(Z) \triangleright_b (\mathtt{Msg}\; a\; X\; h) \mid (\mathtt{Trig}\; Y\; a\; l) \\
\mathtt{Rew} = (l)(l(Z) \triangleright Z\, l)
\end{array}
$$

Figure 7: Encoding rhoπ processes

Before commenting the encoding of rhoπ processes in more detail, let us introduce the Rew process, and the idea behind it. Killer processes allow a process to rollback, but we have to give also the possibility to *undo* a rollback decision. This is due to the fact that at any moment each process should be given both the possibility to go forward and to go backward. Assume this is not the case, and consider a parallel composition of processes. If one branch decides (spontaneously) to rollback (by interacting with its killer process) while the other branches do not, then the rolled-back process would be stuck unless we add the possibility to undo its rollback decision. This is the purpose of a process of the form (Rew l), whose behavior is to read an abstraction carried in a message on the key channel l and then to re-instantiate the abstraction with the same key. Naturally this makes the encoding divergent, but divergence is quite natural in a reversible calculus. In fact, each rhoπ configuration which is not stuck can diverge by continuously doing and

$$
\begin{aligned}
&(\!|\mathbf{0}|\!) = \mathbf{0}\\
&(\!|M \mid N|\!) = (\!|M|\!) \mid (\!|N|\!)\\
&(\!|\nu u.\, M|\!) = \nu u.\, (\!|M|\!)\\
&(\!|k : P|\!) = ((\!|P|\!)\ k)\\
&(\!|\langle h_i, \tilde{h}\rangle \cdot k : P|\!) = ((\!|P|\!)\ h_i) \mid \mathtt{Kill}_{\langle h_i, \tilde{h}\rangle \cdot k}\\
&(\!|[\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k]|\!) = (\mathtt{Mem}\ ((X\ c)c\langle(\!|Q|\!)\rangle)\ a\ (\!|P|\!)\ (\!|\kappa_1|\!)\ k\ (\!|\kappa_2|\!)) \mid\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{Kill}_{\kappa_1} \mid \mathtt{Kill}_{\kappa_2}\\
&(\!|k|\!) = k\\
&(\!|\langle h_i, \tilde{h}\rangle \cdot k|\!) = h_i
\end{aligned}
$$

$$
\begin{aligned}
&\mathtt{Kill}_{\langle h_1, \tilde{h}\rangle \cdot k} = \nu \tilde{l}.\, (\mathtt{KillP}\ h_1\ l_1\ k) \mid \prod_{i=2}^{n-2} (\mathtt{KillP}\ h_i\ l_i\ l_{i-1}) \mid (\mathtt{KillP}\ h_{n-1}\ h_n\ l_{n-2})\\
&\mathtt{Kill}_{\kappa} = \mathbf{0} \quad \text{otherwise}
\end{aligned}
$$

Figure 8: Encoding rhoπ configurations

undoing the same communication. Thus the issue is not particularly relevant.

Let us now comment on the encoding of rhoπ processes in Figure 7. As we already said, all the translations of rhoπ processes are abstractions over a channel, and this channel is the tag of the process (or part of it in the case of a complex tag). The zero process $\mathbf{0}$ is translated as a message on the abstracted channel along with a Rew process. This translation, as well as translations of other processes, is by itself diverging. Consider the encoding of the rhoπ process $l : \mathbf{0}$:

$$l\langle \mathtt{Nil}\rangle \mid (\mathtt{Rew}\ l) \to l\langle \mathtt{Nil}\rangle \mid l(Z) \triangleright (Z\ l) \to (\mathtt{Nil}\ l) \to l\langle \mathtt{Nil}\rangle \mid (\mathtt{Rew}\ l)$$

Divergence here could be avoided by removing the Rew process from the translation of $\mathbf{0}$, and considering it as just a message on its abstracted channel. That is:

$$(\!|\mathbf{0}|\!) = (l)l\langle \mathtt{Nil}\rangle$$

We stick however to the first translation to preserve the symmetry with the translations of the other primitive processes (messages and triggers), since this simplifies the statement of some invariants of our encoding, and since, as already said, divergence here is not a relevant issue.

The translation of a message $k : a\langle P\rangle$ after a few applications becomes a process of the form:

$$a\langle (\!|P|\!), k\rangle \mid (\texttt{KillM}\ a\ k)$$

consisting in a message on channel a carrying a pair (here we can see why we use bi-adic channels) in parallel with its killer process. The message carries the translation of the original message content P along with the abstracted channel. The abstracted channel k is needed to ensure that the message will be rolled back only by its own `KillM`. Indeed, the process $(\texttt{KillM}\ a\ k)$ consumes a message on the channel a only if it carries the name k. This is why the `KillM` process is an abstraction over two channels, and explains the need of sub-addressing.

The translation of a parallel composition is quite straightforward: two new key channels are created and given to the translations of the two sub-processes. A `KillP` process will await on these two channels the rollback of the two sub-processes, in order to notify its rollback by building back the entire parallel process into its key channel. This is why we use binary join patterns (the translation of triggers uses join patterns too).

The translation of a trigger $l : a(X) \triangleright Q$ is a process of the form:

$$\nu\texttt{t}.\,\overline{\texttt{t}} \mid (a(X,h) \mid \texttt{t} \triangleright_f \nu k, c.\,(Y\ X\ c) \mid (c(Z) \triangleright (Z\ k)) \mid (\texttt{Mem}\ Y\ a\ X\ h\ k\ l)) \mid (\texttt{KillT}\ Y\ \texttt{t}\ l\ a)$$

with $Y = ((X\ c)c\langle (\!|Q|\!)\rangle)$. Here, a token $\overline{\texttt{t}}$ is used as a lock. In this way either the trigger itself or its killer can acquire this lock and then execute by letting the other process blocked forever. Since all the messages on channel $a \in \mathcal{N}$ are translated into bi-adic messages, triggers are translated in order to read such messages. The continuation of a translated trigger mimics exactly the $\mathsf{rho}\pi$ forward rule: it creates a new key channel k, it substitutes the process variable X with the message content (which is an abstraction) in the trigger continuation. This substitution is mimicked by the application $(Y\ X\ c)$. For example, assume that after a communication we obtain the following process $k : Q\{^{P}/_{X}\}$ where Q is the body of the trigger. Then this substitution is mimicked in our encoding by the following application:

$$((X\ d)d\langle (\!|Q|\!)\rangle)\ (\!|P|\!)\ c \to c\langle (\!|Q|\!)\{^{(\!|P|\!)}/_{X}\}\rangle$$

The trigger continuation, resulting from the substitution, is then applied to the new key channel as follows:

$$c\langle (\!|Q|\!)\{^{(\!|P|\!)}/_X\}\rangle \mid c(Z) \triangleright Z\ k \rightarrow ((\!|Q|\!)\{^{(\!|P|\!)}/_X\})k$$

obtaining the process corresponding to the translation of $k : Q\{^P/_X\}$. A memory process Mem is also created. The Mem process mimics exactly the backward rule of rhoπ: it awaits the rollback of its continuation (a message on the key channel k that the memory bears) and then it releases again the translations of the original rhoπ message and trigger who gave rise to the communication (and to the memory).

The next sections are devoted to prove that the encoding is faithful, i.e. that it preserves the semantics of the original rhoπ configuration. More precisely, we will prove the following theorem.

Theorem 3 (Faithfulness). *For any closed* rhoπ *process P, $\nu k.\ k : P \dot{\approx} (\!|\nu k.\ k : P|\!)$.*

Before proving the theorem we give a brief outline of our proof strategy.

Proof outline. Since the relation $\dot{\approx}$ (see Definition 4) distinguishes three kinds of reductions, forward, backward and administrative, we first divide all the reductions induced by the encoding into these three kinds. Then we give a notion of *normal form* on HOπ^+ processes, whose intent is to consider processes equivalent up to applications in active contexts. Then we characterize the garbage processes generated by the encoding, because of the machinery added to simulate reversibility, with the function addG, and we define a new congruence $\equiv_{Ex}$ to ensure that translations of structurally congruent rhoπ configurations are structurally congruent. We then study the interplay between reduction in normal form and structural congruence $\equiv_{Ex}$, and between reduction in normal form and administrative steps. We then show that a rhoπ reduction can be matched by the encoding, and that $\equiv_{Ex}$ is itself a weak bf barbed bisimulation. We also prove that the garbage produced by the encoding (characterized by addG) does not induce unwanted behaviors. That is, if P is a process derived from the encoding, then P and (any process in) addG(P) are weakly bf barbed bisimilar. We then show that all the reductions of the encoding are matched by rhoπ and finally compose all the obtained results to prove our faithfulness theorem.

4.3. Auxiliary relations

This section provides four main tools needed for proving the faithfulness of the encoding: (i) the reduction system giving to the translation a backward and forward structure, (ii) a characterization of the garbage added by the machinery in the translation, (iii) a normal form for HOπ^+ processes, and (iv) a congruence on HOπ^+ mimicking the one on $\mathsf{rho}\pi$.

To give to the reduction system a backward and forward structure we partition the reductions of the translation into forward, backward and administrative. The basic idea is that administrative reductions can be used both as forward and as backward. Remember that in HOπ^+ we have two kinds of reductions: applications and communications. All applications are administrative reductions. Communications can be either administrative, forward or backward according to the involved trigger. To distinguish the different triggers, we decorate them with labels. This will not change the operational semantics of the calculus.

A labelled trigger is a trigger of the form $J \rhd_b P$ or $J \rhd_f P$. Triggers like the first one will be referred to as backward while triggers like the second one will be referred to as forward. From now on we will denote with $\mathcal{R}_?$ the reflexive closure of a relation $\mathcal{R}$, and with $\mathcal{R}^*$ its reflexive and transitive closure. We now define the two reduction relations $\twoheadrightarrow$ and $\rightsquigarrow$ on HOπ^+ processes.

Definition 11 (Forward and backward HOπ^+ reductions). Let $\twoheadrightarrow$ be a reduction involving a forward trigger and $\rightsquigarrow$ a reduction involving a backward trigger. Moreover let $\hookrightarrow$ be the reduction relation composed by applications and communications involving non labelled triggers. We define $\Rightarrow_f = \hookrightarrow^* \twoheadrightarrow \hookrightarrow^*$ and $\Rightarrow_b = \hookrightarrow^* \rightsquigarrow \hookrightarrow^*$.

Definition 12 (Administrative communications). Let $\mapsto$ be the reduction relation involving communications due to non labelled triggers.

All the applications in the encoding have the form below.

Definition 13 (Applications in the encoding). Let $\rightharpoondown$ be the least evaluation closed relation including the pairs: $\{\langle (h)P\ v, P\{^v/_h\}\rangle \mid P \in \mathcal{P}\} \cup \{(X)P\ (\!|Q|\!)\ , P\{^{(\!|Q|\!)}/_X\} \mid P, (\!|Q|\!) \in \mathcal{P}\}$.

From the definitions above it is clear that $\mapsto \subseteq \hookrightarrow$.

One cannot simply prove that given a (consistent) configuration M, if $M \twoheadrightarrow M'$ then $(\!|M|\!) \Rightarrow_f (\!|M'|\!)$, and similarly for backward reductions. In fact, this does not hold, since the translated processes produce some garbage (in terms of additional processes) due to the execution, and since structural congruent $\mathtt{rho}\pi$ processes do not always have structural congruent translations. Thus we need some auxiliary machinery.

First, we introduce a notion of well formed process, a HOπ^+ process obtained by letting a process of the form $(\!|R|\!)l$ compute.

Definition 14 (Well formed process). A HOπ^+ process P is well formed if there is a $\mathtt{rho}\pi$ process R such that $(\!|R|\!)l \Rightarrow P$.

Then, we characterize the garbage described above by defining a function $\mathtt{addG}(P)$ allowing to add arbitrary garbage to a HOπ^+ process P.

Definition 15. Let P be a HOπ^+ process such that $P \equiv \nu\tilde{a}.\, P'$. Then, $\mathtt{addG}(P) = \{P_G \mid P_G \equiv \nu\tilde{a}.\, (P' \mid \nu\tilde{b}.\, Q)\}$, where Q is a parallel composition (possibly empty) of processes of one of the forms below, or obtained from them by applications:

$$\begin{array}{ll} \mathtt{Rew}\, l & \mathtt{KillM}\ a\ l \\ \nu c, \mathtt{t}.\, (\mathtt{KillT}\ ((X)c\langle(\!|P|\!)\rangle)\ \mathtt{t}\ l\ a) & \nu \mathtt{t}.\, (a(X,k)|\mathtt{t} \triangleright S) \end{array}$$

The last two closures of the $\mathtt{addG}$ function characterize the garbage processes produced by the consumption of the token $\overline{\mathtt{t}}$ in the translation of the trigger process. Both processes are blocked (but for an application of the first one), since the name $\mathtt{t}$ is restricted and the token $\overline{\mathtt{t}}$ has been consumed. To better understand how the token $\overline{\mathtt{t}}$ is used and how garbage can be created, let us consider the following reductions, where a trigger begins a rollback and then undoes it, producing some garbage:

$$\begin{array}{l} (\!|k : a(X) \triangleright \mathbf{0}|\!) = (\!|a(X) \triangleright \mathbf{0}|\!)k = ((h)\mathtt{Trig}\ Y\ a\ h)\ k \to (\mathtt{Trig}\ Y\ a\ k) \Rightarrow \\ \nu \mathtt{t}.\, \overline{\mathtt{t}} \mid (a(X, \backslash h)|\mathtt{t} \triangleright Q) \mid (\mathtt{t} \triangleright k\langle(h)\mathtt{Trig}\ Y\ a\ h\rangle \mid (\mathtt{Rew}\ k)) \to \\ \nu \mathtt{t}.\, (a(X, \backslash h)|\mathtt{t} \triangleright Q) \mid k\langle(h)\mathtt{Trig}\ Y\ a\ h\rangle \mid (\mathtt{Rew}\ k) \to \\ \nu \mathtt{t}.\, (a(X, \backslash h)|\mathtt{t} \triangleright Q) \mid k\langle(h)\mathtt{Trig}\ Y\ a\ h\rangle \mid (k(Z) \triangleright Z\ k) \to \\ \nu \mathtt{t}.\, (a(X, \backslash h)|\mathtt{t} \triangleright Q) \mid ((h)\mathtt{Trig}\ Y\ a\ h)\ k \in \mathtt{addG}((\!|k : a(X) \triangleright \mathbf{0}|\!)) \end{array}$$

where $Q = \nu l, c.\, (Y\ \ X\ \ c) \mid (c(Z) \triangleright Z\ \ l) \mid (\mathtt{Mem}\ Y\ \ a\ \ X\ \ h\ \ l\ \ k)$ and $Y = (X\ \ c)c\langle(\!|\mathbf{0}|\!)\rangle$. The token allows us to avoid to use primitives such as passivation (see [36, 37]) or mixed choice to kill input processes.

We now define a notion of *normal form* for processes, corresponding to processes where all the enabled applications have been executed. Thus, a process in normal form has no enabled applications.

Definition 16 (Normal form). Let $\mathtt{nf}(.)$ be a function from $\mathcal{P}_{\mathrm{HO}\pi^+}$ to $\mathcal{P}_{\mathrm{HO}\pi^+}$ defined as follows:

$$\begin{array}{ll}
\mathtt{nf}(\nu u.\, P) = \nu u.\, \mathtt{nf}(P) & \mathtt{nf}(P \mid Q) = \mathtt{nf}(P) \mid \mathtt{nf}(Q) \\
\mathtt{nf}(a\langle P\rangle) = a\langle P\rangle & \mathtt{nf}(a(X) \triangleright P) = a(X) \triangleright P \\
\mathtt{nf}((X)P\ Q) = \mathtt{nf}(P\{^Q/_X\}) & \mathtt{nf}((h)P\ l) = \mathtt{nf}(P\{^l/_h\}) \\
\mathtt{nf}(0) = 0 &
\end{array}$$

Since reduction to normal form applies only applications in active context, the reduction to normal form is the identity on triggers and messages.

We extend the congruence $\equiv$ on HOπ^+ processes to match the effect of $\mathsf{rho}\pi$ structural congruence after the translation, in order to show that congruent $\mathsf{rho}\pi$ processes are translated into congruent HOπ^+ processes.

Definition 17. Let $\equiv_{Ax}$ be the smallest congruence on HOπ^+ processes satisfying the rules for structural congruence $\equiv$ plus the axioms below.

$$\text{(Ax.C)}\ \mathtt{KillP}\ l\ h\ k \equiv_{Ax} \mathtt{KillP}\ h\ l\ k$$

$$\text{(Ax.A)} \quad \begin{array}{r} \nu l'.\, (\mathtt{KillP}\ l_1\ l_2\ l') \mid (\mathtt{KillP}\ l'\ l_3\ l) \equiv_{Ax} \\ \nu l'.\, (\mathtt{KillP}\ l_1\ l'\ l) \mid (\mathtt{KillP}\ l_2\ l_3\ l') \end{array}$$

$$\text{(Ax.P)}\ \frac{P, Q \text{ closed}}{l_1\langle(\!|P|\!)\rangle \mid l_2\langle(\!|Q|\!)\rangle \mid \mathtt{KillP}\ l_1\ l_2\ l \equiv_{Ax} l\langle(h)Par\ (\!|P|\!)\ (\!|Q|\!)\ h\rangle \mid \mathtt{Rew}\ l}$$

$$\text{(Ax.Unfold)}\ (\!|P|\!)l \equiv_{Ax} \nu\tilde{u}.\, l\langle(\!|Q|\!)\rangle \mid (\mathtt{Rew}\, l) \text{ with } P \equiv \nu\tilde{u}.\, Q$$

$$\text{(Ax.Adm)}\ \nu c.\, (c\langle(\!|P|\!)\rangle \mid c(Z) \triangleright (Z\ k)) \equiv_{Ax} (\!|P|\!)k$$

Definition 18. Let $\equiv_{Ex}$ be the smallest congruence including for each axiom $L \equiv_{Ax} R$ in $\equiv_{Ax}$ both $L \equiv_{Ex} R$ and $\mathtt{nf}(L) \equiv_{Ex} \mathtt{nf}(R)$.

Axioms Ax.C and Ax.A extend respectively the commutativity and associativity of the parallel composition operator to the translation. Axioms Ax.P and Ax.Adm capture the effect of some auxiliary reductions. Axiom Ax.Unfold captures the property of the translation of rhoπ processes of being able to rollback by sending their own code on their key channel.

We show now a few properties of the relations defined above.

The congruence $\equiv_{Ex}$ captures the effect of rhoπ structural congruence $\equiv$ on normal form of translations.

Lemma 13. *Let M, N be closed consistent configurations. Then $M \equiv N$ implies* $\mathtt{nf}(\llparenthesis M \rrparenthesis) \equiv_{Ex} \mathtt{nf}(\llparenthesis N \rrparenthesis)$*.*

Proof. By induction on the derivation of $M \equiv N$. The proof is in Appendix C.1. □

Structural congruence $\equiv_{Ex}$ is preserved by normal form.

Lemma 14. *If P and Q are well formed and $P \equiv_{Ex} Q$ then* $\mathtt{nf}(P) \equiv_{Ex} \mathtt{nf}(Q)$*.*

Proof. See Appendix C.1. □

The congruence $\equiv_{Ex}$ is not influenced by garbage introduced by function $\mathtt{addG}(\bullet)$.

Lemma 15. *If* $\mathtt{nf}(P) \equiv_{Ex} \mathtt{nf}(P')$ *then for each $Q \in \mathtt{addG}(P)$ there exists $Q' \in \mathtt{addG}(P')$ such that* $\mathtt{nf}(Q) \equiv_{Ex} \mathtt{nf}(Q')$*.*

Proof. See Appendix C.1. □

We now prove two invariants on the form of well formed HOπ^+ processes, useful to study properties of the function $\mathtt{addG}(\bullet)$.

If a translation contains a message on a key channel l, then the process contains also the term $\mathtt{Rew}\ l$.

Lemma 16 (Rew Invariant). *If P' is well formed and $P' \equiv \mathbb{C}[l\langle R\rangle]$ with $l \in \mathcal{K}$ for some n-ary context $\mathbb{C}$ then $P' \equiv \mathbb{C}'[l\langle R\rangle \mid S]$ with $S = \mathtt{Rew}\ l$ or $S = l(Z) \triangleright Z\ l$ for some n-ary context $\mathbb{C}'$.*

Proof. See Appendix C.1.

If a process contains a message on a channel $a \in \mathcal{N}$, then the process contains also a corresponding `KillM` process.

Lemma 17 (`KillM` Invariant). *If P' is well formed and $P' \equiv \mathbb{C}[a\langle (\!|P|\!), l\rangle]$ with $a \in \mathcal{N}$ for some n-ary context $\mathbb{C}$ then $P' \equiv \mathbb{C}'[a\langle (\!|P|\!), l\rangle \mid S]$ with $S = (\texttt{KillM}\, a\ l)$ or $S = (a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h\rangle \mid \texttt{Rew}\ l)$ for some n-ary context $\mathbb{C}'$.*

Proof. See Appendix C.1.

Function $\texttt{addG}(\bullet)$ does not add new behaviors.

Lemma 18. *Let P be a well formed HOπ^+ process and $Q \in \texttt{addG}(P)$. If $\texttt{nf}(Q) \hookrightarrow Q'$ then there exists P' such that $P \hookrightarrow^* P'$ with $Q' \in \texttt{addG}(P')$.*

Proof. See Appendix C.1. □

Lemma 19. *Let P be a well formed HOπ^+ process and $Q \in \texttt{addG}(P)$. If $\texttt{nf}(Q) \twoheadrightarrow Q'$ then there exists P' such that $P \hookrightarrow^* \twoheadrightarrow P'$ with $Q' \in \texttt{addG}(P')$.*

Proof. It is easy to see from the definition of function `addG` that the added processes cannot enable $\twoheadrightarrow$ reductions. Hence the reduction $\twoheadrightarrow$ can be done by $\texttt{nf}(P)$, and it is sufficient to chose P' such that $P \hookrightarrow^* \texttt{nf}(P) \twoheadrightarrow P'$. □

Lemma 20. *Let P be a well formed HOπ^+ process and $Q \in \texttt{addG}(P)$. If $\texttt{nf}(Q) \rightsquigarrow Q'$ then there exists P' such that $P \hookrightarrow^* \rightsquigarrow P'$ with $Q' \in \texttt{addG}(P')$.*

Proof. Similar to the one of Lemma 19 □

4.4. Operational correspondance

This section proves a few results on the behavior of the translation, leading to the operational correspondance result at the end of the section.

We start by proving a few basic properties of the translation.

The encoding is well-behaved w.r.t. substitutions:

Lemma 21 (Substitution). *For each $\textsf{rho}\pi$ process P, Q: $(\!|P|\!)\{^{(\!|Q|\!)}/_X\} = (\!|P\{^Q/_X\}|\!)$.*

Proof. By induction on the structure of P. □

Names corresponding to keys in $\mathcal{K}$ are always bound.

Lemma 22. *If* $(\!|\nu k.\, k : P|\!) \Rightarrow P'$ *then* $\mathtt{fn}(P') \cap \mathcal{K} = \emptyset$.

Proof. By induction on the number of steps in $\Rightarrow$. $\square$

We now prove that, essentially, a translation of a $\mathtt{rho}\pi$ process P can always rollback, and the result of the rollback is a message on the process key channel. The rollback is not *perfect*, in the sense that the content of the message is not exactly equal to the translation of the original process P. This is due to the fact that, once a name has been created, there is no way to reverse its creation. However we can prove that the content of the message, wrapped by restrictions on extruded names, is structural congruent to the original process P, plus some garbage. Formally we have:

Lemma 23. *For each closed* $\mathtt{rho}\pi$ *process* P, $(\!|P|\!)k \hookrightarrow^* \nu\tilde{u}.\, k\langle(\!|Q|\!)\rangle \mid \mathtt{Rew}\ k \mid S$ *with* $k \notin \tilde{u}$, $S = \prod R_i$, $R_i = \mathtt{Rew}\ k_i$ *or* $R_i = \nu \mathtt{t}.\, (a(X,h)|\mathtt{t}\triangleright R)$ *and* $P \equiv \nu\tilde{u}.\, Q$.

Proof. See Appendix C.2. $\square$

The encoding never generates two messages on the same key channel, never generates two `KillP` processes waiting for the same rollbacks or never generates a `KillP` and a `Mem` waiting for the same rollback signal.

Lemma 24. *For any* $\mathtt{rho}\pi$ *process* R, *if* $(\!|R|\!)l \Rightarrow P$ *then the following conditions hold:*

1. $P \not\equiv \mathbb{C}[l_1\langle P_1\rangle \mid l_1\langle P_2\rangle]$, *with* $l_1 \in \mathcal{K}$.
2. $P \not\equiv \mathbb{C}[(\mathtt{KillP}\ l_1\ l_2\ l_3) \mid (\mathtt{KillP}\ l_4\ l_5\ l_6)]$, *with* $l_1, l_2, l_3, l_4, l_5, l_6 \in \mathcal{K}$ *and* $\{l_1, l_2\} \cap \{l_4, l_5\} \neq \emptyset$ *or* $l_3 = l_6$.
3. $P \not\equiv \mathbb{C}[(\mathtt{KillP}\ l_1\ l_2\ l) \mid (\mathtt{Mem}\ P\ a\ Q\ h\ l_3\ k)]$, *with* $l, l_1, l_2, l_3, h, k \in \mathcal{K}$ *and* $l_1 = l_3$ *or* $l_2 = l_3$.

Proof. See Appendix C.2.

Processes congruent according to $\equiv_{Ex}$ have the same weak reductions (up to $\equiv_{Ex}$).

Lemma 25. *If* $\mathtt{nf}(P_1) \equiv_{Ex} \mathtt{nf}(P_2)$ *and* $\mathtt{nf}(P_1) \to P_1'$ *then* $\mathtt{nf}(P_2) \Rightarrow \mathtt{nf}(P_2')$ *with* $\mathtt{nf}(P_1'') \equiv_{Ex} \mathtt{nf}(P_2')$ *and* $P_1'' \in \mathtt{addG}(P_1')$. *Furthermore, if* $\to$ *is forward then* $\Rightarrow$ *is* $\Rightarrow_f$, *if* $\to$ *is backward then* $\Rightarrow$ *is* $\Rightarrow_b$, *if* $\to$ *is administrative then* $\Rightarrow$ *is* $\hookrightarrow^*$.

Proof. By case analysis on the used axiom $P \equiv_{Ex} Q$ and on the structure of $\mathtt{nf}(P_1)$. The proof is in Appendix C.2. $\square$

Two applications can always be swapped.

Lemma 26. *For each HOπ^+ process P, if $P \rightharpoondown P_1$ and $P \rightharpoondown P_2$ then there is a HOπ^+ process P_3 such that $P_2 \rightharpoondown P_3$ and $P_1 \rightharpoondown P_3$.*

More in general, applications can be swapped with arbitrary reductions, but they can disappear becuase of the swap.

Reductions and applications commute.

Lemma 27. *If $P \Rightarrow P'$ and $P \rightharpoondown^* P''$ then $P' \rightharpoondown^* Q$ and $P'' \Rightarrow Q$.*

Proof. By induction on the length of $\Rightarrow$ and $\rightharpoondown^*$, showing that if $P \rightarrow_? P'$ and $P \rightharpoondown_? P''$ then $P'' \rightarrow_? Q$ and $P' \rightharpoondown_? Q$. $\square$

We now prove a form of Loop Lemma for auxiliary reductions. Hence, if P is a translation and $P \hookrightarrow^* Q$, Q can somehow go back to P. Reversibility of this computation is however not perfect, but it holds up to garbage and structural congruence $\equiv_{Ex}$.

Lemma 28. *For any consistent configuration M, if $(\!|M|\!) \Rightarrow P$ and $P \hookrightarrow^* Q$ then there exist Q' and P' such that $Q \hookrightarrow^* Q'$, $P' \in \mathtt{addG}(P)$ with $\mathtt{nf}(P') \equiv_{Ex} \mathtt{nf}(Q')$.*

Proof. See Appendix C.2. $\square$

The next theorem proves a form of behavioral correctness for our encoding, showing that the encoding of a process can mimic the process reductions.

Theorem 4. *For each consistent* rhoπ *configuration M, if $M \twoheadrightarrow N$ then $\mathtt{nf}((\!|M|\!)) \Rightarrow_f P$ and if $M \rightsquigarrow N$ then $\mathtt{nf}((\!|M|\!)) \Rightarrow_b P$, and there exists $P' \in \mathtt{addG}((\!|N|\!))$ such that $\mathtt{nf}(P) \equiv_{Ex} \mathtt{nf}(P')$.*

Proof. By induction on the derivation of $M \rightarrow N$, with a case analysis on the last rule applied.

R.Fw: we have that

$$M = \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \twoheadrightarrow$$
$$\nu k.\,(Q\{^P/_X\} \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q\ ;\ k]) = N$$

Moreover we have that $(\!|M|\!) = (\!|\kappa_1 : a\langle P\rangle|\!) \mid (\!|\kappa_2 : a(X) \triangleright Q|\!)$. We distinguish four cases, depending on whether κ_1, κ_2 are complex or not. Let us consider the case $\kappa_1 = k_1$ and $\kappa_2 = k_2$. Assume $Y = ((X\ c)c\langle(\!|Q|\!)\rangle)$. Then:

$$\begin{aligned}
&\mathtt{nf}((\!|M|\!)) = \mathtt{nf}((l)(\mathtt{Msg}\ a\ (\!|P|\!)\ l)k_1) \mid \mathtt{nf}((l)(\mathtt{Trig}\ Y\ a\ l)k_2) = \\
&\nu\mathtt{t}.\, a\langle(\!|P|\!), k_1\rangle \mid \mathtt{nf}(\mathtt{KillM}\ a\ k_1) \mid \overline{\mathtt{t}} \mid \\
&\quad (\mathtt{t}|a(X,h) \triangleright_f \nu k, c.\,(Y\ X\ c) \mid c(Z) \triangleright Z\ k \mid (\mathtt{Mem}\ Y\ a\ X\ h\ k\ k_2)) \mid \\
&\quad \mathtt{nf}(\mathtt{KillT}\ Y\ \mathtt{t}\ k_1\ a) \twoheadrightarrow \\
&\nu c, k, \mathtt{t}.\,\mathtt{nf}(\mathtt{KillM}\ a\ k_1) \mid (Y\ (\!|P|\!)\ c) \mid (c(Z) \triangleright Z\ k) \mid (\mathtt{Mem}\ Y\ a\ X\ k_1\ k\ k_2) \mid \\
&\qquad \mathtt{nf}(\mathtt{KillT}\ Y\ \mathtt{t}\ k_2\ a) \rightharpoondown \\
&\nu c, k, \mathtt{t}.\,\mathtt{nf}(\mathtt{KillM}\ a\ k_1) \mid c\langle(\!|Q|\!)\{^{(\!|P|\!)}/_X\}\rangle \mid (c(Z) \triangleright Z\ k) \mid \\
&\qquad (\mathtt{Mem}\ Y\ a\ X\ k_1\ k\ k_2) \mid \mathtt{nf}(\mathtt{KillT}\ Y\ \mathtt{t}\ k_2\ a) \hookrightarrow \\
&\nu c, k, \mathtt{t}.\,\mathtt{nf}(\mathtt{KillM}\ a\ k_1) \mid ((\!|Q|\!)\{^{(\!|P|\!)}/_X\}\ k) \mid (\mathtt{Mem}\ Y\ a\ X\ k_1\ k\ k_2) \mid \\
&\qquad \mathtt{nf}(\mathtt{KillT}\ Y\ \mathtt{t}\ k_2\ a) = R
\end{aligned}$$

By using Lemma 21 we have that $(\!|Q|\!)\{^{(\!|P|\!)}/_X\} = (\!|Q\{^P/_X\}|\!)$, thus:

$$\begin{aligned}
R = \nu c, k, \mathtt{t}.\,&\mathtt{nf}(\mathtt{KillM}\ a\ k_1) \mid ((\!|Q\{^P/_X\}|\!)\ k) \mid \\
&(\mathtt{Mem}\ Y\ a\ X\ k_1\ k\ k_2) \mid \mathtt{nf}(\mathtt{KillT}\ Y\ \mathtt{t}\ k_2\ a)
\end{aligned}$$

We can conclude by noting that:

$$\mathtt{nf}(R) = \mathtt{nf}(\mathtt{KillM}\ a\ k_1) \mid \mathtt{nf}((\!|N|\!)) \mid \nu c, \mathtt{t}.\,\mathtt{nf}(\mathtt{KillT}\ Y\ \mathtt{t}\ k_2\ a)$$

Since $P' = (\mathtt{KillM}\ a\ k_1) \mid (\!|N|\!) \mid \nu c, \mathtt{t}.\,(\mathtt{KillT}\ Y\ \mathtt{t}\ k_2\ a) \in \mathtt{addG}((\!|N|\!))$ and $\mathtt{nf}(R) \equiv_{Ex} \mathtt{nf}(P')$ the thesis follows.

Let us consider the case in which $\kappa_1 = \langle h_i, \tilde{h}\rangle \cdot k$. We have that:

$$(\!|M|\!) = (l)(\mathtt{Msg}\ a\ (\!|P|\!)\ l)h_i \mid \mathtt{Kill}_{\langle h_i, \tilde{h}\rangle\cdot k} \mid (l)(\mathtt{Trig}\ Y\ a\ l)k_2$$

Using the same reductions as above we have that:

$$\begin{aligned}
&\mathtt{nf}(\llparenthesis M\rrparenthesis) \Rightarrow_f\\
&\nu c, k, \mathtt{t}.\, \mathtt{nf}(\mathtt{KillM}\ a\ k_1) \mid \mathtt{nf}(\mathtt{Kill}_{\langle h_i, \tilde{h}\rangle \cdot k}) \mid \llparenthesis Q\rrparenthesis\{^{\llparenthesis P\rrparenthesis}/_X\}\ k \mid\\
&\qquad (\mathtt{Mem}\ Y\ a\ X\ h_i\ k\ k_2) \mid \mathtt{nf}(\mathtt{KillT}\ Y\ \mathtt{t}\ k_2\ a) = R
\end{aligned}$$

and by using Lemma 21 we have that

$$\begin{aligned}
R = \nu c, k, \mathtt{t}.\, &\mathtt{nf}(\mathtt{KillM}\ a\ h_i) \mid \mathtt{nf}(\mathtt{Kill}_{\langle h_i, \tilde{h}\rangle \cdot k}) \mid (\llparenthesis Q\{^{\llparenthesis P\rrparenthesis}/_X\}\rrparenthesis\ k) \mid\\
&(\mathtt{Mem}\ Y\ a\ X\ h_i\ k\ k_2) \mid \mathtt{nf}(\mathtt{KillT}\ Y\ \mathtt{t}\ k_2\ a)
\end{aligned}$$

We can conclude by noting that $P' = (\mathtt{KillM}\ a\ h_i) \mid \mathtt{Kill}_{\langle h_i, \tilde{h}\rangle \cdot k} \mid (\llparenthesis Q\{^{\llparenthesis P\rrparenthesis}/_X\}\rrparenthesis\ k) \mid (\mathtt{Mem}\ Y\ a\ X\ h_i\ k\ k_2) \mid \nu c, \mathtt{t}.\, (\mathtt{KillT}\ Y\ \mathtt{t}\ k_2\ a) \in \mathtt{addG}(\llparenthesis N\rrparenthesis)$ and $\mathtt{nf}(R) \equiv_{Ex} \mathtt{nf}(P')$.

The two other cases are similar.

R.Bw: we have that $M = k : R \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q\ ;\ k] \rightsquigarrow \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q = N$. Assume that $Y = ((X\ c)c\langle\llparenthesis Q\rrparenthesis\rangle)$. Then, by definition of $\mathtt{nf}(\bullet)$ we have:

$$\begin{aligned}
&\mathtt{nf}(\llparenthesis M\rrparenthesis) =\\
&\mathtt{nf}(\llparenthesis R\rrparenthesis k) \mid \mathtt{nf}(\mathtt{Mem}\ Y\ a\ \llparenthesis P\rrparenthesis\ \llparenthesis \kappa_1\rrparenthesis\ k\ \llparenthesis \kappa_2\rrparenthesis) \mid \mathtt{nf}(\mathtt{Kill}_{\kappa_1}) \mid \mathtt{nf}(\mathtt{Kill}_{\kappa_2}) =\\
&\mathtt{nf}(\llparenthesis R\rrparenthesis k) \mid (k(Z) \triangleright (\mathtt{Msg}\ a\ \llparenthesis P\rrparenthesis\ \llparenthesis \kappa_1\rrparenthesis) \mid (\mathtt{Trig}\ Y\ a\ \llparenthesis \kappa_2\rrparenthesis)) \mid \mathtt{Kill}_{\kappa_1} \mid \mathtt{Kill}_{\kappa_2}
\end{aligned}$$

From Lemma 23 we know that $\llparenthesis R\rrparenthesis k \hookrightarrow^* \nu\tilde{u}.\, k\langle\llparenthesis R'\rrparenthesis\rangle \mid \mathtt{nf}(S) \mid \mathtt{Rew}\ k$ with S a parallel composition of garbage processes and $R \equiv \nu\tilde{u}.\, R'$. Thanks to Lemma 27 we have $\mathtt{nf}(\llparenthesis R\rrparenthesis k) \hookrightarrow^* \nu\tilde{u}.\, k\langle\llparenthesis R'\rrparenthesis\rangle \mid \mathtt{nf}(S) \mid \mathtt{nf}(\mathtt{Rew}\ k)$. Hence:

$$\begin{aligned}
&\mathtt{nf}(\llparenthesis R\rrparenthesis k) \mid (k(Z) \triangleright (\mathtt{Msg}\ a\ \llparenthesis P\rrparenthesis\ \llparenthesis \kappa_1\rrparenthesis) \mid (\mathtt{Trig}\ Y\ a\ \llparenthesis \kappa_2\rrparenthesis)) \mid\\
&\mathtt{nf}(\mathtt{Kill}_{\kappa_1}) \mid \mathtt{nf}(\mathtt{Kill}_{\kappa_2}) \hookrightarrow^*\\
&\nu\tilde{u}.\, k\langle\llparenthesis R'\rrparenthesis\rangle \mid \mathtt{nf}(S) \mid \mathtt{nf}(\mathtt{Rew}\ k) \mid (k(Z) \triangleright_b (\mathtt{Msg}\ a\ \llparenthesis P\rrparenthesis\ \llparenthesis \kappa_1\rrparenthesis) \mid\\
&\qquad (\mathtt{Trig}\ Y\ a\ \llparenthesis \kappa_2\rrparenthesis)) \mid \mathtt{nf}(\mathtt{Kill}_{\kappa_1}) \mid \mathtt{nf}(\mathtt{Kill}_{\kappa_2}) \rightsquigarrow\\
&\nu\tilde{u}.\, (\mathtt{Msg}\ a\ \llparenthesis P\rrparenthesis\ \llparenthesis \kappa_1\rrparenthesis) \mid (\mathtt{Trig}\ Y\ a\ \llparenthesis \kappa_2\rrparenthesis) \mid \mathtt{nf}(\mathtt{Kill}_{\kappa_1}) \mid \mathtt{nf}(\mathtt{Kill}_{\kappa_2}) \mid\\
&\qquad \mathtt{nf}(S) \mid \mathtt{nf}(\mathtt{Rew}\ k) = R
\end{aligned}$$

We have that $P' = \nu\tilde{u}.\, (\mathtt{Msg}\ a\ \llparenthesis P\rrparenthesis\ \llparenthesis \kappa_1\rrparenthesis) \mid (\mathtt{Trig}\ Y\ a\ \llparenthesis \kappa_2\rrparenthesis) \mid \mathtt{Kill}_{\kappa_1} \mid \mathtt{Kill}_{\kappa_2} \mid S \mid \mathtt{Rew}\ k \in \mathtt{addG}(\llparenthesis \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q\rrparenthesis)$ and $\mathtt{nf}(R) \equiv_{Ex} \mathtt{nf}(P')$ as desired.

$$\begin{tikzcd}
M & \equiv & M' & & & & \\
& 13 & & & & & \\
\mathtt{nf}(\llparenthesis M\rrparenthesis) \arrow[dddd, Rightarrow, "f"'] & \equiv_{Ex} & \mathtt{nf}(\llparenthesis M'\rrparenthesis) \arrow[dd, Rightarrow, "f"] & & N' & \equiv & N \\
& 25 & & & & 13 & \\
& & P \arrow[dd, "*"] & ind & \mathtt{nf}(\llparenthesis N'\rrparenthesis) & \equiv_{Ex} & \mathtt{nf}(\llparenthesis N\rrparenthesis) \\
& & & & \mathtt{addG} & 15 & \mathtt{addG} \\
Q \arrow[dd, "*"] & & \mathtt{nf}(P) & \equiv_{Ex} & \mathtt{nf}(R') & \equiv_{Ex} & \mathtt{nf}(R) \\
& & \mathtt{addG} & 15 & \mathtt{addG} & 15 & \mathtt{addG} \\
\mathtt{nf}(Q) & \equiv_{Ex} & \mathtt{nf}(P') & \equiv_{Ex} & \mathtt{nf}(R''') & \equiv_{Ex} & \mathtt{nf}(R'')
\end{tikzcd}$$

Figure 9: The encoding respects $\equiv_{Ex}$ (numbers refer to Lemmas).

Equiv: we have two cases, one for forward reductions and one for backward reductions. We consider the first one, the second being analogous. We have that $M \twoheadrightarrow N$ with hypothesis $M \equiv M'$, $M' \twoheadrightarrow N'$ and $N' \equiv N$. Figure 9 (numbers refers to the used lemmas, *ind* means that inductive hypothesis is applied and $\mathtt{addG}$ that garbage is added) shows the proof schema we use. By inductive hypothesis we have that $\mathtt{nf}(\llparenthesis M'\rrparenthesis) \Rightarrow_f P$ with $\mathtt{nf}(P) \equiv_{Ex} \mathtt{nf}(R')$ and $R' \in \mathtt{addG}(\llparenthesis N'\rrparenthesis)$. By Lemma 13, we have that if $M \equiv M'$ then $\mathtt{nf}(\llparenthesis M\rrparenthesis) \equiv_{Ex} \mathtt{nf}(\llparenthesis M'\rrparenthesis)$, and by Lemma 25 we have that if $\mathtt{nf}(\llparenthesis M'\rrparenthesis) \Rightarrow_f \mathtt{nf}(P)$ then $\mathtt{nf}(\llparenthesis M\rrparenthesis) \Rightarrow_f \mathtt{nf}(Q')$ with $Q' \in \mathtt{addG}(Q)$ and with $\mathtt{nf}(Q) \equiv_{Ex} \mathtt{nf}(P)$. By inductive hypothesis we have that $\mathtt{nf}(P) \equiv_{Ex} \mathtt{nf}(R')$, but since by hypothesis we had $N' \equiv N$ by Lemma 13 we have that $\mathtt{nf}(\llparenthesis N'\rrparenthesis) \equiv_{Ex} \mathtt{nf}(\llparenthesis N\rrparenthesis)$ and by Lemma 15 we have that there exists $R \in \mathtt{addG}(\llparenthesis N\rrparenthesis)$ such that $\mathtt{nf}(R') \equiv_{Ex} \mathtt{nf}(R)$. Thanks to Lemma 15 there exists $P' \in \mathtt{addG}(P)$ with $\mathtt{nf}(Q') \equiv_{Ex} \mathtt{nf}(P')$, and $R''' \in \mathtt{addG}(R') = \mathtt{addG}(\llparenthesis N'\rrparenthesis)$ and $R'' \in \mathtt{addG}(R) = \mathtt{addG}(\llparenthesis N\rrparenthesis)$ such that $\mathtt{nf}(P') \equiv_{Ex} \mathtt{nf}(R''') \equiv_{Ex} \mathtt{nf}(R'')$. We can conclude by saying that $\mathtt{nf}(\llparenthesis M\rrparenthesis) \Rightarrow_f Q'$ and that $\mathtt{nf}(Q') \equiv_{Ex} \mathtt{nf}(R'')$ with $R'' \in \mathtt{addG}(\llparenthesis N\rrparenthesis)$, as desired.

Ctx: by simply induction on the structure of the context, noting that the translation of active contexts is isomorphic. □

4.5. Observations

In this section we study the properties of the encoding from an observational point of view.

Barbs are preserved by administrative steps.

Lemma 29. *If $M \downarrow_a$ and $(\!|M|\!) \hookrightarrow^* Q$ then $Q \hookrightarrow^* \downarrow_a$.*

Proof. See Appendix C.3. □

Weak barbs of encoded processes are originated by barbs of the original process.

Lemma 30. *If $(\!|M|\!) \hookrightarrow^* \downarrow_a$ then $M \downarrow_a$.*

Proof. See Appendix C.3. □

The function `addG` does not remove barbs, that is:

Lemma 31. *If $P \hookrightarrow^* \downarrow_a$ then $\texttt{addG}(P) \hookrightarrow^* \downarrow_a$.*

Proof. Since $P \hookrightarrow^* P' \downarrow_a$, we can express P as $\mathbb{E}[0]$ and P' as $\mathbb{E}'[0]$ with $\mathbb{E}[0] \hookrightarrow^* \mathbb{E}'[0] \downarrow_a$. Let $\texttt{addG}(P) = \mathbb{E}[R]$, we still have that $\mathbb{E}[R] \hookrightarrow^* \mathbb{E}'[R] \downarrow_a$, as desired. □

Axioms in $\equiv$ are correct with respect to weak bf bisimulation.

Lemma 32. *The relation $\mathcal{R} = \{(P, Q) \mid P \equiv Q\}$ where P, Q are HOπ^+ processes is a weak bf barbed bisimulation.*

Proof. By induction on the length of the derivation of $P \equiv Q$, with a case analysis on the last applied axiom. All the cases are easy. □

The same holds for axioms in $\equiv_{Ex}$.

Proposition 1. *The relation $\mathcal{R} = \{(P, Q) \mid P \equiv_{Ex} Q\}$ where P, Q are HOπ^+ processes is a weak bf barbed bisimulation.*

Proof. See Appendix C.3. □

4.6. Final proof

Before proving the theorem we show two results ensuring completeness of forward and backward transitions, respectively.

The first one relies on the result below.

Lemma 33. *If* $\mathtt{nf}(\llparenthesis M \rrparenthesis) \twoheadrightarrow P$ *then* $M \twoheadrightarrow M'$ *with* $P \hookrightarrow^* P'$ *and* $\mathtt{nf}(P') \equiv \mathtt{nf}(Q')$ *and* $Q' \in \mathtt{addG}(\llparenthesis M' \rrparenthesis)$*.*

Proof. See Appendix C.4. □

Lemma 34. *Let* M *be a* $\mathsf{rho}\pi$ *configuration. If* $\llparenthesis M \rrparenthesis \hookrightarrow^* Q \twoheadrightarrow Q'$ *then there are* M'*,* Q'' *and* Q''' *such that* $M \twoheadrightarrow M'$*,* $Q' \hookrightarrow^* Q''$ *and* $\mathtt{nf}(Q'') \equiv_{Ex} \mathtt{nf}(Q''')$ *and* $Q''' \in \mathtt{addG}(\llparenthesis M' \rrparenthesis)$*.*

Proof. Assume $\llparenthesis M \rrparenthesis \hookrightarrow^* Q \twoheadrightarrow Q'$. By definition of normal form using Lemma 27 we have that $\llparenthesis M \rrparenthesis \hookrightarrow^* Q$ implies that $\mathtt{nf}(\llparenthesis M \rrparenthesis) \hookrightarrow^* Q_1$ with $Q \rightharpoondown^* Q_1$. Moreover we have that $Q_1 \rightharpoondown^* \mathtt{nf}(Q_1)$ and that $\mathtt{nf}(\llparenthesis M \rrparenthesis) \hookrightarrow^* \mathtt{nf}(Q_1)$. Since $Q \twoheadrightarrow Q'$ and $Q \rightharpoondown^* \mathtt{nf}(Q_1)$ by Lemma 27 we have that also $\mathtt{nf}(Q_1) \twoheadrightarrow Q_2$ for some Q_2 such that $Q' \rightharpoondown^* Q_2$. In order to apply Lemma 33 we have to show that also $\mathtt{nf}(\llparenthesis M \rrparenthesis) \twoheadrightarrow P$ for some P. We want to show that we can re-arrange the trace $\mathtt{nf}(\llparenthesis M \rrparenthesis) \hookrightarrow^* \mathtt{nf}(Q_1) \twoheadrightarrow Q_2$ in order to obtain a trace of the form $\mathtt{nf}(\llparenthesis M \rrparenthesis) \twoheadrightarrow\hookrightarrow^* P_2$ with $Q_2 \in \mathtt{addG}(P_2)$. Using Lemma 26 we can write $\mathtt{nf}(\llparenthesis M \rrparenthesis) \hookrightarrow^* \mathtt{nf}(Q_1)$ as $\mathtt{nf}(\llparenthesis M \rrparenthesis) \mapsto\rightharpoondown^* \mathtt{nf}(R_1) \mapsto\rightharpoondown^* \ldots \mapsto\rightharpoondown^* \mathtt{nf}(R_n) \mapsto\rightharpoondown^* \mathtt{nf}(Q_1)$.

We now proceed by induction on the length of this trace, showing that we can either remove reductions or move them after the forward reduction $\mathtt{nf}(Q_1) \twoheadrightarrow Q_2$. The base case (no auxiliary reductions) is trivial. For the inductive case we have a case analysis on the last reduction $\mapsto$.

Internal reduction in a trigger: this case never happens. In fact, by inspecting the encoding, one can see that such a reduction is enabled only after a $\twoheadrightarrow$ reduction takes place.

Reduction due to a `Rew` process: a `Rew` process consumes a message on a key channel, and this kind of message is not present in $\mathtt{nf}(\llparenthesis M \rrparenthesis)$. Thus, back in the trace there exists a $\mapsto$ reduction due to a killer process producing it. Let us select the nearest such $\mapsto$. Since this is the nearest one, there are no intermediate reductions that consume the same message. Hence we can move the killer reduction forward in order to be adjacent to the Rew one, and we can then eliminate both the reductions obtaining a shorter trace. Since garbage may be produced by this pair of reductions, the process obtained by eliminating the two reductions may contain less garbage.

Reduction due to a killer process: the $\mapsto$ reduction can be moved after the $\twoheadrightarrow$ reduction since it does not remove processes that contribute to the $\twoheadrightarrow$ reduction, and we obtain again a shorter trace.

Since $\mathtt{nf}(\llparenthesis M\rrparenthesis) \hookrightarrow^* \mathtt{nf}(Q_1) \twoheadrightarrow Q_2$ implies $\mathtt{nf}(\llparenthesis M\rrparenthesis) \twoheadrightarrow P \hookrightarrow^* P_2$ for some P we can apply Lemma 33 and we have that $M \twoheadrightarrow M'$ and $P \hookrightarrow^* P'$ with $\mathtt{nf}(P') \equiv_{Ex} \mathtt{nf}(P''')$ and $P''' \in \mathtt{addG}(\llparenthesis M'\rrparenthesis)$. By using Lemma 28 there exists Q_3 such that $P_2 \hookrightarrow^* Q_3$ with $\mathtt{nf}(P) \equiv_{Ex} \mathtt{nf}(P_3)$ and $P_3 \in \mathtt{addG}(Q_3)$. Since $P \hookrightarrow^* P'$ we have that $\mathtt{nf}(P) \hookrightarrow^* \mathtt{nf}(P')$ and from Proposition 1 we have that $\mathtt{nf}(P_3) \hookrightarrow^* Q_4 \equiv_{Ex} \mathtt{nf}(P')$. By definition of addG and by transitivity we have that $Q' \hookrightarrow^* Q''$ with $Q'' \in \mathtt{addG}(Q_4)$ and $Q_4 \equiv_{Ex} \mathtt{nf}(P''')$ and $P''' \in \mathtt{addG}(\llparenthesis M'\rrparenthesis)$. Then $Q'' \equiv_{Ex} \mathtt{nf}(Q''')$ for a $Q''' \in \mathtt{addG}(\llparenthesis M'\rrparenthesis)$ by adding further garbage. This concludes the proof. $\square$

Lemma 35. *Let M be a* rhoπ *configuration. If $\llparenthesis M\rrparenthesis \hookrightarrow^* Q \rightsquigarrow Q'$ then there exists M', Q'' and Q''' such that $M \rightsquigarrow M'$, $Q' \hookrightarrow^* Q''$ and $\mathtt{nf}(Q'') \equiv_{Ex} \mathtt{nf}(Q''')$ and $Q''' \in \mathtt{addG}(\llparenthesis M'\rrparenthesis)$.*

Proof. We have that $Q \rightsquigarrow Q'$ and by applying Lemma 27 we also have that $\mathtt{nf}(Q) \rightsquigarrow Q_1$ with $Q' \rightharpoondown^* Q_1$. This implies that $\mathtt{nf}(Q) \equiv \nu\tilde{u}.\, R \mid \mathtt{nf}(\llparenthesis \mathtt{Mem}\ Y\ a\ \llparenthesis P\rrparenthesis\ k_1\ k\ k_2\rrparenthesis) \mid k\langle\llparenthesis C\rrparenthesis\rangle$ with $Y = (X\ c)c\langle\llparenthesis P_1\rrparenthesis\rangle$ and that $Q' \equiv \nu\tilde{u}.\, R \mid (\mathtt{Msg}\ a\ \llparenthesis P\rrparenthesis\ k_1) \mid (\mathtt{Trig}\ Y\ a\ k_2)$. Since administrative reductions $\hookrightarrow$ do not remove memories, the memory needs to be already present both in the configuration M and in its normal form. Since $\llparenthesis M\rrparenthesis \hookrightarrow^* Q$ and $\llparenthesis M\rrparenthesis \rightharpoondown^* \mathtt{nf}(\llparenthesis M\rrparenthesis)$ by Lemma 27 we also have that $\mathtt{nf}(\llparenthesis M\rrparenthesis) \hookrightarrow^* R$ with $Q \rightharpoondown^* R$, hence $\llparenthesis M\rrparenthesis \hookrightarrow^* R$, moreover since $Q \rightsquigarrow Q'$ we also have that $R \rightsquigarrow R'$ with $Q' \rightharpoondown^* R'$. By definition of $\llparenthesis \_\rrparenthesis$ and $\mathtt{nf}(\_)$, the process $\mathtt{nf}(\llparenthesis M\rrparenthesis)$ cannot contain a message on a key channel such as $k\langle\llparenthesis C\rrparenthesis\rangle$. Hence, such a message has been generated by the reductions $\mathtt{nf}(\llparenthesis M\rrparenthesis) \hookrightarrow^* R$. We distinguish two cases: either all the communications in $\hookrightarrow^*$ contribute to create such a message, or not. In the first case all the communications are due to killer processes, and we have that $\mathtt{nf}(\llparenthesis M\rrparenthesis) \equiv \nu\tilde{u}.\, \mathtt{nf}(\llparenthesis N\rrparenthesis) \mid \mathtt{nf}(\mathtt{Mem}\ Y\ a\ \llparenthesis P\rrparenthesis\ k_1\ k\ k_2) \mid \mathtt{nf}(\llparenthesis C\rrparenthesis k)$ for some N, hence $M \equiv \nu\tilde{u}.\, N \mid [k_1 : a\langle P\rangle \mid k_2 : a(X) \triangleright P_1; k] \mid k : C$. Since the administrative reductions just create the message on the channel

k, by using Lemma 23 (where S is garbage) we have that $\mathtt{nf}(\llparenthesis M \rrparenthesis) \hookrightarrow^* \nu\tilde{u}.\, \mathtt{nf}(\llparenthesis N \rrparenthesis) \mid (k(Z) \triangleright (\mathtt{Msg}\ a\ \llparenthesis P \rrparenthesis\ k_1) \mid (\mathtt{Trig}\ Y\ a\ k_2)) \mid k\langle \llparenthesis C \rrparenthesis \rangle \mid S \rightsquigarrow R'$ with $R' = \nu\tilde{u}.\, \mathtt{nf}(\llparenthesis N \rrparenthesis) \mid (\mathtt{Msg}\ a\ \llparenthesis P \rrparenthesis\ k_1) \mid (\mathtt{Trig}\ Y\ a\ k_2)) \mid S$. On the other side we have that $M \rightsquigarrow \nu\tilde{u}.\, N \mid k_1 : a\langle P \rangle \mid k_2 : a(X) \triangleright P_1 = M'$ and we can conclude since $\mathtt{nf}(R') = \mathtt{nf}(Q''')$ with $Q''' \in \mathtt{addG}(\llparenthesis M' \rrparenthesis)$.

If there are administrative reductions that do not contribute to the creation of the message on k, we re-arrange the trace $\mathtt{nf}(\llparenthesis M \rrparenthesis) \hookrightarrow^* R \rightsquigarrow R'$ so to have first all the reductions that contribute to create the message on k, then the $\rightsquigarrow$ reduction and finally all the unrelated reductions. We proceed by induction on the length of the reduction $\mathtt{nf}(\llparenthesis M \rrparenthesis) \hookrightarrow^* R$, with a case analysis on the last reduction that does not contribute to the creation of the message on k. It can be either a communication due to a killer process unrelated to the process with tag k, or a communication due to a $\mathtt{Rew}\ l$. In the first case the kill does not concerns the process labelled by k nor a parallel composition due to the split of the key k. Thus, it can be moved after the $\rightsquigarrow$ reduction and we can conclude by induction on a shorter trace.

If the reduction is due to a $\mathtt{Rew}$ process we have two cases: either it deals with processes related to the one tagged by k, or not. In the second case we proceed as in the case above and we conclude by induction on a shorter trace. In the first case, note that a reduction due to a $\mathtt{Rew}$ process instantiates a process. Since this reduction is related to the process on channel k, the instantiated process must be re-killed by possibly many successive kills. Hence, we can remove the $\mathtt{Rew}$ reduction and the corresponding kills, and we can conclude by induction on a shorter trace.

At the end, we have a trace of the form $\mathtt{nf}(\llparenthesis M \rrparenthesis) \hookrightarrow^* R_1 \rightsquigarrow R'_1 \hookrightarrow^* R'$ with the first trace $\hookrightarrow^*$ containing all the administrative reductions related to the creation of the message on k. As in the first case we know that $\mathtt{nf}(\llparenthesis M \rrparenthesis) \hookrightarrow^* R_1 \rightsquigarrow R'_1$ implies that $M \rightsquigarrow M'$ with $\mathtt{nf}(R'_1) = \mathtt{nf}(R'_2)$ and $R'_2 \in \mathtt{addG}(\llparenthesis M' \rrparenthesis)$. Moreover we have that $R'_1 \hookrightarrow^* R'$ and that $\mathtt{nf}(R'_1) \hookrightarrow^* R''$ with $R' \rightharpoondown^* R''$. By using Lemma 28 we have that there exist Q'' such that $R'' \hookrightarrow^* Q''$ and $\mathtt{nf}(Q'') \equiv_{Ex} \mathtt{nf}(Q_3)$ with $Q_3 \in \mathtt{addG}(R'_1)$. Since $\mathtt{nf}(R'_1) = \mathtt{nf}(R'_2)$ and applications do not change garbage we have that there is Q''' such that $\mathtt{nf}(Q_3) = \mathtt{nf}(Q''')$ and $Q''' \in \mathtt{addG}(\llparenthesis M' \rrparenthesis)$, as desired. $\square$

We can now prove our main result.

Proof of Theorem 3. We prove that the following relation is a weak backward and forward barbed bisimulation:

$$\begin{aligned}\mathcal{R} = \{(M, R) \mid& \\ &\nu k.\ k : P \Rightarrow M \wedge \mathtt{nf}(R) \equiv_{Ex} \mathtt{nf}(Q') \wedge Q' \in \mathtt{addG}(Q) \wedge (\!|M|\!) \hookrightarrow^* Q\}\end{aligned}$$

We have to check the different conditions for weak backward and forward barbed bisimulation.

Assume $M \downarrow_a$. Note that from the definition of barbs only names in $\mathcal{N}$ produce barbs. From Lemma 29 $Q \hookrightarrow^* \downarrow_a$. Since $\mathtt{addG}$ never removes barbs then, thanks to Lemma 31, $Q' \hookrightarrow^* \downarrow_a$. Then also $\mathtt{nf}(Q') \hookrightarrow^* \downarrow_a$. Thanks to Proposition 1 we have that $\mathtt{nf}(R) \hookrightarrow^* \downarrow_a$ and thus also $R \hookrightarrow^* \downarrow_a$.

Assume now $R \downarrow_a$. Thanks to Lemma 22 we have that $a \in \mathcal{N}$. We also have that $\mathtt{nf}(R) \downarrow_a$. Thanks to Proposition 1 $\mathtt{nf}(Q') \hookrightarrow^* \downarrow_a$ and also $Q' \hookrightarrow^* \downarrow_a$. Then $Q \hookrightarrow^* \downarrow_a$. Finally, thanks to Lemma 30 $M \downarrow_a$.

Let us consider reductions. If $M \twoheadrightarrow M'$ then by Theorem 4 we have $\mathtt{nf}((\!|M|\!)) \Rightarrow_f P$ with $\mathtt{nf}(P) \equiv_{Ex} \mathtt{nf}(P')$ and $P' \in \mathtt{addG}((\!|M'|\!))$. By hypothesis we have that $(\!|M|\!) \hookrightarrow^* Q$, but also $\mathtt{nf}((\!|M|\!)) \hookrightarrow^* \mathtt{nf}(Q)$ and by Lemma 28 we have that there are Q_1, Q_2 such that $\mathtt{nf}(Q) \hookrightarrow^* \mathtt{nf}(Q_1)$ and $\mathtt{nf}(Q_1) \equiv_{Ex} \mathtt{nf}(Q_2)$ with $Q_2 \in \mathtt{addG}(\mathtt{nf}((\!|M|\!)))$. Since $\mathtt{nf}((\!|M|\!)) \Rightarrow_f P$ then $Q_2 \Rightarrow_f P_1$ with $P_1 \in \mathtt{addG}(P)$. Since $\mathtt{nf}(Q_2) \equiv_{Ex} \mathtt{nf}(Q_1)$ by Lemma 25 we also have that $\mathtt{nf}(Q_1) \Rightarrow_f P_2$ and $\mathtt{nf}(P_2) \equiv_{Ex} \mathtt{nf}(P_1)$. Since $\mathtt{nf}(P) \equiv_{Ex} \mathtt{nf}(P')$ by transitivity we have that there is $P_3 \in \mathtt{addG}((\!|M'|\!))$ such that $\mathtt{nf}(P_2) \equiv_{Ex} \mathtt{nf}(P_3)$. We can conclude by noting that the pair $(M', P_2) \in \mathcal{R}$ (no administrative reductions are performed from $(\!|M'|\!)$). The backward case is similar.

For the other direction, assume $R \rightarrow R'$. We have a case analysis according to the kind of reduction.

If $R \rightharpoondown R'$ then the thesis follows trivially since $\mathtt{nf}(R) = \mathtt{nf}(R')$.

If instead $R \mapsto R'$ then by Lemma 27 $\mathtt{nf}(R) \hookrightarrow^* R''$ and $R' \rightharpoondown^* R''$ for some R''. Thanks to Lemma 26 $\mathtt{nf}(R'') = \mathtt{nf}(R')$. Thus we have $\mathtt{nf}(R) \hookrightarrow^* \mathtt{nf}(R')$. Thanks to Proposition 1 from $\mathtt{nf}(R) \equiv_{Ex} \mathtt{nf}(Q')$ we have that there is R_1 such that $\mathtt{nf}(Q') \hookrightarrow^* R_1$ and $\mathtt{nf}(R') \equiv_{Ex} R_1$. Thanks to Lemma 18 from $\mathtt{nf}(Q') \hookrightarrow^* R_1$ we have that $Q \hookrightarrow^* R_2$ with $R_1 \in \mathtt{addG}(R_2)$. By using Lemma 14 from $\mathtt{nf}(R') \equiv_{Ex} R_1$ we obtain $\mathtt{nf}(R') \equiv_{Ex} \mathtt{nf}(R_1)$. We can conclude since $\mathtt{nf}(R') \equiv_{Ex} \mathtt{nf}(R_1)$ with $R_1 \in \mathtt{addG}(R_2)$ and $(\!|M|\!) \hookrightarrow^* Q \hookrightarrow^* R_2$, thus $(M, R') \in \mathcal{R}$.

Assume now $R \twoheadrightarrow R'$. By Lemma 27 $\mathtt{nf}(R) \twoheadrightarrow R''$ and $R' \rightharpoondown^* R''$ for some R''. Thanks to Lemma 26 $\mathtt{nf}(R'') = \mathtt{nf}(R')$. Thanks to Proposition 1 we have that $\equiv_{Ex}$ is a weak bf barbed bisimulation, and from $\mathtt{nf}(R) \equiv_{Ex} \mathtt{nf}(Q')$

we have that there is R_1 such that $\texttt{nf}(Q') \hookrightarrow^* \twoheadrightarrow \hookrightarrow^* R_1$ and $R'' \equiv_{Ex} R_1$. From Lemma 14 also $\texttt{nf}(R'') = \texttt{nf}(R') \equiv_{Ex} \texttt{nf}(R_1)$. From Lemma 18 and Lemma 19 we have that there are R_2, Q_1 and Q_1' such that $Q \hookrightarrow^* Q_1 \twoheadrightarrow Q_1' \hookrightarrow^* R_2$ and $\texttt{nf}(R_1) = \texttt{nf}(R''')$ with $R''' \in \texttt{addG}(R_2)$. By hypothesis, we have that $(\!|M|\!) \hookrightarrow^* Q$. By Lemma 34 we have that $(\!|M|\!) \hookrightarrow^* Q_1 \twoheadrightarrow Q_1'$ implies that there exist M', Q_2 and Q_3 such that $M \twoheadrightarrow M'$ and $Q_1' \hookrightarrow^* Q_2$ with $\texttt{nf}(Q_2) \equiv_{Ex} \texttt{nf}(Q_3)$ and $Q_3 \in \texttt{addG}((\!|M'|\!))$.

We can apply Lemma 28 obtaining Q_4 and Q_5 such that $Q_2 \hookrightarrow^* Q_4$ and $\texttt{nf}(Q_4) \equiv_{Ex} \texttt{nf}(Q_5)$ with $Q_5 \in \texttt{addG}(Q_1')$. Since $Q_1' \hookrightarrow^* R_2$ we have that $\texttt{nf}(Q_5) \hookrightarrow^* \texttt{nf}(Q_6)$ with $Q_6 \in \texttt{addG}(R_2)$. Note that R''' and Q_6 differ only because of garbage. Since garbage has no impact on the semantics we can consider them equal (this can be formalized more precisely as an up-to technique). Thus $\texttt{nf}(Q_6) = \texttt{nf}(R_1)$, and since $\texttt{nf}(R') \equiv_{Ex} \texttt{nf}(R_1)$ we also have $\texttt{nf}(R') \equiv_{Ex} \texttt{nf}(Q_6)$. We want to show that the pair $(M', R') \in \mathcal{R}$. We have that $Q_2 \hookrightarrow^* Q_4$ but also $\texttt{nf}(Q_2) \hookrightarrow^* \texttt{nf}(Q_4)$ and since $\texttt{nf}(Q_3) \equiv_{Ex} \texttt{nf}(Q_2)$ and $\equiv_{Ex}$ is a weak bf barbed bisimulation there exists R_3 such that $\texttt{nf}(Q_3) \hookrightarrow^* R_3$ with $R_3 \equiv_{Ex} \texttt{nf}(Q_4)$. Hence by using Lemma 18 we also have that $\texttt{nf}((\!|M'|\!)) \hookrightarrow^* R_4$ with $R_3 \in \texttt{addG}(R_4)$. We have $\texttt{nf}(Q_4) \equiv_{Ex} \texttt{nf}(Q_5)$ and $\texttt{nf}(Q_5) \hookrightarrow^* \texttt{nf}(Q_6)$. Since $\equiv_{Ex}$ is a weak bf barbed bisimulation there exists R_5 such that $\texttt{nf}(Q_4) \hookrightarrow^* R_5$ with $R_5 \equiv_{Ex} \texttt{nf}(Q_6)$. Using the same reasoning we have that $R_3 \hookrightarrow^* R_6$ with $R_6 \equiv_{Ex} R_5$. Since $R_3 \in \texttt{addG}(R_4)$ we have that $R_4 \hookrightarrow^* R_7$ with $R_6 \in \texttt{addG}(R_7)$. Using Lemma 15 and Lemma 14 we have that $\texttt{nf}(R_5) \equiv_{Ex} \texttt{nf}(Q_6)$ and $\texttt{nf}(R_6) \equiv_{Ex} \texttt{nf}(R_5)$. To conclude we can note that $(\!|M'|\!) \rightharpoondown^* \texttt{nf}((\!|M'|\!)) \hookrightarrow^* R_4 \hookrightarrow^* R_7$ with $R_6 \in \texttt{addG}(R_7)$ and $\texttt{nf}(R_6) \equiv_{Ex} \texttt{nf}(Q_6) \equiv_{Ex} \texttt{nf}(R')$. This implies that $(M', R') \in \mathcal{R}$, as desired.

If $R \rightsquigarrow R'$ we can use the same reasoning of the $\twoheadrightarrow$ case by using Lemma 35 and Lemma 20 instead of Lemma 34 and Lemma 19. $\square$

5. Related work

Research into reversible computing has already a long history, that originates in the 1960s. Bennett provides an account [1] of early research on the subject. A full review of works on reversible computing, and the closely related subjects of program inversion (see, e.g., [38] and the references therein) and bidirectional transformation and languages (see, e.g., [39, 40] and the references therein), is out of the scope of this paper but we can highlight works related to our three main contributions: (i) reversible languages and models, (ii) causal semantics and back and forth bisimulation, (iii) translating between reversible and irreversible computations.

Reversible languages and models. The notion of reversible Turing machine seems to date back at least to Lecerf in the early 60s [41], who provides an early encoding of irreversible Turing machines into reversible ones, rediscovered by Bennett in [42]. For a recent survey of reversible Turing machines, their relation to reversible boolean logic, and various reversible models of computation, see [43].

Several works have tackled the problem of adding reversibility to sequential programming languages or to sequential abstract machines. Early work focused on reversible execution [44] and adding undo capabilities to programming languages. Leeman [45] provides an early survey as well as a general framework for adding an *undo* capability to a sequential programming language. Computational history is saved by means of undo-lists, storing previous states of the execution. Primitives dealing with undo-lists are formalized, and different undo operators can be defined by composing them. A way to map compositionally high-level functional programs into a certain kind of reversible automata is given by Abramsky in [46]. In [47], Danos and Regnier give a compositional translation of the λ-calculus into a form of reversible abstract machine called Interaction Abstract Machine (IAM). Other reversible abstract machines for sequential programming such as the SEMCD machine [48] and the Reversible Virtual Machine (RVM) [49] have been proposed.

Whereas the latter virtual machines keep an explicit track of execution history to reconstruct backward computation, several works study natively reversible sequential programming languages where reversibility is obtained without the need to keep additional information to reconstruct backward computation. These include work on the Janus language whose origin dates back to the early 1980s [50, 51], work on sequential flow charts [52], the Inv [53], RFUN [54], and Π [55] reversible functional languages. The key aspect of Janus is that all its constructs, including assignments, are made *bijective* (and hence reversible), and the language does not allow I/O. The Π language constitutes a reversible core programming model which is claimed to be at the heart of linear logic and quantum computation. It is shown in [55] how to translate a conventional first-order functional language with loops to Π, making explicit the information effects implicit in the irreversible computation of a conventional functional program as manipulations of a global heap and garbage dump.

Reversibility in concurrent models has been considered only more recently, starting with the seminal work of Danos and Krivine on RCCS [12]. In contrast to sequential settings, the notion of reversibility is less easy to define, and the key contribution of [12] is to define the criterion of *causal consistency* for semantic reversibility, i.e. the ability to go back in a computation along equivalent concurrent paths. This work later gave rise to several studies (including the one reported in this paper), including [13] which shows how to accommodate a notion of communicating transaction in the RCCS setting, and how using RCCS as a means of specifying such transactions one can gain both in expressivity of specifications and in ease of verification of transactional systems. Phillips and Ulidowski show in [14] how to obtain reversible variants of process calculi defined with GSOS inference rules. [56] showed how the results obtained in [13] can be obtained by means of a universal categorical construction involving categories of fractions and categories of computation paths. Extensions of Danos and Krivine's work on CCS to the (higher-order and first-order) π-calculus appear in our own work [16], and in the recent paper by Cristescu, Krivine and Varacca [20], which defines a labeled transition semantics for a reversible variant of the first-order π-calculus called Rπ. The latter work is closest to ours, but the reversible machinery in Rπ is substantially different: as in RCCS, Rπ processes are built upon simple π processes to which a stack of events, called a memory, is added to keep track of past actions. Every entry in a memory records a past communication event and can be used to trigger backward moves. In contrast, in rhoπ only simple keys are associated to processes, and specific processes are used to record the causal relationships between keys. It is easy to define a reversible variant of the first-order π-calculus using our reversible machinery of keys. One can also define for such a calculus a labeled transition system (LTS) semantics where actions are standard π-calculus actions annotated with keys, but it is less easy to directly compare the two resulting reversible π-calculi. We surmise that the reversible π obtained via our late LTS semantics would indeed be strongly bisimilar to Rπ, whereas the strong back and forth bisimulation associated with the early variant LTS of our reversible π would provide a coinductive characterization of contextual equivalence in rhoπ, but this is left for further study.

Foundational studies of reversible and concurrent computations have been largely inspired by areas such as chemical and biological systems where operations are reversible and only an injection of energy and/or a change of entropy can move the computational system in a desired direction. A reversible variant of CCS to model biological systems is given in [57]. Frequently these systems are *massively concurrent*, i.e. different processes of the same shape are indistinguishable. Thus, unique tags like ours cannot be used, since there is no way to distinguish different instances of the same molecule during interaction. In these systems standard notions of causality and independence of events need to be adapted. Reversible structures [4] allow to model such systems. In reversible structures processes are called *gates*, and are expressed as a sequence of inputs followed by a sequence of outputs. Following the approach of [14], the past computational history of a gate is stored in the gate itself. That is, since gates are a sequence of actions, a cursor "ˆ" is used to point to the next action of a gate. Said otherwise, the cursor ˆ divides a gate into two parts: past actions and future actions. Each time a gate performs a forward (backward) action its cursor is moved forward (backward) by one position. Different output processes (called signals) on the same channel may have the same identifier, hence they are indistinguishable. So it may happen that while computing backward a gate gets back a signal that has not been generated by the gate itself, but it is indistinguishable from it. Reversibility is proven correct even in presence of indistinguishable signals. Another work considering reversible concurrent systems in relation with biological modeling is the recent work by Phillips and Ulidowski [58], which presents a reversible concurrent model where backward moves are controlled by a form of superposition construct. In this model, backward computations are not necessarily causally consistent.

Notions of reversible computation appear also in works on *reversible debuggers* [59, 60, 61**?** , 62] and on computer simulation tools [63]. Two techniques are commonly used in reversible debugging: *replay* and *state saving*. The first one, typical of interpreted languages, consists in re-executing the program to the point at which the programmer wants to get back. This technique can be improved by using periodic or incremental checkpoints, thus reducing the number of instructions that have to be re-played. The second technique consists in saving the entire program state during the computation, and then restoring it when needed. Usually, due to space overhead, the range of actions that can be reverted is limited and it has to be decided before launching the debugging mode. Both the techniques work fine in the sequential setting. In the concurrent setting also information about the *scheduling*, i.e. the order of execution of concurrent processes, has to be taken into account. [64] gives a technique to achieve repeatable execution of highly parallel programs. During execution, the relative order of significant events is saved. Then by imposing the same order during replay, and using the same inputs from the external environment, it is possible to reproduce the same behavior. Building on [18], which shows that one can build primitives to control rhoπ reversibility, [62] shows how to force a concurrent execution back in a causally consistent way so as to undo a specific past action in a way similar to those of reversible debuggers but without impacting non causally related threads. This is in contrast, for instance, with [65], where the user is asked, while debugging, to specify all the actions to be undone and the order in which to undo them. The partial order among concurrent actions induced by the rhoπ tag mechanism can be exploited also in re-playing techniques.

Translating between reversible and irreversible models of computation. Interest for translation between reversible and irreversible models of computation has centered around encoding of irreversible models into reversible ones, or between reversible ones.

As reported in [1], early interest was concerned with the encoding of irreversible computations into a reversible computational model such as a reversible Turing machine or reversible boolean logic. The more recent work by Burhman et al. [66] provides a general upper bound on the tradeoff between time and space that suffices for the simulation by a reversible Turing machine of an irreversible one. Together with a later paper by Vitanyi [67], it provides a useful survey of prior work on this “reversible simulation” problem. The work by Cardelli and Laneve [4], already mentioned, shows that, by disallowing indistinguishable signals, reversible structures can implement the asynchronous version of RCCS. This is stated by a (weak) completeness theorem but nothing is said about the correctness of the encoding. The survey paper [43] relates different reversible models, including reversible Turing machines, reversible boolean logic and reversible cellular automata.

To the best of our knowledge, we are the first in [16] and in this paper to study an encoding of a reversible concurrent model of computation into an irreversible one. In our paper [16] we have defined an encoding very similar to the one presented in this paper, but we were only able to prove the faithfulness of the translation using a weak barbed congruence, which, as discussed in Section 2.5, is a rather coarse equivalence. In this paper, with a slight modification of our encoding, we have been able to prove a much stronger result, which is optimal in the sense that the equivalence we use is a strong back and forth simulation with weak administrative moves, i.e. each forward or backward step in rhoπ translates into a forward or backward step, respectively, modulo administrative moves. Both encodings faithfully implement the reversible calculus, but, as the operational semantics of $\mathsf{rho}\pi$ itself, they are rather wasteful in terms of space. To see this informally, notice first that a forward computation step in $\mathsf{rho}\pi$ requires retaining in a memory the message $a\langle P\rangle$ and the receiver process $a(X) \triangleright Q$ that participated in it. Thus the space overhead of a computation step in reversible HOπ compared to standard HOπ is at least $\|P\|$, the size of the payload of message $a\langle P\rangle$. Now consider the following recursive programs: $P = c(X) \triangleright P \mid a\langle X \mid X\rangle$ and $Q = a(X) \triangleright Q \mid c\langle X \mid X\rangle$. We have $a\langle R\rangle \mid P \mid Q \to P \mid Q \mid c\langle R \mid R\rangle$ so the space overhead of this first step starting from $a\langle R\rangle \mid P \mid Q$ is at least $\|R\|$. On the second step we have $P \mid Q \mid c\langle R \mid R\rangle \to P \mid Q \mid a\langle R \mid R \mid R \mid R\rangle$, so the space overhead of this second step is at least $2\|R\|$. By induction, one can see that the space overhead associated with making the program $a\langle R\rangle \mid P \mid Q$ reversible is at least $2^{n-1}\|R\|$, where n is the number of computation steps taken from the initial state $a\langle R\rangle \mid P \mid Q$. Our encodings are at least as wasteful in terms of space. However we have shown in [22] that the space overhead required to implement a small language close to $\mathsf{rho}\pi$ is only linear in the number of computation steps, and in fact only linear in the number of non-deterministic events occurring during a computation.

Causal semantics and back and forth simulations. For proving the correctness of our encoding, we have used a notion of back and forth bisimulation, where both forward and backward moves are taken into account in the bisimulation game. Different forms of simulations taking into account forward and backward moves have been studied in the past but mostly in the context of verifying standard transition systems. Such notions appear in the late 80's and early 90's in works such as [24, 68]. The two survey papers [69, 70] by Lynch and Vaandrager study the relationships between different kinds of simulations between (standard, timed and untimed) transition systems, including refinements, forward and backward simulations, hybrid forward-backward and backward-forward simulations, and history and prophecy relations. More recently, notions of forward and backward simulation have been studied from a coalgebraic point of view by Hasuo [25].

More related to reversible models of computation are recent works by Phillips and Ulidowski. The paper [71] proposes extensions to event structures to take into account reversibility in transition systems. The paper [72] defines several forms of bisimulations mixing forward and reverse observations, and studies the relationships between various equivalences on stable configuration structures, including step bisimulation, step bisimulation with reverse steps, interleaving bisimulation with reverse steps, and hereditary history-preserving bisimulation. Notably, they show that, in absence of auto-concurrency, interleaving bisimulation with reverse steps is as strong as hereditary history-preserving bisimulation. The latter results illustrates the observational power gained by the ability to take into account backward moves in a bisimulation game. It squares nicely with our result in Section 3.2 that relates our reversible machinery in rhoπ to the causal semantics for π developed by Boreale and Sangiorgi [21], and which states that a causally consistent reversible semantics essentially constitutes a causal semantics for (the forward part of) the calculus. This intuition is further compounded by the work on Rπ [20], which shows that Rπ semantics provides a non-interleaving semantics for the π-calculus that in addition agrees with the causality induced by reductions (τ transitions). Much work remains to be done, however, to better understand the relationships between our reduction semantics for (higher-order) π and the resulting barbed congruence, the LTS semantics of Rπ and its associated bisimilarity, and the various causal semantics of the π calculus that have been developed in the past, including, e.g., [21, 73, 29, 74].

6. Conclusion

We have presented a reversible asynchronous higher-order π-calculus, called rhoπ, which we have shown to be causally consistent. The paper gets its inspiration from Danos and Krivine work [12] and makes three original contributions. The first one is a novel way to introduce reversibility in a process calculus which preserves the classical structural congruence laws of the π-calculus, and which relies on simple name tags for identifying threads and explicit memory processes. Our approach contrasts with the two previous approaches of RCCS [12], that relied on memory stacks as thread tags, and of Phillips and Ulidowski [14], that relied on making the structure of terms in SOS rules static and on keeping track of causality by tagging actions in SOS rules, as well as with the recent work on Rπ, a reversible variant of the π calculus developed by Cristescu, Krivine and Varacca [20]. The paper by Danos et al. [56] provides an abstract categorical analysis of the RCCS constructions, but it leaves intact the question of devising an appropriate "syntactic representation of the reversible history category" for the target formalism (in our case, asynchronous HOπ), which is not entirely trivial. The second contribution of the paper is the analysis, by means of a bisimulation relation called causal correspondence, of the relationship between the causal semantics of the π-calculus developed by Boreale and Sangiorgi [21], and the causality tracking machinery used in our reduction semantics for $\mathsf{rho}\pi$. The third contribution of the paper is a faithful encoding of our reversible HOπ calculus into a variant of HOπ, showing that adding reversibility does not change substantially the expressive power of HOπ. The result obtained in this paper considerably strengthens that obtained in our previous work [16] by showing that, modulo administrative reduction steps, a $\mathsf{rho}\pi$ process and its translation are strong back and forth bisimilar.

References

[1] C. H. Bennett, Notes on the history of reversible computation, IBM Journal of Research and Development 32 (1) (1988) 16–23.

[2] R. Landauer, Irreversibility and heat generated in the computing process, IBM Journal of Research and Development 5 (1961) 183 –191.

[3] M. P. Frank, Introduction to reversible computing: motivation, progress, and challenges, in: 2nd Conference on Computing Frontiers, ACM, 2005, pp. 385–390.

[4] L. Cardelli, C. Laneve, Reversible structures, in: CMSB, ACM, 2011, pp. 131–140.

[5] T. Akgul, V. J. Mooney III, Assembly instruction level reverse execution for debugging, ACM Trans. Softw. Eng. Methodol. 13 (2) (2004) 149–198.

[6] T. Altenkirch, J. Grattage, A functional quantum programming language, in: LICS, IEEE Computer Society, 2005, pp. 249–258.

[7] P. Bishop, Using reversible computing to achieve fail-safety, in: The Eighth International Symposium On Software Reliability Engineering, 1997, pp. 182 –191.

[8] A. Avizienis, J.-C. Laprie, B. Randell, C. E. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Dependable Sec. Comput. 1 (1) (2004) 11–33.

[9] B. Jacobs, F. Piessens, Failboxes: Provably safe exception handling, in: ECOOP, 2009, pp. 470–494.

[10] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, D. B. Johnson, A survey of rollback-recovery protocols in message-passing systems, ACM Comput. Surv. 34 (3) (2002) 375–408.

[11] G. Weikum, G. Vossen, Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery, Morgan Kaufmann, 2002.

[12] V. Danos, J. Krivine, Reversible communicating systems, in: CONCUR, Vol. 3170 of LNCS, Springer, 2004, pp. 292–307.

[13] V. Danos, J. Krivine, Transactions in RCCS, in: CONCUR, Vol. 3653 of LNCS, Springer, 2005, pp. 398–412.

[14] I. C. C. Phillips, I. Ulidowski, Reversing algebraic process calculi, J. Log. Algebr. Program. 73 (1-2) (2007) 70–96.

[15] R. Milner, Communicating and mobile systems - the π-calculus, Cambridge University Press, 1999.

[16] I. Lanese, C. A. Mezzina, J.-B. Stefani, Reversing higher-order pi, in: CONCUR, Vol. 6269 of LNCS, Springer, 2010, pp. 478–493.

[17] D. Sangiorgi, Bisimulation for higher-order process calculi, Inf. Comput. 131 (2) (1996) 141–178.

[18] I. Lanese, C. A. Mezzina, A. Schmitt, J.-B. Stefani, Controlling reversibility in higher-order pi, in: CONCUR, Vol. 6901 of LNCS, Springer, 2011, pp. 297–311.

[19] I. Lanese, M. Lienhardt, C. A. Mezzina, A. Schmitt, J.-B. Stefani, Concurrent flexible reversibility, in: 22nd European Symposium on Programming (ESOP), Vol. 7792 of Lecture Notes in Computer Science, Springer, 2013, pp. 370–390.

[20] I. Cristescu, J. Krivine, D. Varacca, A compositional semantics for the reversible p-calculus, in: LICS, IEEE Computer Society, 2013, pp. 388–397.

[21] M. Boreale, D. Sangiorgi, A fully abstract semantics for causality in the π-calculus, Acta Informatica 35 (5) (1998) 353–400.

[22] M. Lienhardt, I. Lanese, C. A. Mezzina, J.-B. Stefani, A reversible abstract machine and its space overhead, in: FMOODS/FORTE, 2012, pp. 1–17.

[23] D. Sangiorgi, Expressing mobility in process algebras: First-order and higher-order paradigms, PhD thesis CST–99–93, University of Edinburgh (1992).

[24] R. De Nicola, U. Montanari, F. W. Vaandrager, Back and forth bisimulations, in: CONCUR, Vol. 458 of LNCS, Springer, 1990, pp. 152–165.

[25] I. Hasuo, Generic forward and backward simulations, in: CONCUR, Vol. 4137 of LNCS, Springer, 2006, pp. 406–420.

[26] I. C. C. Phillips, I. Ulidowski, A logic with reverse modalities for history-preserving bisimulations, in: EXPRESS, Vol. 64 of EPTCS, 2011, pp. 104–118.

[27] J.-J. Lévy, An algebraic interpretation of the *lambda beta* k-calculus; and an application of a labelled *lambda*-calculus, Theor. Comput. Sci. 2 (1) (1976) 97–114.

[28] G. L. Cattani, P. Sewell, Models for name-passing processes: interleaving and causal, Inf. Comput. 190 (2).

[29] S. Crafa, D. Varacca, N. Yoshida, Compositional event structure semantics for the internal pi-calculus, in: CONCUR, Vol. 4703 of LNCS, Springer, 2007, pp. 317–332.

[30] D. Varacca, N. Yoshida, Typed event structures and the linear pi-calculus, Theor. Comput. Sci. 411 (19) (2010) 1949–1973.

[31] C. Fournet, G. Gonthier, The reflexive chemical abstract machine and the join-calculus, in: POPL, ACM, 1996, pp. 372–385.

[32] C. Fournet, G. Gonthier, The join calculus: A language for distributed mobile programming, in: APPSEM, Vol. 2395 of LNCS, Springer, 2000, pp. 268–332.

[33] R. Milner, Functions as processes, Mathematical Structures in Computer Science 2 (2) (1992) 119–141.

[34] D. Sangiorgi, From lambda to pi; or, rediscovering continuations, Mathematical Structures in Computer Science 9 (4) (1999) 367–401.

[35] H. P. Barendregt, The Lambda Calculus – Its Syntax and Semantics, revised Edition, North-Holland, 1984.

[36] A. Schmitt, J.-B. Stefani, The M-calculus: a higher-order distributed process calculus, in: POPL, ACM, 2003, pp. 50–61.

[37] S. Lenglet, A. Schmitt, J.-B. Stefani, Normal bisimulations in calculi with passivation, in: FOSSACS, Vol. 5504 of LNCS, Springer, 2009, pp. 257–271.

[38] S. M. Abramov, R. Glück, Principles of inverse computation and the universal resolving algorithm, in: The Essence of Computation, Vol. 2566 of Lecture Notes in Computer Science, Springer, 2002.

[39] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, J. F. Terwilliger, Bidirectional transformations: A cross-discipline perspective, in: 2nd Int. Conference on Theory and Practice of Model Transformations (ICMT), Vol. 5563 of Lecture Notes in Computer Science, Springer, 2009.

[40] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, A. Schmitt, Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem, ACM Trans. Program. Lang. Syst. 29 (3).

[41] Y. Lecerf, Machines de turing réversibles. Insolubilité récursive en $n \in n$ de l'équation $n = \theta^n$, où θ est un "isomorphisme de codes", Comptes-Rendus Académie des Sciences 257 (1963) 2597–2600.

[42] C. H. Bennett, Logical reversibility of computation, IBM Journal of Research and Development 17 (6) (1973) 525–532.

[43] K. Morita, Reversible computing and cellular automata - a survey, Theoretical Computer Science 395 (1) (2008) 101–131.

[44] M. Zelkowitz, Reversible execution, Commun. ACM 16 (9).

[45] G. B. Leeman Jr., A formal approach to undo operations in programming languages, ACM Trans. Program. Lang. Syst. 8 (1) (1986) 50–87.

[46] S. Abramsky, A structural approach to reversible computation, Theor. Comput. Sci. 347 (3) (2005) 441–464.

[47] V. Danos, L. Regnier, Reversible, irreversible and optimal lambda-machines, Theor. Comput. Sci. 227 (1-2) (1999) 79–97.

[48] W. E. Kluge, A reversible se(m)cd machine, in: 11th International Workshop on Implementation of Functional Languages (IFL), no. 1868 in Lecture Notes in Computer Science, Springer, 1999.

[49] B. Stoddart, R. Lynas, F. Zeyda, A virtual machine for supporting reversible probabilistic guarded command languages, Electronic Notes in Theoretical Computer Science 253 (6).

[50] T. Yokoyama, R. Glück, A reversible programming language and its invertible self-interpreter, in: PEPM, 2007, pp. 144–153.

[51] T. Yokoyama, H. B. Axelsen, R. Glück, Principles of a reversible programming language, in: 5th Conference on Computing Frontiers, ACM, 2008, pp. 43–54.

[52] T. Yokoyama, H. B. Axelsen, R. Glück, Reversible flowchart languages and the structured reversible program theorem, in: ICALP, Vol. 5126 of LNCS, Springer, 2008, pp. 258–270.

[53] S. Mu, Z. Hu, M. Takeichi, An injective language for reversible computation, in: 7th International Conference Mathematics of Program Construction (MPC), no. 3125 in Lecture Notes in Computer Science, Springer, 2004, pp. 289–313.

[54] T. Yokoyama, H. B. Axelsen, R. Glück, Towards a reversible functional language, in: 3rd International Workshop on Reversible Computation (RC), Vol. 7165 of Lecture Notes in Computer Science, Springer, 2011.

[55] R. P. James, A. Sabry, Information effects, in: 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), ACM, 2012, pp. 73–84.

[56] V. Danos, J. Krivine, P. Sobocinski, General reversibility, in: EXPRESS, Vol. 175, 2007, pp. 75–86.

[57] V. Danos, J. Krivine, Formal molecular biology done in CCS-R, in: BioConcur, Vol. 180 of Electronic Notes in Theoretical Computer Science, Elsevier, 2007, pp. 31–49.

[58] I. Phillips, I. Ulidowski, S. Yuen, A reversible process calculus and the modelling of the erk signalling pathway, in: 4th International Workshop on Reversible Computation (RC), Vol. 7581 of Lecture Notes in Computer Science, 2013.

[59] R. M. Balzer, Exdams: extendable debugging and monitoring system, in: May 14-16, 1969, Spring Joint Computer Conference, AFIPS '69 (Spring), ACM, 1969, pp. 567–580.

[60] S. I. Feldman, C. B. Brown, Igor: A system for program debugging via reversible execution, in: Workshop on Parallel and Distributed Debugging, 1988, pp. 112–123.

[61] B. Boothe, Efficient algorithms for bidirectional debugging, in: ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), ACM, 2000, pp. 299–310.

[62] E. Giachino, I. Lanese, C. A. Mezzina, Causal-consistent reversible debugging, in: 17th International Conference Fundamental Approaches to Software Engineering (FASE), Vol. 8411 of Lecture Notes in Computer Science, 2014, pp. 370–384.

[63] C. D. Carothers, K. S. Perumalla, R. Fujimoto, Efficient optimistic parallel simulations using reverse computation, ACM Transactions on Modeling and Computer Simulation 9 (3) (1999) 224–253.

[64] T. LeBlanc, J. Mellor-Crummey, Debugging parallel programs with instant replay, IEEE Trans. Comput. 36 (4).

[65] J. J. Cook, Reverse execution of java bytecode, Comput. J. 45 (6).

[66] H. Buhrman, J. Tromp, P. M. B. Vitányi, Time and space bounds for reversible simulation, in: 28th International Colloquium on Automata, Languages and Programming (ICALP), Vol. 2076 of Lecture Notes in Computer Science, Springer, 2001.

[67] P. M. B. Vitányi, Time, space, and energy in reversible computing, in: 2nd Conference on Computing Frontiers, ACM, 2005.

[68] N. KLarlund, F. Schneider, Proving nondeterministically specified safety properties using progress measures, Information and Computation 107 (1) (1993) 151–170.

[69] N. A. Lynch, F. W. Vaandrager, Forward and backward simulations: I. untimed systems, Inf. Comput. 121 (2).

[70] N. A. Lynch, F. W. Vaandrager, Forward and backward simulations, ii: Timing-based systems, Inf. Comput. 128 (1).

[71] I. Phillips, I. Ulidowski, Reversibility and asymmetric conflict in event structures, in: 24th International Conference on Concurrency Theory (CONCUR), Vol. 8052 of Lecture Notes in Computer Science, 2013.

[72] I. C. C. Phillips, I. Ulidowski, Reverse bisimulations on stable configuration structures, in: Proceedings 6th Workshop on Structural Operational Semantics (SOS), Vol. 18 of EPTCS, 2009.

[73] P. Degano, C. Priami, Non-interleaving semantics for mobile processes, Theoretical Computer Science 216 (1-2) (1999) 237–270.

[74] S. Crafa, D. Varacca, N. Yoshida, Event structure semantics of the parallel extrusion in the pi-calculus, in: 15th International Conference Foundations of Software Science and Computational Structures (FOSSACS), Vol. 7213 of Lecture Notes in Computer Science, Springer, 2012, pp. 225–239.

Appendix A. Proofs of Section 2

Appendix A.1. Proofs of Section 2.3

We prove in this section the Consistency Preservation Lemma. We need a few auxiliary results first. The lemma below gives a syntactic characterization of forward reductions.

Lemma 36. *Let M, N be configurations. Then $M \twoheadrightarrow N$ iff $M \equiv M'$ and $N' \equiv N$ with:*

$$M' = \nu\tilde{u}.\, \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \mid \prod_{i\in I} \kappa_i : \rho_i \mid \prod_{j\in J} [\mu_j; k_j]$$

$$N' = \nu\tilde{u}, k.\, k : Q\{^P/_X\} \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] \mid \prod_{i\in I} \kappa_i : \rho_i \mid \prod_{j\in J} [\mu_j; k_j]$$

Proof. Let us start with the if direction. The proof is by induction on the derivation of the reduction $\twoheadrightarrow$. We have a case analysis on the last applied rule:

R.Fw: by hypothesis $M = \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q$ and $M \twoheadrightarrow \nu k.\, k : Q\{^P/_X\} \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] = N$. The thesis follows by choosing $M' = M$ and $N' = N$.

R.Eqv: the thesis follows by transitivity of structural congruence.

R.Ctx: the proof is by case analysis on the structure of the context. The proof for the empty context is trivial. If the context is a restriction then we have that $\nu u.\, M \twoheadrightarrow \nu u.\, N$ with $M \twoheadrightarrow N$ as hypothesis. The thesis follows by adding u to $\tilde{u}$. For (left) parallel context we have that $M_1 \mid M \twoheadrightarrow M_1 \mid N$ with $M \twoheadrightarrow N$ as hypothesis. By inductive hypothesis $M \equiv M'$ and $N' \equiv N$ with:

$$M' = \nu\tilde{u}.\, \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \mid \prod_{i\in I} \kappa_i : \rho_i \mid \prod_{j\in J} [\mu_j; k_j]$$
$$N' = \nu\tilde{u}, k.\, k : Q\{^P/_X\} \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] \mid \prod_{i\in I} \kappa_i : \rho_i \mid \prod_{j\in J} [\mu_j; k_j]$$

Also, from Lemma 1 $M_1 \equiv \nu\tilde{v}.\ \prod_{i \in I'}(\kappa_i : \rho_i) \mid \prod_{j \in J'}[\mu_j; k_j]$. Then

$$M_1 \mid M \equiv \nu\tilde{u}, \tilde{v}.\, \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \mid \prod_{i \in I \cup I'} \kappa_i : \rho_i \mid \prod_{j \in J \cup J'} [\mu_j; k_j]$$

$$N'' = \nu\tilde{u}, \tilde{v}, k.\, k : Q\{^P/_X\} \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] \mid \prod_{i \in I \cup I'} \kappa_i : \rho_i \mid \prod_{j \in J \cup J'} [\mu_j; k_j]$$

with $N'' \equiv M_1 \mid N$ as desired. The case of right parallel context is similar.

For the only if direction, the desired reduction can be derived by applying rule (R.Fw) followed by (R.Ctx):

$$\nu\tilde{u}.\, \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \mid \prod_{i \in I} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j] \twoheadrightarrow$$
$$\nu\tilde{u}, k.\, k : Q\{^P/_X\} \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] \mid \prod_{i \in I} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j]$$

The thesis then follows by applying rule (R.Eqv). □

The following lemma is similar to Lemma 36, but it considers backward reductions.

Lemma 37. *Let M, N be configurations. Then $M \rightsquigarrow N$ iff $M \equiv M'$ and $N' \equiv N$ with:*

$$M' = \nu\tilde{u}, k.\, k : R \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] \mid \prod_{i \in I} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j]$$

$$N' = \nu\tilde{u}.\, \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \mid \prod_{i \in I} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j]$$

Proof. Similar to the proof of Lemma 36. □

We can now prove Lemma 3.

Lemma 3 (Consistency preservation). *Let M be a consistent configuration. If $M \to N$ then N is a consistent configuration.*

PROOF. The proof proceeds by case analysis on the derivation of $M \to N$.

Let us consider the case $M \twoheadrightarrow N$. By Lemma 36 we have $M \equiv M'$ and $N' \equiv N$ with $M' = \nu\tilde{u}.\, \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \mid \prod_{i\in I} \kappa_i : \rho_i \mid \prod_{j\in J}[\mu_j; k_j]$ and $\nu\tilde{u}, k.\, k : Q\{^P/_X\} \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] \mid \prod_{i\in I} \kappa_i : \rho_i \mid \prod_{j\in J}[\mu_j; k_j] = N'$. By hypothesis M is a consistent configuration, thus M' is also consistent (consistency is preserved by $\equiv$ since it is defined up to $\equiv$ itself). We have to prove that N' is consistent. The properties 1-4 of Definition 1 check uniqueness of keys. They are all satisfied for existing tags, and they are satisfied by the new tag k since it is a fresh key. The condition 5 holds for the new memory $[\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k]$ because of the form of the continuation. It holds for the other memories by hypothesis. Note that the condition on memories that generated the two threads identified by κ_1 and κ_2 participating to the communication still holds, since the two threads are just moved from an active context to a memory.

The case $M \rightsquigarrow N$ is similar to the previous one, using Lemma 37 instead of Lemma 36. $\square$

Appendix A.2. Proofs of Section 2.4

We prove in this section results relating rhoπ and HOπ reductions. We first prove an auxiliary result relating rhoπ and HOπ structural congruences.

Lemma 38. *For all closed configurations M, N if $M \equiv N$ then $\gamma(M) \equiv_\pi \gamma(N)$.*

PROOF. It is enough to prove that the thesis holds for each axiom (since γ is defined by structural induction). We have a case for each axiom. For the rules E.PARC, E.PARA, E.NILM, E.NEWN, E.NEWC, E.NEWP and E.α there is a corresponding rule in HOπ. Rules E.TAGN and E.TAGP instead reduce to the identity. $\square$

Lemma 4. *For all closed configurations M, N, if $M \twoheadrightarrow N$ then $\gamma(M) \to_\pi \gamma(N)$*

PROOF. By induction on the derivation of $M \twoheadrightarrow N$.

R.Fw: $M = \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \twoheadrightarrow \nu k.\, k : Q\{^P/_X\} \mid [a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] = N$. By definition $\gamma(M) = a\langle P\rangle \mid a(X) \triangleright Q \to_\pi Q\{^P/_X\} = \gamma(N)$.

R.Eqv: $M \twoheadrightarrow N$ with hypothesis $M \equiv M'$, $M' \twoheadrightarrow N'$ and $N' \equiv N$. By using the inductive hypothesis we have that $M' \twoheadrightarrow N'$ implies that $\gamma(M') \to_\pi \gamma(N')$ and since structural equivalence is preserved by γ (by Lemma 38) we can conclude.

R.Ctx: the proof is by induction on the context. The case of the empty context is trivial. The case of a restriction of a key is trivial since the restriction is removed by γ. The case of restriction of a name follows by induction. The case of parallel composition follows by induction since $\gamma(M \mid N) = \gamma(M) \mid \gamma(N)$.

□

To prove Lemma 5 we need a few auxiliary results. The first one characterizes the configurations M such that $\gamma(M) = P$ for a given HOπ process P.

Lemma 39. *Let P be a HOπ process. If $\gamma(M) = P$ and $P \equiv_\pi P'$ with $P' = \nu\tilde{a}.\ \prod_{i\in I} \rho_i$ and $\tilde{a} \subseteq \mathtt{fn}(\prod_{i\in I} \rho_i)$ then $M \equiv \nu\tilde{a}, \tilde{a}', \tilde{k}.\ \prod_{i\in I} \kappa_i : \rho_i \mid \prod_{j\in J} m_j$ with $\tilde{a}' \cap \mathtt{fn}(\prod_{i\in I} \kappa_i : \rho_i) = \emptyset$.*

Proof. By Lemma 1 $M \equiv \nu\tilde{u}.\ \prod_{i'\in I'}(\kappa'_i : \rho'_i) \mid \prod_{j'\in J'} m_{j'} \equiv \nu\tilde{b}, \tilde{h}.\ \prod_{i'\in I'}(\kappa'_i : \rho'_i) \mid \prod_{j'\in J'} m_{j'} = M'$ where we distinguish between names $\tilde{b}$ and keys $\tilde{h}$. By definition $\gamma(M') = \nu\tilde{b}.\ \prod_{i'\in I'} \rho_{i'}$, but since $M \equiv M'$ by Lemma 38 we also have $\gamma(M) \equiv_\pi \gamma(M')$. Since $\gamma(M) = P \equiv_\pi P'$ we have $P' \equiv_\pi \gamma(M')$. Thus we have to prove that if $\nu\tilde{b}.\ \prod_{i'\in I'} \rho_{i'} \equiv_\pi \nu\tilde{a}.\ \prod_{i\in I} \rho_i$ then $\nu\tilde{b}, \tilde{h}.\ \prod_{i'\in I'}(\kappa'_i : \rho'_i) \mid \prod_{j'\in J'} m_{j'} \equiv \nu\tilde{a}, \tilde{a}', \tilde{k}.\ \prod_{i\in I} \kappa_i : \rho_i \mid \prod_{j\in J} m_j$. We can set $\tilde{b} = \tilde{b}_1, \tilde{b}_2$ where $\tilde{b}_1 \subseteq \mathtt{fn}(\prod_{i'\in I'} \rho_{i'})$ and $\tilde{b}_2 \cap \mathtt{fn}(\prod_{i'\in I'} \rho_{i'}) = \emptyset$. We have $\nu\tilde{b}_1.\ \prod_{i'\in I'} \rho_{i'} \equiv_\pi \nu\tilde{a}.\ \prod_{i\in I} \rho_i$ which is derived using only α-conversion and axioms E.ParC, E.ParA and E.NilM. Using the same axioms we can derive also $\nu\tilde{b}, \tilde{h}.\ \prod_{i'\in I'}(\kappa'_i : \rho'_i) \mid \prod_{j'\in J'} m_{j'} \equiv \nu\tilde{b}_2, \tilde{a}, \tilde{h}.\ \prod_{i\in I}(\kappa_i : \rho_i) \mid \prod_{j'\in J'} m'_{j'}$ (memories may be affected by α-conversion). The thesis follows by choosing $\tilde{a}' = \tilde{b}_2$, $\tilde{h} = \tilde{k}$ and $\prod_{j'\in J'} m'_{j'} = \prod_{j\in J} m_j$. □

The next lemma is the inverse of Lemma 38.

Lemma 40. *Let P and P' be HOπ processes. If $P \equiv_\pi P'$ then for each configuration M such that $\gamma(M) = P$ there is a configuration N such that $N \equiv M$ and $\gamma(N) = P'$.*

PROOF. Since both $\equiv_\pi$ and $\equiv$ are equivalence relations, it is enough to show the thesis for derivations of length one. The idea is that each axiom applied on HOπ processes can be applied to each corresponding $\mathsf{rho}\pi$ configuration. This may require additional applications of axioms to deal with the additional $\mathsf{rho}\pi$ structure. For instance, lifting axiom E.NEWC may require additional applications of the same axiom to deal with restrictions on keys. We do not report the detailed case analysis. $\square$

We can finally prove the inverse of Lemma 4.

Lemma 5. *For all closed HOπ processes R, S if $R \rightarrow_\pi S$ then for all closed configurations M such that $\gamma(M) = R$ there is N such that $M \twoheadrightarrow N$ and $\gamma(N) = S$.*

PROOF. By induction on the derivation of the reduction $\rightarrow_\pi$.

Com: $R = a\langle P\rangle \mid a(X) \triangleright Q \rightarrow_\pi Q\{^P/_X\} = S$. Since $\gamma(M) = R$ by Lemma 39 we have that $M \equiv \nu\tilde{a'}, \tilde{k}.\, \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \mid M_1$ with $\tilde{a'} \cap \mathtt{fn}(\prod_{i\in I} \kappa_i : \rho_i) = \emptyset$ and M_1 composed only by memories. We have that $M \twoheadrightarrow \nu\tilde{a'}, \tilde{k}, h.\, h : Q\{^P/_X\} \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; h] \mid M_1 = N$. Also, $\gamma(N) = \nu\tilde{a'}.\, Q\{^P/_X\} \equiv_\pi Q\{^P/_X\} = S$ as required.

Eqv: we have that $R \rightarrow_\pi S$ with hypothesis $R \equiv_\pi R'$, $R' \rightarrow_\pi S'$ and $S' \equiv_\pi S$. Taken M such that $\gamma(M) = R$ from Lemma 40 there is $M' \equiv M$ such that $\gamma(M') = R'$. Then by inductive hypothesis there is M'' such that $M' \twoheadrightarrow M''$ and $\gamma(M'') = S'$. By applying again Lemma 40 we know that there is M''' such that $M''' \equiv M''$ and $\gamma(M''') = S$. The thesis follows by applying rule (R.EQV).

Ctx: we have $\mathbb{C}[R] \rightarrow_\pi \mathbb{C}[S]$ with hypothesis $R \rightarrow_\pi S$. Take M such that $\gamma(M) = \mathbb{C}[R]$. Then there are $\mathbb{C}'$ and M' such that $M = \mathbb{C}'[M']$ and $\gamma(M') = R$. Thus the thesis follows by inductive hypothesis using rule (R.CTX).

$\square$

Appendix B. Proofs of Section 3

Appendix B.1. Proofs of Section 3.1

We prove in this section that rhoπ is causal consistent.

Lemma 9 (Square Lemma). *If $t_1 = M \xrightarrow{\eta_1} M_1$ and $t_2 = M \xrightarrow{\eta_2} M_2$ are two coinitial concurrent transitions, then there exist two cofinal transitions $t_2/t_1 = M_1 \xrightarrow{\eta_2} N$ and $t_1/t_2 = M_2 \xrightarrow{\eta_1} N$.*

Proof. By case analysis on the form of transitions t_1 and t_2.

- $M \xrightarrow{m_1 \twoheadrightarrow} N_1$ and $M \xrightarrow{m_2 \twoheadrightarrow} N_2$. By Lemma 36 if $M \twoheadrightarrow N_1$ then $M \equiv M'$, $N' \equiv N_1$ with:

$$M' = \nu\tilde{u}.\,(\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q) \mid \prod_{i \in I} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j]$$
$$N' = \nu\tilde{u}, k.\,(k : Q\{^P/_X\}) \mid m_1 \mid \prod_{i \in I} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j]$$

and $m_1 = [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k]$. Similarly, if $M \twoheadrightarrow N_2$ then $M \equiv M''$, $N'' \equiv N_2$ with:

$$M'' = \nu\tilde{u}.\,(\kappa'_1 : a'\langle P'\rangle) \mid (\kappa'_2 : a'(X) \triangleright Q') \mid \prod_{i \in I'} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j]$$
$$N'' = \nu\tilde{u}, k'.\,k' : Q'\{^{P'}/_X\} \mid m_2 \mid \prod_{i \in I'} \kappa_i : \rho_i \mid \prod_{j \in J} [\mu_j; k_j]$$

and $m_2 = [\kappa'_1 : a'\langle P'\rangle \mid \kappa'_2 : a'(X) \triangleright Q'; k']$. Since the two transitions are concurrent (by hypothesis) we have that $\{\kappa_1, \kappa_2, k\} \cap \{\kappa'_1, \kappa'_2, k'\} = \emptyset$. Thus, we have:

$$M \equiv \nu\tilde{u}.\,(\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q) \mid (\kappa'_1 : a'\langle P'\rangle) \mid (\kappa'_2 : a'(X) \triangleright Q') \mid \prod_{i \in I''} \kappa_i : \rho_i \mid \prod_{j \in J} m_j$$
$$N_1 \equiv \nu\tilde{u}, k.\,(\kappa'_1 : a'\langle P'\rangle) \mid (\kappa'_2 : a'(X) \triangleright Q') \mid \prod_{i \in I''} \kappa_i : \rho_i \mid (\prod_{j \in J} m_j) \mid m_1$$

with $m_1 = [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k]$ and

$$N_2 \equiv \nu\tilde{u}, k'.\,(\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q) \mid \prod_{i \in I''} \kappa_i : \rho_i \mid (\prod_{j \in J} m_j) \mid m_2$$

We have that $N_2 \xrightarrow{m_1 \twoheadrightarrow} \nu\tilde{u}, k', k.\ \prod_{i \in I''} \kappa_i : \rho_i \mid (\prod_{j \in J} m_j) \mid m_2 \mid m_1$ and $N_1 \xrightarrow{m_2 \twoheadrightarrow} \nu\tilde{u}, k', k.\ \prod_{i \in I''} \kappa_i : \rho_i \mid (\prod_{j \in J} m_j) \mid m_2 \mid m_1$, as desired.

- $M \xrightarrow{m_1 \rightsquigarrow} N_1$ and $M \xrightarrow{m_2 \twoheadrightarrow} N_2$. Since $M \xrightarrow{m_1 \rightsquigarrow} N_1$ by Lemma 37 $M \equiv M'$ with:

$$M' = \nu\tilde{u}, k.\, k : R \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] \mid \prod_{i\in I'} \kappa_i : \rho_i \mid \prod_{j\in J'} m_j$$
$$N_1 \equiv \nu\tilde{u}.\, \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \mid \prod_{i\in I} \kappa_i : \rho_i \mid \prod_{j\in J} m_j$$

and since $M \xrightarrow{m_2 \twoheadrightarrow} N_2$ then by Lemma 36 $M \equiv M''$ with:

$$M'' = \nu\tilde{u}.\, \kappa_1' : a'\langle P'\rangle \mid \kappa_2' : a'(X) \triangleright Q' \mid \prod_{i\in I} \kappa_i : \rho_i \mid \prod_{j\in J'} m_j$$
$$N_2 \equiv \nu\tilde{u}, k'.\, k' : Q'\{{}^{P'}/_X\} \mid m_2' \mid \prod_{i\in I} \kappa_i : \rho_i \mid \prod_{j\in J'} [\mu_j; k_j]$$

and $m_2' = [\kappa_1' : a\langle P'\rangle \mid \kappa_2' : a(X) \triangleright Q'; k']$. By hypothesis the two transitions are concurrent so k is neither equal nor a suffix of κ_1' or κ_2'. Thus, we have that

$$M \equiv \nu\tilde{u}, k'.\, k : R \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] \mid (\kappa_1' : a'\langle P'\rangle) \mid$$
$$(\kappa_2' : a'(X) \triangleright Q') \mid \prod_{i\in I''} \kappa_i : \rho_i \mid \prod_{j\in J''} m_j$$
$$N_1 \equiv \nu\tilde{u}.\, \kappa_1' : a'\langle P'\rangle \mid \kappa_2' : a'(X) \triangleright Q' \mid \kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q \mid$$
$$\prod_{i\in I''} \kappa_i : \rho_i \mid \prod_{j\in J''} m_j$$
$$N_2 \equiv \nu\tilde{u}, k', k.\, m_2' \mid k' : Q'\{{}^{P'}/_X\} \mid [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k] \mid$$
$$k : R \mid \prod_{i\in I''} \kappa_i : \rho_i \mid \prod_{j\in J''} m_j$$

We have that:

$$N_2 \xrightarrow{m_1 \rightsquigarrow} \nu\tilde{u}, k'.\, (\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q) \mid m_2' \mid k' : Q'\{{}^{P'}/_X\} \mid$$
$$\prod_{i\in I''} \kappa_i : \rho_i \mid \prod_{j\in J''} m_j$$
$$N_1 \xrightarrow{m_2 \twoheadrightarrow} \nu\tilde{u}, k'.\, (\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q) \mid m_2' \mid k' : Q'\{{}^{P'}/_X\} \mid$$
$$\prod_{i\in I''} \kappa_i : \rho_i \mid \prod_{j\in J''} m_j$$

as desired.

- $M \xrightarrow{m_1\twoheadrightarrow} N_1$ and $M \xrightarrow{m_2\rightsquigarrow} N_2$, similar to the case above.
- $M \xrightarrow{m_1\rightsquigarrow} N_1$ and $M \xrightarrow{m_2\rightsquigarrow} N_2$, similar to the first case.

□

Lemma 10 (Rearranging lemma). *Let σ be a trace. There exist forward traces σ' and σ'' such that $\sigma \asymp \sigma'_\bullet; \sigma''$.*

Proof. The proof is by lexicographic induction on the length of σ and on the distance between the beginning of σ and the earliest pair of transitions in σ of the form $t; t'_\bullet$ (where t and t' are forward). If there is no such pair we are done. If there is one, we have two possibilities: either t and t' are concurrent, or they are in conflict. In the first case, we can swap them by using Lemma 9, resulting in a later earliest contradicting pair, and by induction the result follows since swapping transitions keeps the total length constant. In the second case we have that there is a conflict on a tag κ. We have two cases: either the memory involved in the two transitions is the same or not. In the first case we have $t = t'$, and we can apply the Loop lemma removing $t; t_\bullet$. Hence the total length of σ decreases and again by induction the result follows. In the second case thanks to the property of consistent configurations the only possible conflict is between the thread tag of a memory and a tag in the configuration part of the other memory. Assume a conflict between the thread tag of memory m of transition t and a tag in the configuration part of memory m' of transition $t'_\bullet$. In this case t has created a memory of the form $[\delta_1 : a\langle P\rangle \mid \gamma_1 : a(X) \triangleright Q; k_1]$ and a process $k_1 : R$. Thus from conditions 1 and 4 in the definition of consistent configuration this case never happens. Assume now the opposite case: a conflict between the thread tag of memory m' and a tag in the configuration part of memory m. In this case transition $t'_\bullet$ deletes a memory of the form $[\delta_2 : a\langle P\rangle \mid \gamma_2 : a(X) \triangleright Q; k_2]$, but this requires having a process $k_2 : R$. Again from conditions 1 and 4 this case never happens. □

Lemma 11 (Shortening lemma). *Let σ_1, σ_2 be coinitial and cofinal traces, with σ_2 forward. Then, there exists a forward trace σ'_1 of length at most that of σ_1 such that $\sigma'_1 \asymp \sigma_1$.*

Proof. We prove this lemma by induction on the length of σ_1. If σ_1 is a forward trace we are already done.

Otherwise by Lemma 10 we can write σ_1 as $\sigma_\bullet;\sigma'$ (with σ and σ' forward). Let $t_\bullet;t'$ be the only two successive transitions in σ_1 with opposite direction, with m_1 belonging to $t_\bullet$. Since m_1 is removed by $t_\bullet$ then m_1 has to be put back by another forward transition otherwise this difference will stay visible since σ_2 is a forward trace. Let t_1 be the earliest such transition in σ_1. Since it is able to put back m_1 it has to be the exact opposite of $t_\bullet$, so $t_1 = t$. Now we can swap t_1 with all the transitions between t_1 and $t_\bullet$, in order to obtain a trace in which t_1 and $t_\bullet$ are adjacent. To do so we use the Square Lemma (Lemma 9), since all the transitions in between are concurrent. Assume in fact that there is a transition involving memory m_2 which is not concurrent to t_1, with $\lambda(m_1) = \{\delta_1, \gamma_1, k_1\}$, $\lambda(m_2) = \{\delta_2, \gamma_2, k_2\}$. Thanks to consistency conditions the only possible conflicts are (1) between k_1 and δ_2 or between k_1 and γ_2 or (2) between k_2 and δ_1 or k_2 and γ_1. The first case can never happen since k_1 is fresh (generated by the forward rule) and thus cannot coincide nor been a prefix of γ_2 or δ_2. Similarly the second case can never happen since k_2 is fresh and thus cannot occur in m_1. When $t_\bullet$ and t are adjacent we can remove both of them using $\asymp$. The resulting trace is shorter, thus the thesis follows by inductive hypothesis. □

Theorem 1 (Causal consistency). *Let σ_1 and σ_2 be coinitial traces, then $\sigma_1 \asymp \sigma_2$ if and only if σ_1 and σ_2 are cofinal.*

Proof. By construction of $\asymp$, if $\sigma_1 \asymp \sigma_2$ then σ_1 and σ_2 must be coinitial and cofinal, so this direction of the theorem is verified. Now we have to prove that σ_1 and σ_2 being coinitial and cofinal implies that $\sigma_1 \asymp \sigma_2$. By Lemma 10 we know that the two traces can be written as composition of a backward trace and a forward one. The proof is by lexicographic induction on the sum of the lengths of σ_1 and σ_2 and on the distance between the end of σ_1 and the earliest pair of transitions t_1 in σ_1 and t_2 in σ_2 which are not equal. If all the transitions are equal then we are done. Otherwise we have to consider three cases depending on the direction of the two transitions.

t_1 **forward and** t_2 **backward:** we have $\sigma_1 = \sigma_\bullet;t_1;\sigma'$ and $\sigma_2 = \sigma_\bullet;t_2;\sigma''$. Moreover we know that $t_1;\sigma'$ is a forward trace, so we can apply the Lemma 11 to the traces $t_1;\sigma'$ and $t_2;\sigma''$ (since σ_1 and σ_2 are coinitial and cofinal by hypothesis, also $t_1;\sigma'$ and $t_2;\sigma''$ are coinitial and cofinal) and we obtain that $t_2; \sigma''$ has a shorter equivalent forward trace and so also σ_2 has a shorter equivalent forward trace. We can conclude by induction.

t_1 and t_2 forward: by assumption the two transitions are different. If they are not concurrent then they should conflict on a thread process $\kappa : P$ that they both consume and store in different memories. Since the two traces are cofinal there should be t_2' in σ_2 creating the same memory as t_1. However no other process $\kappa : P$ is ever created in σ_2 thus this is not possible. So we can assume that t_1 and t_2 are concurrent. Again let t_2' be the transition in σ_2 creating the same memory of t_1. We have to prove that t_2' is concurrent to all the previous transitions. This holds since no previous transition can remove one of the processes needed for triggering t_2' and since forward transitions can never conflict on k. Thus we can repetitively apply the Square Lemma to derive a trace equivalent to σ_2 where t_2 and t_2' are consecutive. We can apply a similar transformation to σ_1. Now we can apply the Square Lemma to t_1 and t_2 to have two traces of the same length as before but where the first pair of different transitions is closer to the end. The thesis follows by inductive hypothesis.

t_1 and t_2 backward: t_1 and t_2 cannot remove the same memory. Let m_1 be the memory removed by t_1. Since the two traces are cofinal, either there is another transition in σ_1 putting back the memory or there is a transition t_1' in σ_2 removing the same memory. In the first case, t_1 is concurrent to all the backward transitions following it, but the ones that consume processes generated by it. All the transitions of this kind have to be undone by corresponding forward transitions (since they are not possible in σ_2). Consider the last such transition: we can use the Square Lemma to make it the last backward transition. The forward transition undoing it should be concurrent to all the previous forward transitions (the reason is the same as in the previous case). Thus we can use the Square Lemma to make it the first forward transition. Finally we can apply the simplification rule $t_\bullet; t \asymp \epsilon_{\texttt{target}(t)}$ to remove the two transitions, thus shortening the trace. The thesis follows by inductive hypothesis.

$\square$

Appendix B.2. Proofs of Section 3.2

Appendix B.2.1. Labelled transition system semantics for choπ

In [21], Boreale and Sangiorgi do not define their causal π-calculus via a reduction semantics but with a labelled transition system. To show that our notion of causal process indeed corresponds to that of Boreale and Sangiorgi, we present here a labelled transition system semantics for choπ directly adapted from [21] to our higher-order context. We then prove that the labelled transition system semantics and the reduction semantics for choπ are in agreement.

The labelled transition system semantics of choπ is given in Figure B.10, where α ranges over channel names a and their complements of the form $\overline{a}$. The rules in Figure B.10 are a direct adaptation of the rules in [21] to the asynchronous higher-order π, with the use of *concretions* and *abstractions*, following Milner and Sangiorgi [15, 17]. A concretion takes the form $\nu\tilde{a}.\,\langle P\rangle A$, where $\tilde{a} \subseteq \mathtt{fn}(P)$. An abstraction takes the form $(X)A$. We use C, D and their decorated variants to range over concretions, and F, G and ther decorated variants to range over abstractions. We call agent an abstraction, a concretion or a causal process, and by abuse of notation, we also use A, B and their decorated variants to range over agents. The application operator $\bullet$ is defined by the following rule (where by convention $\tilde{a} \cap \mathtt{fn}(A) = \emptyset$):

$$(X)A \bullet \nu\tilde{a}.\,\langle P\rangle B = \nu\tilde{a}.\,A\{^{P}/_{X}\} \mid B$$

In addition, we define in Figure B.11 operations $\nu a.\,A$, $A \mid B$, $B \mid A$, and $K :: A$ for arbitrary agents A and causal processes B. Finally, we define inductively the operation of substitution of a key k by a set of keys K in a causal process as follows:

$$\begin{array}{ll}
(K' \cup \{k\})\{^{K}/_{k}\} = K' \cup K & \text{if } k \notin K' \\
K'\{^{K}/_{k}\} = K' & \text{if } k \notin K' \\
P\{^{K}/_{k}\} = P & \\
(K' :: A)\{^{K}/_{k}\} = K'\{^{K}/_{k}\} :: A\{^{K}/_{k}\} & \\
(\nu a.\,A)\{^{K}/_{k}\} = \nu a.\,(A\{^{K}/_{k}\}) & \\
(A_1 \mid A_2)\{^{K}/_{k}\} = (A_1\{^{K}/_{k}\}) \mid (A_2\{^{K}/_{k}\}) &
\end{array}$$

The agreement between the two semantics is given by the following result.

Proposition 2. *$A \xrightarrow{\tau}\equiv A'$ if and only if $A \to A'$.*

$$(\textsc{Out})\ a\langle P\rangle \xrightarrow[\emptyset;k]{\overline{a}} \langle P\rangle\mathbf{0} \qquad (\textsc{In})\ a(X)\triangleright P \xrightarrow[\emptyset;k]{a} \{k\} :: (X)P$$

$$(\textsc{Cau})\ \frac{A \xrightarrow[K;k]{\alpha} A'}{K' :: A \xrightarrow[K\cup K';k]{\alpha} K' :: A'} \qquad (\textsc{Res})\ \frac{A \xrightarrow[K;k]{\alpha} A' \qquad \mathtt{fn}(\alpha) \neq a}{\nu a.\, A \xrightarrow[K;k]{\alpha} \nu a.\, A'}$$

$$(\textsc{ParL})\ \frac{A_1 \xrightarrow[K;k]{\alpha} A_1'}{A_1 \mid A_2 \xrightarrow[K;k]{\alpha} A_1' \mid A_2} \qquad (\textsc{ParR})\ \frac{A_1 \xrightarrow[K;k]{\alpha} A_1'}{A_2 \mid A_1 \xrightarrow[K;k]{\alpha} A_2 \mid A_1'}$$

$$(\text{T-}\textsc{cau})\ \frac{A \xrightarrow{\tau} A'}{K :: A \xrightarrow{\tau} K :: A'} \qquad (\text{T-}\textsc{res})\ \frac{A \xrightarrow{\tau} A'}{\nu a.\, A \xrightarrow{\tau} \nu a.\, A'}$$

$$(\text{T-}\textsc{parl})\ \frac{A_1 \xrightarrow{\tau} A_1'}{A_1 \mid A_2 \xrightarrow{\tau} A_1' \mid A_2} \qquad (\text{T-}\textsc{parr})\ \frac{A_1 \xrightarrow{\tau} A_1'}{A_2 \mid A_1 \xrightarrow{\tau} A_2 \mid A_1'}$$

$$(\textsc{ComL})\ \frac{A_1 \xrightarrow[K_1;k]{\overline{a}} C \qquad A_2 \xrightarrow[K_2;k]{a} F \qquad k \notin \mathtt{fk}(A_1, A_2)}{A_1 \mid A_2 \xrightarrow{\tau} (F \bullet C)\{^{K_1}/_{k}\}}$$

$$(\textsc{ComR})\ \frac{A_1 \xrightarrow[K_1;k]{\overline{a}} C \qquad A_2 \xrightarrow[K_2;k]{a} F \qquad k \notin \mathtt{fk}(A_1, A_2)}{A_2 \mid A_1 \xrightarrow{\tau} (F \bullet C)\{^{K_1}/_{k}\}}$$

Figure B.10: Transition system rules for $\mathsf{cho}\pi$

$$\nu a.\, ((X)A) = (X)\nu a.\, A \qquad \nu a.\, (\nu\tilde{c}.\, \langle P\rangle A) = \nu a, \tilde{c}.\, \langle P\rangle A \quad \text{if } a \in \mathtt{fn}(P)$$
$$\nu a.\, (\nu\tilde{c}.\, \langle P\rangle A) = \nu\tilde{c}.\, \langle P\rangle \nu a.\, A \quad \text{if } a \notin \mathtt{fn}(P)$$

$$((X)A) \mid B = (X)(A \mid B) \qquad B \mid ((X)A) = (X)(B \mid A)$$
$$(\nu c.\, \langle P\rangle A) \mid B = \nu c.\, \langle P\rangle(A \mid B) \qquad B \mid (\nu c.\, \langle P\rangle A) = \nu c.\, \langle P\rangle(B \mid A)$$

$$K :: ((X)A) = (X)K :: A \qquad K :: (\nu c.\, \langle P\rangle A) = \nu c.\, \langle P\rangle K :: A$$

Figure B.11: Operations on agents

The proof of this proposition is long but completely standard. We give details below.

Note that, thanks to the agreement with the reduction semantics, $A \downarrow_\alpha$ if and only if $A \xrightarrow[K;k]{\alpha} A'$ for some k, A'.

Appendix B.2.2. Proof of Proposition 2

We prove in this section the equivalence between the labelled transition system semantics and the reduction semantics for $\mathsf{cho}\pi$. We start with a few additional notions used in the proofs.

We extend the structural congruence relation $\equiv$ to agents using the additional rules below (considering that rule E.α extends to agents as well):

$$\text{(E-Cnc)}\ \frac{P \equiv Q \qquad A \equiv B}{\nu\tilde{a}.\,\langle P\rangle A \equiv \nu\tilde{a}.\,\langle Q\rangle B} \qquad\qquad \text{(E-Abs)}\ \frac{A \equiv B}{(X)A \equiv (X)B}$$

An easy induction on the derivation of $A \equiv B$ gives us the following lemmas.

Lemma 41. *For all agents A, B and process P, if $A \equiv B$ then $A\{^P/_X\} \equiv B\{^P/_X\}$.*

Lemma 42. *For all agents A, B, $A \equiv B$ implies $A\{^K/_k\} \equiv B\{^K/_k\}$.*

Using Lemma 42, an easy induction on the structure of a causal process gives us the following lemma:

Lemma 43. *For all causal processes A, $k \notin K$, $(K :: A)\{^{K \cup K'}/_k\} \equiv K :: A\{^{K'}/_k\}$.*

Lemma 44. *If $F \equiv F'$ and $C \equiv C'$, then $F \bullet C \equiv F' \bullet C'$.*

Proof. We must have $C = \nu\tilde{a}.\,\langle P\rangle A_2$, $F = (X)A_1$, $C' = \nu\tilde{a}.\,\langle P'\rangle A_2'$, $F' = (X)A_1'$, with $P \equiv P'$, $A_1 \equiv A_1'$ and $A_2 \equiv A_2'$. Now

$$\begin{aligned}
F \bullet C &= A_1\{^P/_X\} \mid A_2 && \text{definition of } \bullet \\
&\equiv A_1\{^{P'}/_X\} \mid A_2 && \text{congruence of } \equiv \\
&\equiv A_1'\{^{P'}/_X\} \mid A_2' && \text{Lemma 41 and congruence of } \equiv \\
&= F' \bullet C' && \text{definition of } \bullet
\end{aligned}$$

. □

Lemma 45. *For all agents F, C, $K :: (F \bullet C) \equiv (K :: F) \bullet (K :: C)$*

PROOF. Let $F = (X)A$ and $C = \nu\tilde{c}.\langle P\rangle B$. We compute:

$$K :: (F \bullet C) = K :: \nu\tilde{c}.\, A\{^P/_X\} \mid B \equiv \nu\tilde{c}.\, K :: A\{^P/_X\} \mid K :: B$$
$$(K :: F) \bullet (K :: C) = ((X)K :: A) \bullet (\nu\tilde{c}.\, K :: B) = \nu\tilde{c}.\, K :: A\{^P/_X\} \mid K :: B$$

□

An easy induction on the derivation of $A \xrightarrow[K;k]{\alpha} A'$ gives us the following lemma:

Lemma 46. *For any causal process A, and any keys k, k', if $A \xrightarrow[K;k]{\alpha} A'$, then $A \xrightarrow[K;k']{\alpha} A'$.*

We can then prove the main lemmas towards the agreement proposition:

Lemma 47. *For all A, A', B, α, K, k, if $A \xrightarrow[K;k]{\alpha} A'$ and $B \equiv A$, then there exists $B' \equiv A'$ such that $B \xrightarrow[K;k]{\alpha} B'$.*

PROOF. The proof is lengthy but completely standard. It proceeds by induction on the derivation of $B \equiv A$. We just detail the non-classical cases arising from the rules in Figure 4.

- (E-PAR): in this case, $A = K' :: A_1 \mid A_2$ and $B = K' :: A_1 \mid K :: A_2$. $A \xrightarrow[K;k]{\alpha} A'$ can only have been derived via (CAU), with $A_1 \mid A_2 \xrightarrow[K'';k]{\alpha} A''$ and $K = K' \cup K''$, $A' = K' :: A''$. In turn, $A_1 \mid A_2 \xrightarrow[K'';k]{\alpha} A''$ can only have been derived via (E-PARL) or (E-PARR). We check only the case (E-PARL) as the other is similar. We thus have $A_1 \xrightarrow[K'';k]{\alpha} A_1'$ with $A'' = A_1' \mid A_2$. Using rule (CAU), we now have $K' :: A_1 \xrightarrow[K' \cup K'';k]{\alpha} K' :: A_1'$. Using rule (PARL), we obtain

$$B = K' :: A_1 \mid K' :: A_2 \xrightarrow[K;k]{\alpha} K' :: A_1' \mid K' :: A_2$$

By (E-PAR), we have $K' :: A_1' \mid K' :: A_2 \equiv K' :: A_1' \mid A_2 = A'$, and thus we have found $B' = K' :: A_1' \mid K' :: A_2$, such that $B \xrightarrow[K;k]{\alpha} B'$ and $B' \equiv A'$, as required.

- (E-Cau): in this case, $A = K_1 :: K_2 :: A_1$ and $B = K_1 \cup K_2 :: A_1$. $A \xrightarrow[K;k]{\alpha} A'$ can only have been derived via (Cau), with $K_2 :: A_1 \xrightarrow[K_1';k]{\alpha} A_1'$ and $K = K_1 \cup K_1'$, $A' = K_1 :: A_1'$. In turn, the latter transition can only have been obtained via (Cau), with $A_1 \xrightarrow[K_2';k]{\alpha} A_1''$ and $K_1' = K_2 \cup K_2'$, $A_1' = K_2 :: A_1''$. Using (Cau) we get $K_1 \cup K_2 :: A_1 \xrightarrow[K_1 \cup K_2 \cup K_2';k]{\alpha} K_1 \cup K_2 :: A_1''$ and thus $B = K_1 \cup K_2 :: A_1 \xrightarrow[K;k]{\alpha} K_1 \cup K_2 :: A_1''$. By rule (E-cau), we have $K_1 \cup K_2 :: A_1'' \equiv K_1 :: K_2 :: A_1'' = A'$, and thus we have found $B' = K_1 \cup K_2 :: A_1''$ such that $B \xrightarrow[K;k]{\alpha} B'$ and $B' \equiv A'$, as required.

- (E-res): in this case, $A = K' :: \nu a.\, A_1$ and $B = \nu a.\, K' :: A_1$. $A \xrightarrow[K;k]{\alpha} A'$ can only have been derived via (Cau), with $\nu a.\, A_1 \xrightarrow[K'';k]{\alpha} A''$, $K = K' \cup K''$, and $A' = K' :: A''$. In turn, the latter transition can only have been derived via (Res), with $A_1 \xrightarrow[K'';k]{\alpha} A_1'$, $a \neq \mathtt{fn}(\alpha)$, and $A'' = \nu a.\, A_1'$. Using (Cau) we get $K' :: A_1 \xrightarrow[K' \cup K'';k]{\alpha} K' :: A_1'$, and using (Res) we get $B = \nu a.\, K' :: A_1 \xrightarrow[K;k]{\alpha} \nu a.\, K' :: A_1'$. By (E-res) we get $\nu a.\, K' :: A_1' \equiv K' :: \nu a.\, A_1' = A'$, and thus we have found $B' = \nu a.\, K' :: A_1'$ such that $B \xrightarrow[K;k]{\alpha} B'$ and $B' \equiv A'$, as required.

- (E-nil): in this case, $A = \emptyset :: B$. $A \xrightarrow[K;k]{\alpha} A'$ can only have been derived via (Cau), with $B \xrightarrow[K;k]{\alpha} B'$ and $A' = \emptyset :: B'$. Now by (E-nil) we have $A' \equiv B'$, as required.

□

Lemma 48. *For all A, A', B, if $A \xrightarrow{\tau} A'$ and $A \equiv B$, then there exists $B' \equiv A'$ such that $B \xrightarrow{\tau} B'$.*

Proof. The proof proceeds by induction on the derivation of $A \equiv B$. Again, we give details only for the non-classical cases arising from the rules in Figure 4.

- (E.par): in this case, $A = K :: A_1 \mid A_2$ and $B = K :: A_1 \mid K :: A_2$. $A \xrightarrow{\tau} A'$ can only have been derived via (T-cau), with $A_1 \mid A_2 \xrightarrow{\tau} A''$

and $A' = K :: A''$. In turn, the latter transition could only have been derived via (T-PARL), (T-PARR), (COML), or (COMR). We check the different cases:

- (T-PARL): in this case, we have $A_1 \xrightarrow{\tau} A_1'$, $A'' = A_1' \mid A_2$, and $A' = K :: A_1' \mid A_2$. Applying (T-CAU) and (T-PARL), we get $B \xrightarrow{\tau} K :: A_1' \mid K :: A_2$. Now, by (E-CAU) $K :: A_1' \mid K :: A_2 \equiv K :: A_1' \mid A_2 = A'$, hence we have found $B' = K :: A_1' \mid K :: A_2$ as required.
- (T-PARR): this case is similar to the (T-PARL) one.
- (COML): in this case, $A_1 \xrightarrow[K_1;k]{\overline{a}} C$, $A_2 \xrightarrow[K_2;k]{a} F$, $A'' = F\{^{K_1}/_k\} \bullet C$, $k \notin \mathtt{fk}(A_1, A_2)$. Using (CAU) twice and (COML) we get $B \xrightarrow{\tau} B'$, with $B' = (K :: F \bullet K :: C)\{^{K \cup K_1}/_k\}$. Now using Lemma 42 and Lemma 45, we get $B' \equiv (K :: F \bullet C)\{^{K \cup K_1}/_k\}$ and by Lemma 43 (note that $k \notin K$ for k must not be in $\mathtt{fk}(K :: A_1, K :: A_2)$ for applying (COML), a condition which can always be met thanks to Lemma 46) $B' \equiv K :: (F \bullet C)\{^{K_1}/_k\} = A'$, as required.
- (COMR): this case is similar to the (COML) one.

- (E-CAU): in this case, $A = K_1 :: K_2 :: A_1$ and $B = K_1 \cup K_2 :: A_1$. $A \xrightarrow{\tau} A'$ can only have been derived via (T-CAU) twice, leading to $A_1 \xrightarrow{\tau} A_1'$ and $A' = K_1 :: K_2 :: A_1'$. Now, applying (T-CAU) we get $B \xrightarrow{\tau} K_1 \cup K_2 :: A_1'$ and by (E-CAU) we get $B \equiv K_1 :: K_2 :: A_1' = A'$, as required.
- (E-RES): the reasoning proceeds like above, exploiting (RES) and (E-RES).
- (E-NIL): immediate.

□

Lemma 49. *For all causal processes A, the following properties hold:*

1. *If $A \xrightarrow[K;k]{\overline{a}} \nu\tilde{c}.\, \langle P \rangle B$ then $A \equiv \nu\tilde{c}.\, (K :: a\langle P \rangle) \mid B$.*
2. *If $A \xrightarrow[K;k]{a} F$ then $A \equiv \nu\tilde{c}.\, (K :: a(X) \triangleright Q) \mid B$ for some $\tilde{c}, Q, B$, and $F \equiv (X)\nu\tilde{c}.\, (K \cup \{k\} :: Q) \mid B$.*

3. *If* $A \xrightarrow{\tau} A'$ *then* $A \equiv \nu\tilde{c}.\,(K_1 :: a\langle P\rangle) \mid (K_2 : a(X) \triangleright Q) \mid B$ *for some* $\tilde{c}, K_1, K_2, a, P, Q, B,$ *and* $A' \equiv \nu\tilde{c}.\,(K_1 \cup K_2 :: Q\{^P/_X\}) \mid B$.

Proof. We prove property 1 by induction on the derivation of $A \xrightarrow[K;k]{\bar{a}} C$, where $C = \nu\tilde{c}.\,\langle P\rangle B$ for some $\tilde{c}, P, B$.

- (Out): in this case $A = a\langle P\rangle$, $K = \emptyset$ and $C = \langle P\rangle\mathbf{0}$, thus $A \equiv \emptyset : a\langle P\rangle \mid \mathbf{0}$, as required.

- (Cau): in this case $A = K_1 :: A_1$, $K = K_1 \cup K_2$, $C = K_1 :: C_1$, and $A_1 \xrightarrow[K_2;k]{\bar{a}} C_1$. Assume $C_1 = \nu\tilde{c}.\,\langle P\rangle B$, then $C = \nu\tilde{c}.\,\langle P\rangle K_1 :: B$. By induction assumption, we have $A_1 \equiv \nu\tilde{c}.\,(K_2 :: a\langle P\rangle) \mid B$. Thus $A = K_1 :: A_1 \equiv \nu\tilde{c}.\,(K_1 \cup K_2 :: a\langle P\rangle) \mid (K_1 :: B)$, as required.

- (ParL): in this case, $A = A_1 \mid A_2$, $A_1 \xrightarrow[K;k]{\bar{a}} C_1$, and $C = C_1 \mid A_2$. Assume $C_1 = \nu\tilde{c}.\,\langle P\rangle B$, then $C = \nu\tilde{c}.\,\langle P\rangle(B \mid A_2)$. By induction assumption, we have $A_1 \equiv \nu\tilde{c}.\,K :: a\langle P\rangle \mid B$. Thus, $A = A_1 \mid A_2 \equiv \nu\tilde{c}.\,(K :: a\langle P\rangle) \mid (B \mid A_2)$, as required.

- (ParR): this case is similar to the (ParL) one.

- (Res): in this case, $A = \nu e.\,A_1$, $e \neq a$, $A_1 \xrightarrow[K;k]{\bar{a}} C_1$, and $C = \nu e.\,C_1$. Assume $C_1 = \nu\tilde{c}.\,\langle P\rangle B$, then $C = \nu e.\,\nu\tilde{c}.\,\langle P\rangle B$. By induction assumption $A_1 \equiv \nu\tilde{c}.\,K :: a\langle P\rangle \mid B$. Thus $A = \nu e.\,A_1 \equiv \nu e, \tilde{c}.\,K :: a\langle P\rangle \mid B$. We now have two cases to consider, according to whether $e \in \mathtt{fn}(P)$ or not. If $e \in \mathtt{fn}(P)$, we have $C = \nu e, \tilde{c}.\,\langle P\rangle B$, and we are done. If not, we have $C = \nu\tilde{c}.\,\langle P\rangle \nu e.\,B$ and $A \equiv \nu\tilde{c}.\,K :: a\langle P\rangle \mid \nu e.\,B$, as required.

We prove property 2 by induction on the derivation of $A \xrightarrow[K;k]{a} F$.

- (In): in this case, $A = a(X) \triangleright Q$, $K = \emptyset$, and $F = \{k\} :: (X)Q$. Thus we have $A \equiv \emptyset :: a(X) \triangleright Q$ and $F \equiv \emptyset \cup \{k\} :: (X)Q$, as required.

- (Cau): in this case, $A = K_1 :: A_1$, $A_1 \xrightarrow[K_2;k]{a} F_1$, $k = K_1 \cup K_2$, $F = K_1 :: F_1$. By induction assumption, we have $A_1 \equiv \nu\tilde{c}.\,(K_2 :: a(X) \triangleright Q) \mid B$ for some $\tilde{c}, Q, B$, and $F_1 \equiv (X)\nu\tilde{c}.\,(K_2 \cup \{k\} :: Q) \mid B$. Thus, $A \equiv \nu\tilde{c}.\,(K_1 \cup K_2 :: a(X) \triangleright Q) \mid (K_1 :: B)$, and $F \equiv (X)\nu\tilde{c}.\,(K_1 \cup K_2 \cup \{k\} :: Q) \mid (K_1 :: B)$ as required.

- (Parl): in this case, $A = A_1 \mid A_2$, $A_1 \xrightarrow[K;k]{a} F_1$, $F = F_1 \mid A_2$. By induction assumption, we have $A_1 \equiv \nu\tilde{c}.\,(K :: a(X) \triangleright Q) \mid B$ for some $\tilde{c}, Q, B$, and $F_1 \equiv (X)\nu\tilde{c}.\,(K \cup \{k\} :: Q) \mid B$. Thus, $A \equiv \nu\tilde{c}.\,(K :: a(X) \triangleright Q) \mid (B \mid A_2)$, and $F \equiv (X)\nu\tilde{c}.\,(K \cup \{k\} :: Q) \mid (B \mid A_2)$ as required.

- (Parr): this case is handled as the (Parl) one.

- (Res): in this case $A = \nu e.\, A_1$, $e \neq a$, $A_1 \xrightarrow[K;k]{a} F_1$, $F = \nu e.\, F_1$. By induction assumption, we have $A_1 \equiv \nu\tilde{c}.\,(K :: a(X) \triangleright Q) \mid B$ for some $\tilde{c}, Q, B$, and $F_1 \equiv (X)\nu\tilde{c}.\,(K \cup \{k\} :: Q) \mid B$. Thus, $A \equiv \nu e, \tilde{c}.\,(K :: a(X) \triangleright Q) \mid B$ for some $\tilde{c}, Q, B$, and $F \equiv (X)\nu e, \tilde{c}.\,(K \cup \{k\} :: Q) \mid B$, as required.

We prove property 3 by induction on the derivation of $A \xrightarrow{\tau} A'$.

- (T-Cau): in this case, $A = K :: A_1$, $A_1 \xrightarrow{\tau} A_1'$ and $A' = K :: A_1'$. By induction assumption $A_1 \equiv \nu\tilde{c}.\, K_1 :: a\langle P\rangle \mid K_2 :: a(X) \triangleright Q \mid B$, $A_1' \equiv \nu\tilde{c}.\, K_1 \cup K_2 :: Q\{^P/_X\} \mid B$ for some $\tilde{c}, K_1, K_2, a, P, Q, B$. Thus

  $$A \equiv \nu\tilde{c}.\,(K \cup K_1 :: a\langle P\rangle) \mid (K \cup K_2 :: a(X) \triangleright Q) \mid (K :: B)$$

  and

  $$A' \equiv \nu\tilde{c}.\,(K \cup K_1 \cup K_2 :: Q\{^P/_X\}) \mid (K :: B)$$

  as required.

- (T-res), (T-parl), (T-parr): the proof is similar to the one of the (T-Cau) case.

- (Coml): in this case, we have $A = A_1 \mid A_2$, $A_1 \xrightarrow[K_1;k]{\overline{a}} C$, $A_2 \xrightarrow[K_2;k]{a} F$, and $A' = F\{^{K_1}/_k\} \bullet C$. Using property 1, we have $A_1 \equiv \nu\tilde{c}.\, K_1 :: a\langle P\rangle \mid B_1$ for some $\tilde{c}, P, B_1$, and $C = \nu\tilde{c}.\,\langle P\rangle \mid B_1$ Using property 2, we have $A_2 \equiv \nu\tilde{e}.\, K_2 :: a(X) \triangleright Q \mid B_2$ for some $\tilde{e}, Q, B_2$, and $F \equiv \nu\tilde{e}.\,(K_2 \cup \{k\} :: Q) \mid B_2$ Thus,

  $$A \equiv \nu\tilde{c}, \tilde{e}.\,(K_1 :: a\langle P\rangle) \mid (K_2 :: a(X) \triangleright Q) \mid (B_1 \mid B_2)$$

  and

  $$\begin{aligned} A' &\equiv \nu\tilde{c}, \tilde{e}.\,(((K_2 \cup \{k\} :: Q) \mid B_2)\{^P/_X\} \mid B_1)\{^{K_1}/_k\} \\ &\equiv \nu\tilde{c}, \tilde{e}.\,(K_2 \cup K_1 :: Q\{^P/_X\}) \mid (B_1 \mid B_2) \end{aligned}$$

as required (noting that X is not free in B_2 and that $k \notin \texttt{fk}(B_1, B_2)$ for $k \notin \texttt{fk}(A_1, A_2)$).

- (ComR): this case is proved like the (ComL) one.

□

We can finally prove the agreement proposition itself:

Proposition 2. *$A \xrightarrow{\tau}\equiv A'$ if and only if $A \to A'$.*

Proof. The *only if* part is a direct consequence of Lemma 49(3). For the *if* part, we reason by induction on the derivation of $A \to A'$:

- (C-red): in this case, $A = K_1 :: a\langle P\rangle \mid K_2 :: a(X) \triangleright Q$, and $A' = K_1 \cup K_2 : Q\{^P/_X\}$. We can apply (Out) followed by (Cau) to get $K_1 :: a\langle P\rangle \xrightarrow[K_1;k]{\overline{a}} \langle P\rangle K_1 :: \mathbf{0}$. We can apply (In) followed by (Cau) to get $K_2 :: a(X) \triangleright Q \xrightarrow[K_2;k]{a} (X)K_2 \cup \{k\} : Q$. Choosing k such that $k \notin K_1 \cup K_2$, and applying (ComL) we get:
$$A \xrightarrow{\tau} (K_2 \cup \{k\} : Q)\{^P/_X\}\{^{K_1}/_k\} = K_2 \cup K_1 :: Q\{^P/_X\}$$
- closure by $\equiv$: in this case, we have $A \equiv B$, $B \to B'$ and $B' \equiv A'$. By induction assumption, we have $B \xrightarrow{\tau}\equiv B'$. Now by Lemma 48, we have $A \xrightarrow{\tau} A''$ with $A'' \equiv B'$ and thus $A'' \equiv A'$, as required.
- closure by evaluation context: in this case we reason by induction on the form of evaluation context $\mathbb{E}$:
  - $\mathbb{E} = \bullet$: this is the case (C-red) above.
  - $\mathbb{E} = \nu a.\, \mathbb{E}'$: in this case $A = \nu a.\, B$ and $A' = \nu a.\, B'$ for some B, B' such that $B \to B'$. By induction assumption, $B \xrightarrow{\tau} B'' \equiv B'$. The thesis follows by applying rule T-res since $\equiv$ is a congruence.
  - $\mathbb{E} = \mathbb{E}' \mid B_P$ and symmetric: in this case $A = B \mid B_P$ and $A' = B' \mid B_P$ for some B, B' such that $B \to B'$. By induction assumption, $B \xrightarrow{\tau} B'' \equiv B'$. The thesis follows by applying rule T-parl since $\equiv$ is a congruence.

□

Appendix C. Proofs of Section 4

Appendix C.1. Proofs of Section 4.3

Before proving Lemma 13 below, which concerns configurations, we prove a corresponding result on processes.

Lemma 50. *For all names k and* rhoπ *processes P, Q, if $P \equiv Q$ then* $\mathtt{nf}(\llparenthesis P\rrparenthesis k) \equiv_{Ex} \mathtt{nf}(\llparenthesis Q\rrparenthesis k)$.

Proof. By induction on the derivation of $P \equiv Q$. The only interesting case is the base one, corresponding to the application of an axiom. We have a case analysis on the applied axiom. We consider just the most interesting cases.

$P \mid Q \equiv Q \mid P$**.** If either P or Q is congruent to $\mathbf{0}$, the thesis banally follows. Otherwise

$$\begin{aligned}
\mathtt{nf}(\llparenthesis P \mid Q\rrparenthesis k) &= \nu l, h.\, \mathtt{nf}(\llparenthesis P\rrparenthesis l) \mid \mathtt{nf}(\llparenthesis Q\rrparenthesis h) \mid \mathtt{nf}(\mathtt{KillP}\ l\ h\ k) \\
&\equiv_{Ex} \nu l, h.\, \mathtt{nf}(\llparenthesis P\rrparenthesis l) \mid \mathtt{nf}(\llparenthesis Q\rrparenthesis h) \mid \mathtt{nf}(\mathtt{KillP}\ h\ l\ k) \\
&= \mathtt{nf}(\llparenthesis Q \mid P\rrparenthesis k)
\end{aligned}$$

as desired, where we used axiom Ax.C.

$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$**.** If at least one among P, Q and R is equivalent to $\mathbf{0}$ the thesis banally follows. Otherwise

$$\begin{aligned}
&\mathtt{nf}(\llparenthesis P \mid (Q \mid R)\rrparenthesis k) = \\
&= \nu h, l, h', l'.\, \mathtt{nf}(\llparenthesis P\rrparenthesis h) \mid \mathtt{nf}(\llparenthesis Q\rrparenthesis h') \mid \mathtt{nf}(\llparenthesis R\rrparenthesis l') \mid \mathtt{nf}(\mathtt{KillP}\ h\ l\ k) \mid \\
&\quad\ \mathtt{nf}(\mathtt{KillP}\ h'\ l'\ l) \\
&=_\alpha \nu h, l, h', l'.\, \mathtt{nf}(\llparenthesis P\rrparenthesis h') \mid \mathtt{nf}(\llparenthesis Q\rrparenthesis l') \mid \mathtt{nf}(\llparenthesis R\rrparenthesis l) \mid \mathtt{nf}(\mathtt{KillP}\ h'\ h\ k) \mid \\
&\quad\ \mathtt{nf}(\mathtt{KillP}\ l'\ l\ h) \\
&\equiv_{Ex} \nu h, l, h', l'.\, \mathtt{nf}(\llparenthesis P\rrparenthesis h') \mid \mathtt{nf}(\llparenthesis Q\rrparenthesis l') \mid \mathtt{nf}(\llparenthesis R\rrparenthesis l) \mid \mathtt{nf}(\mathtt{KillP}\ h'\ l'\ h) \mid \\
&\quad\ \mathtt{nf}(\mathtt{KillP}\ l\ h\ k) \\
&= \mathtt{nf}(\llparenthesis (P \mid Q) \mid R\rrparenthesis k)
\end{aligned}$$

as desired, where we used axiom Ax.A. □

Lemma 13. *Let M, N be closed consistent configurations. Then $M \equiv N$ implies* $\mathtt{nf}(\llparenthesis M\rrparenthesis) \equiv_{Ex} \mathtt{nf}(\llparenthesis N\rrparenthesis)$.

PROOF. By induction on the derivation of $M \equiv N$. The only interesting case is the base one, corresponding to the application of an axiom. If the axiom is applied to a process only, the thesis follows from Lemma 50 and from the observation that processes are always applied to keys. We show below the most interesting of the other cases:

$(\nu u.\, M) \mid N \equiv \nu u.\, (M \mid N)$**.** By definition:

$$\begin{array}{rcl}
\mathtt{nf}(\llparenthesis(\nu u.\, M) \mid N\rrparenthesis) & = & \mathtt{nf}(\llparenthesis(\nu u.\, M)\rrparenthesis) \mid \mathtt{nf}(\llparenthesis N\rrparenthesis) \\
& = & \mathtt{nf}(\nu u.\, \llparenthesis M\rrparenthesis) \mid \mathtt{nf}(\llparenthesis N\rrparenthesis) \\
& = & \nu u.\, \mathtt{nf}(\llparenthesis M\rrparenthesis) \mid \mathtt{nf}(\llparenthesis N\rrparenthesis) \\
& \equiv & \mathtt{nf}(\nu u.\, (\llparenthesis M\rrparenthesis \mid \llparenthesis N\rrparenthesis)) \\
& = & \mathtt{nf}(\nu u.\, \llparenthesis M \mid N\rrparenthesis) \\
& = & \mathtt{nf}(\llparenthesis \nu u.\, (M \mid N)\rrparenthesis)
\end{array}$$

$\kappa : \nu a.\, P \equiv \nu a.\, \kappa : P$**.** We distinguish two cases, depending on the form of κ.

If $\kappa = \langle h_i, \tilde{h}\rangle \cdot k$ then by definition:

$$\begin{array}{rcl}
\mathtt{nf}(\llparenthesis\langle h_i, \tilde{h}\rangle \cdot k : \nu a.\, P\rrparenthesis) & = & \mathtt{nf}(\llparenthesis \nu a.\, P\rrparenthesis h_i) \mid \mathtt{nf}(\mathtt{Kill}_{\langle h_i, \tilde{h}\rangle \cdot k}) \\
& = & \nu a.\, \mathtt{nf}(\llparenthesis P\rrparenthesis h_i) \mid \mathtt{nf}(\mathtt{Kill}_{\langle h_i, \tilde{h}\rangle \cdot k}) \\
& \equiv & (\nu a.\, \mathtt{nf}(\llparenthesis P\rrparenthesis h_i) \mid \mathtt{nf}(\mathtt{Kill}_{\langle h_i, \tilde{h}\rangle \cdot k}) \\
& = & \nu a.\, \mathtt{nf}(\llparenthesis\langle h_i, \tilde{h}\rangle \cdot k : P\rrparenthesis) \\
& = & \mathtt{nf}(\llparenthesis \nu a.\, \langle h_i, \tilde{h}\rangle \cdot k : P\rrparenthesis)
\end{array}$$

The other case is simpler.

$k : \prod_{i=1}^{n} \tau_i \equiv \nu \tilde{h}.\ \prod_{i=1}^{n}(\langle h_i, \tilde{h}\rangle \cdot k : \tau_i)$. By definition:

$$\begin{aligned}
&\mathtt{nf}(\llparenthesis k : \prod_{i=1}^{n} \tau_i \rrparenthesis) = \mathtt{nf}(\llparenthesis \prod_{i=1}^{n} \tau_i \rrparenthesis k) = \\
&\nu h_1, l_1.\, \mathtt{nf}(\llparenthesis \tau_1 \rrparenthesis h_1) \mid \mathtt{nf}(\llparenthesis \prod_{i=2}^{n} \tau_i \rrparenthesis l_1) \mid \mathtt{nf}(\mathtt{KillP}\ h_1\ l_1\ k) = \\
&\nu h_1, h_2, l_1, l_2.\, \mathtt{nf}(\llparenthesis \tau_1 \rrparenthesis h_1) \mid \mathtt{nf}(\llparenthesis \tau_2 \rrparenthesis h_2) \mid \mathtt{nf}(\llparenthesis \prod_{i=2}^{n} \tau_i \rrparenthesis l_1) \mid \\
&\qquad \mathtt{nf}(\mathtt{KillP}\ h_1\ l_1\ k) \mid \mathtt{nf}(\mathtt{KillP}\ h_2\ l_2\ l_1) = \\
&\nu \tilde{h}, \tilde{l}.\, \prod_{i=1}^{n-1} \mathtt{nf}(\llparenthesis \tau_i \rrparenthesis h_i) \mid \mathtt{nf}(\llparenthesis \tau_n \rrparenthesis l_{n-1}) \mid \mathtt{nf}(\mathtt{KillP}\ h_1\ l_1\ k) \mid \\
&\quad \prod_{i=2}^{n-1} \mathtt{nf}(\mathtt{KillP}\ h_i\ l_i\ l_{i-1})
\end{aligned}$$

Now by α−converting the key of the last τ_n from l_{n-1} into h_n we obtain a term of the form

$$\begin{aligned}
&\nu \tilde{h}, \tilde{l}.\, \prod_{i=1}^{n} \mathtt{nf}(\llparenthesis \tau_i \rrparenthesis h_i) \mid \mathtt{nf}(\mathtt{KillP}\ h_1\ l_1\ k) \mid \prod_{i=2}^{n-2} \mathtt{nf}(\mathtt{KillP}\ h_i\ l_i\ l_{i-1}) \mid \\
&\qquad \mathtt{nf}(\mathtt{KillP}\ h_{n-1}\ h_n\ l_{i-2}) = \\
&\mathtt{nf}(\llparenthesis \nu \tilde{h}.\, \prod_{i=1}^{n} \langle h_i, \tilde{h}\rangle \cdot k : \tau_i \rrparenthesis)
\end{aligned}$$

Note that in this case we assumed that the $\mid$ is right associative, in order to unroll the parallel composition from $\prod_{i=1}^{n} \llparenthesis \tau_i \rrparenthesis$ to $\llparenthesis \tau_1 \rrparenthesis \mid \prod_{i=2}^{n} \llparenthesis \tau_i \rrparenthesis$. $\square$

Before proving Lemma 14 below, we show a few auxiliary results.

First, we characterize the effect of reduction to normal form on a term of the form $\mathbb{C}[P]$. To this end we define normal form on contexts, which also computes the substitution applied to the hole $\bullet$. To work with contexts with one hole only, we require that for all the higher-order applications $(X)P\ \mathbb{C}_1[\bullet]$ either P is linear or P is already in normal form. In the first case the bullet is not replicated, and single-hole contexts are enough. In the second case, no inductive call is required.

Definition 19 (Context normal form). The context normal form function $\mathtt{nfc}(\mathbb{C}[\bullet])$ is defined as $\mathtt{nfc}(\mathbb{C}[\bullet], \emptyset)$, where the second parameter is used for computing the substitution applied to the bullet. The result is also a pair $(\mathbb{C}'[\bullet], \sigma)$. The function $\mathtt{nfc}(\mathbb{C}[\bullet], \sigma)$ is defined as follows:

$\mathtt{nfc}(P \mid \mathbb{C}_1[\bullet], \sigma) = \mathtt{nf}(P) \mid \mathbb{C}'[\bullet], \sigma'$ if $\mathtt{nfc}(\mathbb{C}_1[\bullet], \sigma) = \mathbb{C}'[\bullet], \sigma'$

$\mathtt{nfc}(\nu a.\, \mathbb{C}_1[\bullet], \sigma) = \nu a.\, \mathbb{C}'[\bullet], \sigma'$ if $\mathtt{nfc}(\mathbb{C}_1[\bullet], \sigma) = \mathbb{C}'[\bullet], \sigma'$

$\mathtt{nfc}((X)\mathbb{C}_1[\bullet]\ P, \sigma) = \mathbb{C}'[\bullet], \sigma'$ if $\mathtt{nfc}(\mathbb{C}_1\{{}^{P}/_{X}\}[\bullet], \sigma \cdot \{{}^{P}/_{X}\}) = \mathbb{C}'[\bullet], \sigma'$

$\mathtt{nfc}((X)P\ \mathbb{C}_1[\bullet], \sigma) = \mathbb{C}'[\bullet], \sigma'$ if $\mathtt{nfc}(P\{{}^{\mathbb{C}_1[\bullet]}/_{X}\}, \sigma) = \mathbb{C}', \sigma'$ and P is linear

$\mathtt{nfc}((X)P\ \mathbb{C}_1[\bullet], \sigma) = P\{{}^{\mathbb{C}_1[\bullet]}/_{X}\}, \sigma \cdot \{{}^{\mathbb{C}_1[\bullet]}/_{X}\}$ if P is in normal form

$\mathtt{nfc}((h)\mathbb{C}_1[\bullet]\ l, \sigma) = \mathbb{C}'[\bullet], \sigma'$ if $\mathtt{nfc}(\mathbb{C}_1\{{}^{l}/_{h}\}[\bullet], \sigma \cdot \{{}^{l}/_{h}\}) = \mathbb{C}'[\bullet], \sigma'$

$\mathtt{nfc}(a\langle \mathbb{C}_1[\bullet]\rangle, \sigma) = a\langle \mathbb{C}_1[\bullet]\rangle, \sigma$

$\mathtt{nfc}(a(X) \triangleright \mathbb{C}_1[\bullet], \sigma) = a(X) \triangleright \mathbb{C}_1[\bullet], \sigma$

$\mathtt{nfc}(\bullet, \sigma) = \bullet, \sigma$

This definition enables the lemma below.

Lemma 51. *If* $\mathtt{nfc}(\mathbb{C}[\bullet], \emptyset) = \mathbb{C}_1, \sigma$ *then* $\mathtt{nfc}(\mathbb{C}[\bullet], \sigma') = \mathbb{C}_1, \sigma' \cdot \sigma$

The notions of normal form for processes and for contexts are compatible.

Lemma 52. $\mathtt{nf}(\mathbb{C}[P]) = \mathbb{C}'[\mathtt{nf}(P\sigma)]$ *with* $\mathtt{nfc}(\mathbb{C}[\bullet]) = \mathbb{C}'[\bullet], \sigma$ *if for each higher-order application* $(X)Q\ \mathbb{C}_1[\bullet]\ Q$ *is either linear or in normal form.*

Proof. By structural induction on $\mathbb{C}[\bullet]$. We show a few cases as examples:

- $\mathbb{C}[\bullet] = Q \mid \mathbb{C}_1[\bullet])$ we have that $\mathtt{nf}(Q \mid \mathbb{C}_1[P]) = \mathtt{nf}(Q) \mid \mathtt{nf}(\mathbb{C}_1[P])$. By inductive hypothesis we have that $\mathtt{nf}(\mathbb{C}_1[P]) = \mathbb{C}_1'[\mathtt{nf}(P\sigma)]$ with $\mathtt{nfc}(\mathbb{C}_1[P], \emptyset) = \mathbb{C}_1', \sigma$, and we can conclude by noting that $\mathtt{nfc}(Q \mid \mathbb{C}_1[\bullet], \emptyset) = \mathtt{nf}(Q) \mid \mathtt{nfc}(\mathbb{C}_1[\bullet], \emptyset)$.

- $\mathbb{C}[\bullet] = (X)\mathbb{C}_1[\bullet]\ Q)$ we have to show that $\mathtt{nf}((X)\mathbb{C}_1[P]\ Q) = \mathbb{C}'[\mathtt{nf}(P\sigma)]$ with $\mathtt{nfc}((X)\mathbb{C}_1[\bullet]\ Q, \emptyset) = \mathbb{C}', \sigma$. By definition, $\mathtt{nfc}((X)\mathbb{C}_1[\bullet]\ Q, \emptyset) = \mathtt{nfc}(\mathbb{C}_1[\bullet]\{{}^{Q}/_{X}\}, \{{}^{Q}/_{X}\})$. We have that $\mathtt{nf}(\mathbb{C}_1[P]\{{}^{Q}/_{X}\}) = \mathtt{nf}(\mathbb{C}_1\{{}^{Q}/_{X}\}[P\{{}^{Q}/_{X}\}]) = \mathbb{C}''[\mathtt{nf}(P\{{}^{Q}/_{X}\}\sigma')]$ where $\mathtt{nfc}(\mathbb{C}_1\{{}^{Q}/_{X}\}\ [\bullet], \emptyset) = \mathbb{C}'', \sigma'$ and by using Lemma 51 $\mathtt{nfc}(\mathbb{C}_1\{{}^{Q}/_{X}\}[\bullet], \{{}^{Q}/_{X}\}) = \mathbb{C}'', \{{}^{Q}/_{X}\} \cdot \sigma'$. We have that $\mathbb{C}'', \sigma'\{{}^{Q}/_{X}\} = \mathbb{C}', \sigma$. So $\mathtt{nf}((X)\mathbb{C}_1[P]\ Q) = \mathbb{C}'[\mathtt{nf}(P\sigma)]$ with $\mathtt{nfc}((X)\mathbb{C}_1[\bullet]\ Q, \emptyset) = \mathbb{C}', \sigma$, as desired.

- $\mathbb{C}[\bullet] = \texttt{nf}((X)Q\ \mathbb{C}_1[\bullet])$ with Q linear) we have that $\texttt{nf}((X)Q\ \mathbb{C}_1[P]) = \texttt{nf}(Q\{^{\mathbb{C}_1[P]}/_X\})$, but since Q is linear we can write $Q\{^{\mathbb{C}_1[P]}/_X\} = \mathbb{C}_2[P]$, hence $\texttt{nf}((X)Q\ \mathbb{C}_1[P]) = \texttt{nf}(Q\{^{\mathbb{C}_1[P]}/_X\}) = \texttt{nf}(\mathbb{C}_2[P])$. By inductive hypothesis we have that $\texttt{nf}(\mathbb{C}_2[P]) = \mathbb{C}_2'[P\sigma]$ with $\texttt{nfc}(\mathbb{C}_2[\bullet], \emptyset) = \mathbb{C}_2', \sigma$, as desired.

- $\mathbb{C}[\bullet] = \texttt{nf}((X)Q\ \mathbb{C}_1[\bullet])$ with Q in normal form) we have that $\texttt{nf}((X)Q\ \mathbb{C}_1[P]) = \texttt{nf}(Q\{^{\mathbb{C}_1[P]}/_X\}) = Q\{^{\mathbb{C}_1[P]}/_X\}$ and $\texttt{nfc}((X)Q\ \mathbb{C}_1[P], \emptyset) = \mathbb{C}_1'[\bullet], \sigma$ with $\texttt{nfc}(Q\{^{\mathbb{C}_1[\bullet]}/_X\}, \emptyset) = Q\{^{\mathbb{C}_1[\bullet]}/_X\}, \emptyset$ and $\mathbb{C}_1'[\bullet] = Q\{^{\mathbb{C}_1[\bullet]}/_X\}$, and we can conclude by stating that $\mathbb{C}_1'[P\emptyset] = Q\{^{\mathbb{C}_1[P]}/_X\}$, as desired. Note that if Q is in normal form then $\texttt{nf}(Q\{^P/_X\}) = Q\{^P/_X\}$, since the substitution will take place in non active contexts. □

Substitutions preserve structural congruence $\equiv_{Ex}$.

Lemma 53. *For any substitution $\sigma = \{^l/_h\}$ or $\sigma = \{^{(\!|P|\!)}/_X\}$, if $M \equiv_{Ex} N$ then $M\sigma \equiv_{Ex} N\sigma$.*

Proof. The proof is trivial for name substitutions. For higher-order ones the proof is by induction on the derivation of $M \equiv_{Ex} N$, with a case analysis on the last applied axiom of $\equiv_{Ex}$. All the cases are easy. □

Lemma 14. *If P and Q are well formed and $P \equiv_{Ex} Q$ then $\texttt{nf}(P) \equiv_{Ex} \texttt{nf}(Q)$.*

Proof. Let us consider P and Q generated by the encoding of Figure 7 where instead of having `Trig` processes, we substitute them with their normal form. Thus we can apply Lemma 52. Let us consider one application of an axiom. We have that $P = \mathbb{C}[L]$ and $Q = \mathbb{C}[R]$ with $L \equiv_{Ex} R$ an axiom. By Lemma 52 we have that $\texttt{nf}(\mathbb{C}[L]) = \mathbb{C}'[\texttt{nf}(L\sigma)]$ with $\texttt{nfc}(\mathbb{C}[\bullet], \emptyset) = \mathbb{C}'[\bullet], \sigma$, and the same with $\texttt{nf}(\mathbb{C}[R]) = \mathbb{C}'[\texttt{nf}(R\sigma)]$. By Lemma 53 we have that if $L \equiv_{Ex} R$ then $L\sigma \equiv_{Ex} R\sigma$. Also, $\texttt{nf}(L\sigma) \equiv_{Ex} \texttt{nf}(R\sigma)$ since $\equiv_{Ex}$ is closed under normal form. Finally, $\mathbb{C}'[\texttt{nf}(L\sigma)] \equiv_{Ex} \mathbb{C}'[\texttt{nf}(R\sigma)]$, as desired. □

Lemma 15. *If $\texttt{nf}(P) \equiv_{Ex} \texttt{nf}(P')$ then for each $Q \in \texttt{addG}(P)$ there exists $Q' \in \texttt{addG}(P')$ such that $\texttt{nf}(Q) \equiv_{Ex} \texttt{nf}(Q')$.*

Proof. By definition of `addG` we have that $P \equiv \mathbb{E}[\mathbf{0}]$ and $P' \equiv \mathbb{E}'[\mathbf{0}]$, with $Q \equiv \mathbb{E}[R]$. Let us choose $Q' \equiv \mathbb{E}'[R]$. By using Lemma 52 we have that

$\mathtt{nf}(\mathbb{E}[R]) = \mathbb{E}_1[\mathtt{nf}(R\sigma)]$ and $\mathtt{nf}(\mathbb{E}'[R]) = \mathbb{E}_2[\mathtt{nf}(R\sigma')]$. Since all the processes added by the function addG are closed we have that $R\sigma = R\sigma' = R$ and $\mathtt{nf}(\mathbb{E}[R]) = \mathbb{E}_1[\mathtt{nf}(R)]$ and $\mathtt{nf}(\mathbb{E}'[\mathtt{nf}(R)]) = \mathbb{E}_2[\mathtt{nf}(R)]$. By hypothesis we have that $\mathtt{nf}(\mathbb{E}[\mathbf{0}]) \equiv_{Ex} \mathtt{nf}(\mathbb{E}'[\mathbf{0}])$. We also have $\mathbb{E}_1[\mathtt{nf}(\mathbf{0}\sigma)] = \mathbb{E}_1[\mathbf{0}] \equiv_{Ex} \mathbb{E}_2[\mathbf{0}] = \mathbb{E}_2[\mathtt{nf}(\mathbf{0}\sigma')]$, and we can conclude by saying that also $\mathbb{E}_1[\mathtt{nf}(R)] \equiv_{Ex} \mathbb{E}_2[\mathtt{nf}(R)]$, that is $\mathtt{nf}(Q) \equiv_{Ex} \mathtt{nf}(Q')$, as desired. □

Lemma 16. *If P' is well formed and $P' \equiv \mathbb{C}[l\langle R\rangle]$ with $l \in \mathcal{K}$ for some n-ary context $\mathbb{C}$ then $P' \equiv \mathbb{C}'[l\langle R\rangle \mid S]$ with $S = \mathtt{Rew}\, l$ or $S = l(Z) \triangleright Z\, l$ for some n-ary context $\mathbb{C}'$.*

Proof. By definition of well formedness there exists a rhoπ process P such that $(\!|\nu k.\, k : P|\!) \Rightarrow P'$. The proof is by induction on the number of steps in $\Rightarrow$. The base case is when $(\!|\nu k.\, k : P|\!) = P'$. The proof is easy by inspection on the rules defining the encoding, since all messages on key channels are created together with the corresponding $\mathtt{Rew}\, l$.

In the inductive case we have that $P \Rightarrow P'' \to P'$ with $P' \equiv \mathbb{C}[l\langle R\rangle]$. For simplicity we consider just one instance of a message $l\langle R\rangle$ at the time. Note that messages cannot appear, but existing messages may be duplicated because of communications or applications. We proceed by case analysis on $P'' \to P'$. If the reduction does not involve the message $l\langle R\rangle$ nor the corresponding S the thesis follows easily. If the message $l\langle R\rangle$ is inside a communicated message or the argument of an application, then the corresponding S is inside the same message or the same argument and they are deleted, moved, or replicated together. If the message is read, it disappears and nothing has to be proved. If the message is inside a trigger or in the body of an abstraction on a process then a substitution may be applied to it, but this has no effect on the corresponding S, and the thesis holds by inductive hypothesis. If the reduction is an abstraction of the name l, then the same renaming is done on the corresponding $\mathtt{Rew}\, l$, and the thesis holds by inductive hypothesis. If the reduction is the application of $\mathtt{Rew}\, l$ we move from the first case of the thesis to the second one. If instead the reduction is the communication of $l(Z) \triangleright Z\, l$, this removes the corresponding message and nothing has to be proved (if it removes another message this means that there were two parallel messages on the same channel l, and we may simply exchange them in the correspondence). □

Lemma 17. *If P' is well formed and $P' \equiv \mathbb{C}[a\langle (\!|P|\!), l\rangle]$ with $a \in \mathcal{N}$ for* *some n-ary context* $\mathbb{C}$ *then* $P' \equiv \mathbb{C}'[a\langle (\!|P|\!), l\rangle \mid S]$ *with* $S = (\texttt{KillM}\ a\ l)$ *or* $S = (a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h\rangle \mid \texttt{Rew}\ l)$ *for some n-ary context* $\mathbb{C}'$.

Proof. By inspection of the encoding of Figure 7 we note that a message of the form $a\langle (\!|P|\!), l\rangle$ is only generated in the `Msg` process, together with the corresponding `KillM`. Also, interaction with a `KillM` is the only way to remove such a message. The case analysis is similar to the one in Lemma 16. □

Lemma 18. *Let* P *be a well formed* $HO\pi^+$ *process and* $Q \in \texttt{addG}(P)$. *If* $\texttt{nf}(Q) \hookrightarrow Q'$ *then there exists* P' *such that* $P \hookrightarrow^* P'$ *with* $Q' \in \texttt{addG}(P')$.

Proof. The reduction $\texttt{nf}(Q) \hookrightarrow Q'$ is due to a communication since processes in normal form have no enabled applications.

We distinguish two cases: either the reduction $\hookrightarrow$ is due to the process $\texttt{nf}(P)$ only or it involves garbage added by `addG`.

Let us consider the first case. By definition we have $Q = \mathbb{E}[R]$ with $P \equiv \mathbb{E}[\mathbf{0}]$ where R is the garbage added by the function `addG`. The form of Q is preserved by the $\texttt{nf}(\cdot)$ function, where all the applications are executed. Hence, $\texttt{nf}(Q) \equiv \mathbb{E}'[R']$ with $\texttt{nf}(P) = \mathbb{E}'[\mathbf{0}]$. Since the reduction does not involve R we have that $\mathbb{E}'[R] \hookrightarrow \mathbb{E}''[R] = Q'$, but also $\mathbb{E}'[\mathbf{0}] \hookrightarrow \mathbb{E}''[\mathbf{0}] = P'$. Moreover we have that $P \rightharpoondown^* \mathbb{E}'[\mathbf{0}] \hookrightarrow \mathbb{E}''[\mathbf{0}]$ and we are done.

In the second case, the only garbage processes that may interact with the context are either a $(\texttt{Rew}\ l)$ process or a $(\texttt{KillM}\ a\ l)$ process (the other garbage processes are inactive). If the administrative step is due to the applied form of $(\texttt{Rew}\ l)$ then by Lemma 16 the context $\mathbb{E}$ contains a message on the channel l, that is $\texttt{nf}(Q) = \mathbb{E}[l(Z) \triangleright Z\ l \mid R_1] \equiv \mathbb{E}_1[l\langle S\rangle \mid l(Z) \triangleright Z\ l \mid R_1] \hookrightarrow \mathbb{E}_1[(S\ l) \mid R_1] = Q'$. But since P is well formed and since the process $l(Z) \triangleright Z\ l$ has been added by the `addG` function, by Lemma 16 we have that also $P = \mathbb{E}_1[l\langle S\rangle] \equiv \mathbb{E}_2[l\langle S\rangle \mid l(Z) \triangleright Z\ l] \hookrightarrow \mathbb{E}_2[(S\ l)] = P'$. We can conclude by noting that $Q' \in \texttt{nf}(\texttt{addG}(P'))$. The other case is similar using Lemma 17 instead of Lemma 16. □

Appendix C.2. Proofs of Section 4.4

Lemma 23. *For each closed* $\texttt{rho}\pi$ *process* P, $(\!|P|\!)k \hookrightarrow^* \nu\tilde{u}.\, k\langle (\!|Q|\!)\rangle \mid \texttt{Rew}\ k \mid S$ *with* $k \notin \tilde{u}$, $S = \prod R_i$, $R_i = \texttt{Rew}\ k_i$ *or* $R_i = \nu\texttt{t}.\,(a(X, h)|\texttt{t}\triangleright R)$ *and* $P \equiv \nu\tilde{u}.\, Q$.

Proof. By induction on the structure of P. We show only the most interesting cases:

$P = a(X) \triangleright P'$ **:** be $Y = (X\ c)c\langle(\!|P'|\!)\rangle$, we have that

$$\begin{aligned}
&(\!|P|\!)k = ((l)(\texttt{Trig}\ Y\ a\ l))k \rightharpoondown \texttt{Trig}\ Y\ a\ k \rightharpoondown\\
&\nu\texttt{t}.\,(\overline{\texttt{t}} \mid a(X,h)|\texttt{t} \triangleright R \mid (\texttt{KillT}\ Y\ \texttt{t}\ k\ a)) \rightharpoondown\\
&\nu\texttt{t}.\,(\overline{\texttt{t}} \mid a(X,h)|\texttt{t} \triangleright R \mid \texttt{t} \triangleright k\langle(h)\texttt{Trig}\ Y\ a\ h\rangle \mid \texttt{Rew}\ k) \hookrightarrow\\
&\nu\texttt{t}.\,a(X,h)|\texttt{t} \triangleright R \mid k\langle(h)\texttt{Trig}\ Y\ a\ h\rangle \mid \texttt{Rew}\ k \equiv\\
&(\nu\texttt{t}.\,a(X,h)|\texttt{t} \triangleright R) \mid k\langle(h)\texttt{Trig}\ Y\ a\ h\rangle \mid \texttt{Rew}\ k = k\langle(h)\texttt{Trig}\ Y\ a\ h\rangle \mid S
\end{aligned}$$

as desired.

$P = \nu a.\,P'$ **:** we have that $(\!|\nu a.\,P'|\!)k = ((h)\nu a.\,(\!|P'|\!)h)k \rightharpoondown \nu a.\,(\!|P'|\!)k$. Now by inductive hypothesis we know that $(\!|P'|\!)k \hookrightarrow^* \nu\tilde{u}.\,k\langle(\!|Q|\!)\rangle \mid S$ with $P' \equiv \nu\tilde{u}.\,Q$ and since restriction is an execution context we have $\nu a.\,(\!|P'|\!)k \hookrightarrow^* \nu a.\,\nu\tilde{u}.\,k\langle(\!|Q|\!)\rangle \mid S$ with $\nu a.\,P' \equiv \nu a.\,\nu\tilde{u}.\,Q$, as desired.

$P = P_1 \mid P_2$ **:** we have that

$$\begin{aligned}
&(\!|P|\!)k = ((l)(\texttt{Par}\ (\!|P_1|\!)\ (\!|P_2|\!)\ l)k \rightharpoondown \texttt{Par}\ (\!|P_1|\!)\ (\!|P_2|\!)\ k \rightharpoondown\\
&\nu h,l.\,(\!|P_1|\!)h \mid (\!|P_2|\!)l \mid \texttt{KillP}\ l\ h\ k \rightharpoondown\\
&\nu h,l.\,(\!|P_1|\!)h \mid (\!|P_2|\!)l \mid (h(W)|l(Z) \triangleright k\langle(h)\texttt{Par}\ W\ Z\ h\rangle \mid \texttt{Rew}\ k).
\end{aligned}$$

By inductive hypothesis we have that $(\!|P_1|\!)h \hookrightarrow^* \nu\tilde{u}.\,h\langle(\!|P_1'|\!)\rangle \mid S_1$ with $P_1 \equiv \nu\tilde{u}.\,P_1'$ and $(\!|P_2|\!)l \hookrightarrow^* \nu\tilde{v}.\,l\langle(\!|P_2'|\!)\rangle \mid S_2$ with $P_2 \equiv \nu\tilde{v}.\,P_2'$. Hence we have

$$\begin{aligned}
&\nu h,l.\,(\!|P_1|\!)h \mid (\!|P_2|\!)l \mid h(W)|l(Z) \triangleright (k\langle(h)\texttt{Par}\ W\ Z\ h\rangle \mid \texttt{Rew}\ k) \hookrightarrow^*\\
&\nu h,l,\tilde{u},\tilde{v}.\,h\langle(\!|P_1'|\!)\rangle \mid S_1 \mid l\langle(\!|P_2'|\!)\rangle \mid S_2 \mid h(W)|l(Z) \triangleright k\langle(h)\texttt{Par}\ W\ Z\ h\rangle \mid\\
&\hspace{20em}\texttt{Rew}\ k \hookrightarrow\\
&\nu h,l,\tilde{u},\tilde{v}.\,S_1 \mid S_2 \mid k\langle(h)\texttt{Par}\ P_1'\ P_2'\ h\rangle \mid \texttt{Rew}\ k \equiv\\
&\nu h,l,\tilde{u},\tilde{v}.\,k\langle(h)\texttt{Par}\ (\!|P_1'|\!)\ (\!|P_2'|\!)\ h\rangle \mid S
\end{aligned}$$

with $S = \texttt{Rew}\ k \mid S_1 \mid S_2$, and by garbage collecting names h, l we have $P \equiv \nu\tilde{u},\tilde{v}.\,(P_1' \mid P_2')$ as desired. □

Lemma 24. *For any* rhoπ *process R, if $(\!|R|\!)l \Rightarrow P$ then the following conditions hold:*

1. $P \not\equiv \mathbb{C}[l_1\langle P_1\rangle \mid l_1\langle P_2\rangle]$, *with* $l_1 \in \mathcal{K}$.

2. $P \not\equiv \mathbb{C}[(\texttt{KillP}\ l_1\ l_2\ l_3) \mid (\texttt{KillP}\ l_4\ l_5\ l_6)]$, *with* $l_1, l_2, l_3, l_4, l_5, l_6 \in \mathcal{K}$ *and* $\{l_1, l_2\} \cap \{l_4, l_5\} \neq \emptyset$ *or* $l_3 = l_6$.
3. $P \not\equiv \mathbb{C}[(\texttt{KillP}\ l_1\ l_2\ l) \mid (\texttt{Mem}\ P\ a\ Q\ h\ l_3\ k)]$, *with* $l, l_1, l_2, l_3, h, k \in \mathcal{K}$ *and* $l_1 = l_3$ *or* $l_2 = l_3$.

Proof. By inspecting the encoding of Figure 7.

For condition 1 the proof is by structural induction on R, using as inductive hypothesis that messages are created only on the key channel passed to the process or on fresh key channels. In basic cases (**0** process, message or trigger), one message is created on the received channel, plus one if the `Rew` l is executed, but this consumes a message on the same channel thus preserving the invariant. For restriction, the thesis follows by inductive hypothesis. For `Par`, two distinct fresh names are passed to the parallel processes, thus by inductive hypothesis their messages will not be on the same channel as the one created by the `KillP`. For `Trig`, a fresh name is passed to the continuation, thus by inductive hypothesis its messages will not conflict with the one created by the `KillT` or by the one created by the `Trig` in the `Mem`. Note that these last two messages may not conflict since for them to be created the token `t` is needed, and either `KillT` or the `Trig` may read it, but not both of them.

For condition 2, note that `KillP` is generated only by a `Par` process. The first two arguments are fresh, thus they cannot be used by different `KillP`. For the last argument, we can reason as above to show that at most one `KillP` for each name may be created.

For condition 3 we can see that the channel used by the `Mem` is fresh, and the same for the ones used by `KillP`, thus they may not coincide. □

Before proving Lemma 25 below we show a few auxiliary results.

Lemma 54 (Input key invariant). *For each* rhoπ *process* R *and key* l *we have:*

1. $(\!|R|\!)l \not\Rightarrow \mathbb{E}[l(X) \triangleright P]$, *unless the trigger has been generated by an application of* `Rew` l.
2. $(\!|R|\!)l \not\Rightarrow \mathbb{E}[l(X) | l'(W) \triangleright P]$.
3. $(\!|R|\!)l \not\Rightarrow \mathbb{E}[l'(X) \triangleright P \mid l'(X) \triangleright Q]$ *for each* $l' \in \mathcal{K}$ *and* $l' \neq l$, *unless one of the triggers has been generated by an application of a* `Rew` l'.

Proof. All the cases are proved by induction on n, showing that no such derivation with less than n steps exists. All the base cases are trivial, since the starting term has no trigger in an active position. Let us consider the different inductive cases.

(1) We reason by contradiction. Suppose that $(\!|R|\!)l \Rightarrow \mathbb{E}[l(X) \triangleright P]$ in n steps. By looking at the encoding in Figure 7 we note that a trigger on a key channel can only occur inside a `Mem` term. However the subject k is generated fresh in the clause defining the `Trig` process, thus it cannot be l. This contradiction concludes the proof.

(2) Suppose towards a contradiction that $(\!|R|\!)l \Rightarrow \mathbb{E}[l(X)|l'(Z) \triangleright P]$ in n steps. By looking at the encoding we note that a trigger reading two messages can only be generated by a `KillP` process. However, the two names are generated fresh in the clause defining the `Par` process, against the hypothesis that one of them can be l.

(3) We use the same proof strategy as in (1) and (2). Triggers on keys $l' \in \mathcal{K}$ with $l \neq l$ may be generated only by `Mem` processes, but since the used key is fresh two triggers may not have the same key. $\square$

Messages on channels $l \in \mathcal{K}$ carry translations of processes with no toplevel restrictions.

Lemma 55. *For each* rhoπ *process R, $(\!|R|\!)l \Rightarrow \mathbb{C}[l\langle Q\rangle]$ implies $Q = (\!|Q'|\!)$ where Q' has no toplevel restriction.*

Proof. We proceed by induction on the number of reductions in $\Rightarrow$. The proof of the base case is by structural induction on R. We show a few cases as examples.

$R = a\langle P\rangle$ **:** by definition

$$\begin{aligned}&(\!|a\langle P\rangle|\!)h = (l)((a\ X\ l)\texttt{Msg}\ a\ X\ l)h =\\ &(l)((a\ X\ l)(a\langle X, \backslash l\rangle \mid (\texttt{KillM}\ a\ l))\ a\ (\!|P|\!)\ l)h =\\ &(l)(((a\ X\ l)(a\langle X, \backslash l\rangle \mid (((a\ l)a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h\rangle \mid\\ &(\texttt{Rew}\ l))\ a\ l))a\ (\!|P|\!)\ l)h\end{aligned}$$

as desired since $(h)\texttt{Msg}\ a\ X\ h = (\!|a\langle X\rangle|\!)$.

$R = \nu a.\, P$ **:** by definition $(\!|\nu a.\, P|\!)h = (l)(\nu a.\, (\!|P|\!))h$ and we can conclude by inductive hypothesis on P.

$R = P_1 \mid P_2$ **:** by definition

$$
\begin{aligned}
&(\!|P_1 \mid P_2|\!)h = (l)(\texttt{Par}\ (\!|P_1|\!)\ (\!|P_2|\!)\ l)h = \\
&(l)(((X\ Y\ l)\nu h, k.\, X\ h \mid Y\ k \mid (\texttt{KillP}\ h\ k\ l))\ (\!|P_1|\!)\ (\!|P_2|\!)\ l)h = \\
&(l)(((X\ Y\ l)\nu h, k.\, X\ h \mid Y\ k \mid \\
&\quad (((h\ k\ l)h(W)|k(Z) \triangleright l\langle(l)(\texttt{Par}\ W\ Z\ l)\rangle)\ h\ k\ l))\ (\!|P_1|\!)\ (\!|P_2|\!)\ l)h
\end{aligned}
$$

and we can conclude since $(l)(\texttt{Par}\ W\ Z\ l) = (\!|W \mid Z|\!)$.

For the inductive case, either the message we are considering disappears, and then there is nothing to prove, or the message remains in the context and we can conclude by applying the inductive hypothesis. Note, in fact, that higher order variables are always replaced by translations of $\texttt{rho}\pi$ processes, and that variables are never at top level inside the messages with subject l above. □

Lemma 25. *If* $\texttt{nf}(P_1) \equiv_{Ex} \texttt{nf}(P_2)$ *and* $\texttt{nf}(P_1) \to P_1'$ *then* $\texttt{nf}(P_2) \Rightarrow \texttt{nf}(P_2')$ *with* $\texttt{nf}(P_1'') \equiv_{Ex} \texttt{nf}(P_2')$ *and* $P_1'' \in \texttt{addG}(P_1')$. *Furthermore, if* $\to$ *is forward then* $\Rightarrow$ *is* $\Rightarrow_f$*, if* $\to$ *is backward then* $\Rightarrow$ *is* $\Rightarrow_b$*, if* $\to$ *is administrative then* $\Rightarrow$ *is* $\hookrightarrow^*$.

Proof. By case analysis on the used axiom $P \equiv_{Ex} Q$ and on the structure of $\texttt{nf}(P_1)$. Since we are dealing with processes in normal form, $\to$ is a communication. Hence we can express $\texttt{nf}(P_1)$ as an active context $\mathbb{E}[a\langle R\rangle \mid a(X) \triangleright S]$ (the cases where the trigger has different forms are similar). We will just consider a few interesting cases, the remaining ones are similar. The second part follows by noticing that in the proofs below the same trigger is used to mimic the reduction (possibly together with some auxiliary reductions).

$P \mid Q \equiv_{Ex} Q \mid P$. There are four places where the axiom could be applied: in the context, inside the message content, inside the trigger continuation, or to the context hole. We will consider the second case as an example. We have that $\texttt{nf}(P_1)$ can be written as $\mathbb{E}[a\langle\mathbb{C}[P \mid Q]\rangle \mid a(X) \triangleright S]$ and $\texttt{nf}(P_2)$ as $\mathbb{E}[a\langle\mathbb{C}[Q \mid P]\rangle \mid a(X) \triangleright S]$. We have that $\texttt{nf}(P_1) \to \mathbb{E}[S\{^{\mathbb{C}[P|Q]}/_X\}]$ and $\texttt{nf}(P_2) \to \mathbb{E}[S\{^{\mathbb{C}[Q|P]}/_X\}]$, with $\mathbb{E}[S\{^{\mathbb{C}[P|Q]}/_X\}] \equiv_{Ex} \mathbb{E}[S\{^{\mathbb{C}[Q|P]}/_X\}]$, and we can conclude by applying Lemma 14.

Ax.C. We have that $\texttt{nf}(P_1) = \mathbb{E}[\texttt{nf}(\texttt{KillP}\ l\ h\ k)] = \mathbb{E}[l(Z)|h(W) \triangleright k\langle(h)\texttt{Par}\ Z\ W\ l\rangle]$. If the communication is performed by the context $\mathbb{E}[\bullet]$

then the thesis banally follows. If the communication involves the hole, we should have in the context two messages of the form $l\langle P\rangle$ and $h\langle Q\rangle$, hence $\mathbb{E}[\texttt{nf}(\texttt{KillP}\ l\ h\ k)] \equiv \mathbb{E}'[l\langle P\rangle \mid h\langle Q\rangle \mid l(Z)|h(W) \triangleright k\langle (h)\texttt{Par}\ Z\ W\ l\rangle] \to \mathbb{E}'[k\langle (h)\texttt{Par}\ P\ Q\ l\rangle]$, and by expanding the definition of `Par` we have that $\texttt{nf}(P_1) \to \mathbb{E}'[k\langle (h)(\nu l_1, l_2.\, (P\ l_1) \mid (Q\ l_2) \mid \texttt{KillP}\ l_1\ l_2\ l)\rangle] \equiv_{Ex} \mathbb{E}'[k\langle (h)(\nu l_1, l_2.\, (Q\ l_1) \mid (P\ l_2) \mid \texttt{KillP}\ l_2\ l_1\ \ l)\rangle]$. Since $\texttt{nf}(P_2) \to \mathbb{E}'[k\langle (h)(\nu l_1, l_2.\, (Q\ l_1) \mid (P\ l_2) \mid \texttt{KillP}\ l_2\ l_1\ l)\rangle]$ the thesis follows.

Ax.P. We have $\texttt{nf}(P_1) = \mathbb{E}[l_1\langle (\!|P|\!)\rangle \mid l_2\langle (\!|Q|\!)\rangle \mid \texttt{nf}(\texttt{KillP}\ l_1\ l_2\ l)]$. The context can interact with the hole by reading messages on l_1 or l_2. By Lemma 54 only processes generated by a `Rew`, by a `KillP`, or by a `Mem` can read this kind of messages. Let us consider the `Rew` case. We have that $\texttt{nf}(P_1) \equiv \mathbb{E}'[l_1(Z) \triangleright Z\ l_1 \mid l_1\langle (\!|P|\!)\rangle \mid l_2\langle (\!|Q|\!)\rangle \mid \texttt{nf}(\texttt{KillP}\ l_1\ l_2\ l)] \to \mathbb{E}'[(\!|P|\!)l_1 \mid l_2\langle (\!|Q|\!)\rangle \mid \texttt{nf}(\texttt{KillP}\ l_1\ l_2\ l)]$. By using Lemma 55 and the axiom Ex.Unfold we have that $(\!|P|\!)l_1 \equiv_{Ex} \texttt{Rew}\ l_1 \mid l_1\langle (\!|P|\!)\rangle$, and then $\mathbb{E}'[(\!|P|\!)l_1 \mid l_2\langle (\!|Q|\!)\rangle \mid \texttt{nf}(\texttt{KillP}\ l_1\ l_2\ l)] \equiv_{Ex} \mathbb{E}'[\texttt{Rew}\ l_1 \mid l_1\langle (\!|P|\!)\rangle \mid l_2\langle (\!|Q|\!)\rangle \mid \texttt{nf}(\texttt{KillP}\ l_1\ l_2\ l)]$. So $\mathbb{E}[\texttt{Rew}\ l_1 \mid l_1\langle (\!|P|\!)\rangle \mid l_2\langle (\!|Q|\!)\rangle \mid \texttt{nf}(\texttt{KillP}\ l_1\ l_2\ l)] \equiv_{Ex} \texttt{nf}(\mathbb{E}[\texttt{Rew}\ l_1 \mid l\langle (h)\texttt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ h\rangle \mid \texttt{Rew}\ l])$, as desired. The case of a `KillP` corresponds to a reduction inside the hole, since no other message on the same channel can exist thanks to condition 1 of Lemma 24, and no other `KillP` may read the same messages thanks to condition 2 of Lemma 24. In this case we have that $\texttt{nf}(P_1) = \mathbb{E}[l_1\langle (\!|P|\!)\rangle \mid l_2\langle (\!|Q|\!)\rangle \mid (l_1(Z)|l_2(W) \triangleright (l)\texttt{Par}\ Z\ W\ h \mid \texttt{Rew}\ l)] \to \mathbb{E}[(l)\texttt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ h \mid \texttt{Rew}\ l]$ and the thesis banally follows. The case of a `Mem` can never happen thanks to condition 3 of Lemma 24.

Reductions from the right member of the axiom are trivially matched since the left member reduces to the right one.

Ax.A. We have that $\texttt{nf}(P_1) = \mathbb{E}[\nu l'.\, l_1(Z)|l_2(W) \triangleright l'\langle (h)\texttt{Par}\ Z\ W\ h \mid \texttt{Rew}\ l'\rangle \mid l'(Z')|l_3(W') \triangleright l\langle (h)\texttt{Par}\ Z'\ W'\ h \mid \texttt{Rew}\ l\rangle]$. The hole may perform a communication only if there are two messages on l_1, l_2 in the context. In this case, we have that $\texttt{nf}(P_1) \equiv \mathbb{E}'[\nu l'.\, l_1\langle P\rangle \mid l_2\langle Q\rangle \mid (l_1(Z)|l_2(W) \triangleright l'\langle (h)\texttt{Par}\ Z\ W\ h\rangle \mid \texttt{Rew}\ l') \mid \texttt{nf}(\texttt{KillP}\ l'\ l_3\ l)] \to \mathbb{E}'[l'\langle (h)\texttt{Par}\ P\ Q\ h\rangle \mid \texttt{Rew}\ l' \mid \texttt{nf}(\texttt{KillP}\ l'\ l_3\ l)] \equiv_{Ex} \mathbb{E}[\nu l'.\, l_1\langle P\rangle \mid l_2\langle Q\rangle \mid (l_1(Z)|l_2(W) \triangleright l'\langle (h)\texttt{Par}\ Z\ W\ h\rangle \mid \texttt{Rew}\ l') \mid \texttt{nf}(\texttt{KillP}\ l'\ l_3\ l)]$ using axiom Ax.P, as desired.

Ax.Unfold. Since $P \equiv \nu\tilde{u}.\, Q$ we have that $\texttt{nf}((\!|P|\!)l) \equiv \texttt{nf}((\!|\nu\tilde{u}.\, Q|\!)l) =$

$\nu\tilde{u}.\, \mathtt{nf}((\!|Q|\!)l)$, hence $\mathtt{nf}(P_1) = \mathbb{E}[\nu\tilde{u}.\, \mathtt{nf}((\!|Q|\!)l)]$. If $\mathbb{E}[\nu\tilde{u}.\, \mathtt{nf}((\!|Q|\!)l)] \to R$ then also $\mathtt{nf}(P_2) = \mathbb{E}[\nu\tilde{u}.\, l\langle(\!|Q|\!)\rangle \mid l(Z) \triangleright Z\ l] \to \mathbb{E}[\nu\tilde{u}.\, (\!|Q|\!)l] \rightharpoondown^* \mathbb{E}[\nu\tilde{u}.\, \mathtt{nf}((\!|Q|\!)l)] \to R$, as desired. Similarly, if $\mathtt{nf}(P_2) = \mathbb{E}[\nu\tilde{u}.\, l\langle(\!|Q|\!)\rangle \mid l(Z) \triangleright Z\ l] \to R'$ from Lemma 23 we have $\mathtt{nf}(P_1) = \mathbb{E}[\nu\tilde{u}.\, \mathtt{nf}((\!|Q|\!)l)] \hookrightarrow^* \mathbb{E}[\nu\tilde{u}.\, \mathtt{nf}(\nu\tilde{u'}.\, l\langle(\!|Q'|\!)\rangle \mid \mathtt{Rew}\ l \mid S)]$ where S is garbage. Thanks to Lemma 55 $\tilde{u'}$ is empty and $Q' = Q$, thus $\mathbb{E}[\nu\tilde{u}.\, \mathtt{nf}(\nu\tilde{u'}.\, l\langle(\!|Q'|\!)\rangle \mid \mathtt{Rew}\ l \mid S)] = \mathbb{E}[\nu\tilde{u}.\, l\langle(\!|Q|\!)\rangle \mid l(Z) \triangleright Z\ l \mid S] \to R'$ with $R' \in \mathtt{addG}(R)$ as desired. Note that this is the only case where garbage is generated.

Ax.Adm. By noting that the left member of the axiom can only perform a communication, reducing to the right member of the axiom. □

Before proving Lemma 28 below we show some auxiliary results. In particular, we have to study where in a configuration a key may occur. We call l-process, denoted by P_l, a process of a specific form.

Definition 20. Let $Y = (X\ c)c\langle(\!|Q|\!)\rangle$. An l-process is an HOπ^+process of one of the forms below, or obtained from them via one or more applications.

$(\!|P|\!)l$	$l\langle(\!|P|\!)\rangle \mid \mathtt{Rew}\ l$
$\mathtt{Msg}\ a\ (\!|Q|\!)\ l$	$\mathtt{Trig}\ Y\ a\ l$
$\mathtt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ l$	$\mathtt{KillP}\ h\ k\ l$
$\nu c.\, (Y\ (\!|P|\!)\ c) \mid (c(Z) \triangleright Z\ l) \mid (\mathtt{Mem}\ Y\ a\ (\!|P|\!)\ l_1\ l\ l_2)$	$\nu u.\, (\!|P|\!)l$
$\nu c.\, c\langle(\!|P|\!)\rangle \mid (c(Z) \triangleright Z\ l) \mid (\mathtt{Mem}\ Y\ a\ (\!|P|\!)\ l_1\ l\ l_2)$	$\mathbf{0}$
$\mathtt{Mem}\ Y\ a\ (\!|P|\!)\ h\ k\ l$	$\mathtt{Mem}\ Y\ a\ (\!|P|\!)\ l\ k\ h$

Essentially, each key l occurs at most twice (apart from occurrences in Rew), once in an l-process and possibly once in a killer process or a memory process.

We call primitive context a context originated during the translation of a process.

Definition 21. A context $\mathbb{C}$ is called a primitive context if it is generated by:

$$\mathbb{C} \ ::= \ \bullet \ \mid \ (l')\mathtt{Msg}\ a\ \mathbb{C}\ l' \ \mid \ (l')\mathtt{Trig}\ (X\ c)c\langle\mathbb{C}\rangle\ a\ l' \ \mid \ (l')\nu a.\, (\mathbb{C}\ l') \ \mid \ (l')\mathtt{Par}\ \mathbb{C}\ (\!|Q|\!)\ l' \ \mid \ (l')\mathtt{Par}\ (\!|P|\!)\ \mathbb{C}\ l'$$

Lemma 56. *For any* $\mathtt{rho}\pi$ *process* P *and any* $l \in \mathtt{n}(\llparenthesis P \rrparenthesis) \cap \mathcal{K}$*, we have* $\llparenthesis P \rrparenthesis = \mathbb{C}[(l)P_l]$ *for some primitive context* $\mathbb{C}$ *and some* l*-process* P_l*.*

Proof. By structural induction on P. □

All the messages on channels $a \in \mathcal{N}$ carry a pair whose first element is the translation of a process.

Lemma 57. *For any* $\mathtt{rho}\pi$ *process* P*, if* $\llparenthesis P \rrparenthesis k \Rightarrow \mathbb{E}[a\langle Q, h\rangle]$ *with* $a \in \mathcal{N}$ *then* $Q = \llparenthesis R \rrparenthesis$*.*

Proof. By induction on the number of steps in $\Rightarrow$. The base case is proved by inspection. The inductive case is easy. □

We can now prove the invariant on the use of keys.

Lemma 58. *For any configuration* $\nu k.\, k : P$ *if* $\llparenthesis k : P \rrparenthesis \Rightarrow R$ *then for any key* l *occurring in* R *one of the following statements holds:*

1. $R \equiv \mathbb{E}[\nu l.\, R']$*, with* $R' \in \mathtt{addG}(P_l \mid S)$ *where* P_l *is an* l*-process and* S *is obtained from one of the following terms via* 0 *or more applications:* $(\mathtt{KillP}\, l\, l'\, h)$, $(\mathtt{KillP}\, l'\, l\, h)$, $(\mathtt{Mem}\, Y\, a\, \llparenthesis P \rrparenthesis\, l_1\, l\, l_2)$, $\mathbf{0}$.
2. $R \equiv \mathbb{C}[\nu h.\, \mathbb{C}'[(l)P_l]h]$ *where* P_l *is an* l*-process and* $\mathbb{C}'$ *is generated from a primitive context via* 0 *or more applications.*
3. $R \equiv \mathbb{C}[\nu l.\, ((h)P_h)l]$ *where* P_h *is an* h*-process.*
4. $R \equiv \mathbb{C}[\nu l, c.\, (Y\ \llparenthesis Q \rrparenthesis\ c) \mid (c(Z) \triangleright Z\ l) \mid (\mathtt{Mem}\ Y\ a\ \llparenthesis Q \rrparenthesis\ h\ l\ k)]$ *where* $Y = ((X\ c)c\langle \llparenthesis P \rrparenthesis \rangle)$.

Proof. By induction on the number of steps in $\llparenthesis k : P \rrparenthesis \Rightarrow R$. For the base case ($n = 0$) we have that $\llparenthesis \nu k.\, k : P \rrparenthesis = \nu k.\, \llparenthesis P \rrparenthesis k$. By Lemma 56 we have that for any $l \in \mathtt{n}(\llparenthesis P \rrparenthesis) \cap \mathcal{K}$ we have $\llparenthesis P \rrparenthesis = \mathbb{C}'[(l)P_l]$, that is for any l we can write $\nu k.\, \llparenthesis P \rrparenthesis k \equiv \nu k.\, \mathbb{C}'[(l)P_l]k$, and condition (2) holds.

In the inductive case we distinguish two possibilities: either name l did not exist at the previous step, or it existed. By inspection of the encoding one can see that the first case may only happen when recursive definitions for `Par` or `Trig` are unfolded. In both the cases condition (1) is satisfied for new names, with $P_l = \llparenthesis P \rrparenthesis l$ in the first case and $P_l = \nu c.\, c\langle \llparenthesis P \rrparenthesis \rangle \mid (c(Z) \triangleright Z\, l) \mid (\mathtt{Mem}\, Y\, a\, \llparenthesis P \rrparenthesis\, l_1\, l\, l_2)$ in the second case.

For the second possibility we have a case analysis according to which condition holds before the additional step is done.

Let us consider the condition (1). Reductions involving only $\mathbb{E}$ leave the process in the same form. For other reductions we proceed by case analysis on the form of P_l (we consider applied forms together with the form they derive from). For simplicity we do not write `addG` if it does not change. Adding it is however straightforward.

If $P_l = (\!|P|\!)l$ we proceed by case analysis on the structure of P. If $P = \mathbf{0}$ then we have $\mathbb{E}[\nu l.\, P_l \mid S] \rightharpoondown \mathbb{E}[\nu l.\, l\langle \mathtt{Nil}\rangle \mid \mathtt{Rew}\ l \mid S]$, which satisfies again condition (1). Similarly, if $P = a\langle Q\rangle$ then $\mathbb{E}[\nu l.\, (\!|a\langle Q\rangle|\!) \mid S] \rightharpoondown \mathbb{E}[\nu l.\, \mathtt{Msg}\ a\ (\!|Q|\!)\ l]$, which satisfies again condition (1). The other cases are similar.

If $P_l = l\langle (\!|P|\!)\rangle \mid \mathtt{Rew}\ l$ then $\mathbb{E}[\nu l.\, l\langle (\!|P|\!)\rangle \mid \mathtt{Rew}\ l \mid S]$ can perform several reductions. If $\mathtt{Rew}\ l$ is applied, then we stay in the same case. If $\mathtt{Rew}\ l$ is already in its applied form then $\mathbb{E}[\nu l.\, l\langle (\!|P|\!)\rangle \mid l(Z) \triangleright Z\ l \mid S] \hookrightarrow \mathbb{E}[\nu l.\, (\!|P|\!)l \mid S]$, which again satisfies the conditions. If the message on l is read by S then there are two cases: either it is read by a memory, or by a `KillP`. In the first case we have that:

$$
\begin{aligned}
&\mathbb{E}[\nu l.\, l\langle (\!|P|\!)\rangle \mid \mathtt{Rew}\ l \mid (l(Z) \triangleright \mathtt{Msg}\ a\ (\!|Q|\!)\ l_1 \mid \mathtt{Trig}(X\ c)c\langle (\!|R|\!)\rangle\ a\ l_2) \mid S] \hookrightarrow \\
&\mathbb{E}[\nu l.\, \mathtt{Rew}\ l \mid (\mathtt{Msg}\ a\ (\!|Q|\!)\ l_1) \mid (\mathtt{Trig}(X\ c)c\langle (\!|R|\!)\rangle\ a\ l_2) \mid S] \equiv \\
&\mathbb{E}'[\nu l.\, \mathbf{0} \mid \mathtt{Rew}\ l \mid S]
\end{aligned}
$$

which satisfies again condition (1) since $\nu l.\, \mathbf{0} \mid \mathtt{Rew}\ l \in \mathtt{addG}(\nu l.\, \mathbf{0})$. If the message is read by a `KillP` then the context contains also a message on channel h such that:

$$
\begin{aligned}
&\mathbb{E}[\nu l.\, l\langle (\!|P|\!)\rangle \mid \mathtt{Rew}\ l \mid S] \equiv \\
&\mathbb{E}_1[\nu l.\, h\langle (\!|Q|\!)\rangle \mid l\langle (\!|P|\!)\rangle \mid (l(Z)|h(W) \triangleright k\langle (h)\mathtt{Par}\ Z\ W\ h\rangle \mid \mathtt{Rew}\ k) \mid S] \hookrightarrow \\
&\mathbb{E}_1[(\nu l.\, \mathbf{0} \mid S) \mid k\langle (h)\mathtt{Par}\ (\!|Q|\!)\ (\!|P|\!)\ h\rangle \mid \mathtt{Rew}\ k] \equiv \mathbb{E}_2[\nu l.\, \mathbf{0} \mid S]
\end{aligned}
$$

which satisfies again condition (1).

If $P_l = \mathtt{Msg}\ a\ (\!|Q|\!)\ l$ or $P_l = \mathtt{Trig}\ Y\ a\ l$ we have a few cases. If P_l reduces alone then it is simply applied, and its applied form is still an l-process. Note that neither S nor the context can interact with such a P_l before application.

Let us consider applied forms of $P_l = \mathtt{Msg}\ a\ (\!|Q|\!)\ l$. With one application we get $P_l = a\langle (\!|Q|\!), l\rangle \mid \mathtt{KillM}\ a\ l$. In this case $\mathbb{E}[\nu l.\, P_l \mid S]$ can perform several reductions. If `KillM` is not in its applied form and it is applied, then the thesis banally follows. If `KillM` is in its applied form, then it can interact with the message. In this case, we have that

$$\mathbb{E}[\nu l.\, a\langle (\!|Q|\!), l\rangle \mid a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ (\!|Q|\!)\ h\rangle \mid \texttt{Rew}\ l \mid S] \hookrightarrow$$
$$\mathbb{E}[\nu l.\, l\langle (h)\texttt{Msg}\ a\ (\!|Q|\!)\ h\rangle \mid (\texttt{Rew}\ l) \mid S]$$

and condition (1) still holds. If the message is read by the context, then it is read either by a KillM (in its applied form) or by a Trig (in its applied form). The first case has been treated just above. In the second case (we only need to consider the trigger, and the token $\overline{\texttt{t}}$ needed for its activation) we have that

$$\mathbb{E}[\nu l.\, a\langle (\!|Q|\!), l\rangle \mid (\texttt{KillM}\ a\ l) \mid S] \equiv$$
$$\mathbb{E}_1[\nu l.\, a\langle (\!|Q|\!), l\rangle \mid \overline{\texttt{t}} \mid (a(X,h)|\texttt{t} \triangleright \nu k, c.\, (((X\ c)c\langle (\!|P|\!)\rangle)\ X\ c) \mid (c(Z) \triangleright Z\ k) \mid$$
$$(\texttt{Mem}\ Y\ a\ X\ \ h\ k\ l)) \mid (\texttt{KillM}\ a\ l)| \mid S] \hookrightarrow$$
$$\mathbb{E}_1[\nu l, k, c.\, (((X\ c)c\langle (\!|P|\!)\rangle)\ (\!|Q|\!)\ c) \mid c(Z) \triangleright Z\ k \mid (\texttt{Mem}\ Y\ a\ (\!|P|\!)\ l\ k\ l_1) \mid S] \equiv$$
$$\mathbb{E}_2[\nu l.\, (\texttt{Mem}\ Y\ a\ (\!|P|\!)\ l\ k\ l_1) \mid S]$$

and condition (1) still holds.

Let us consider applied forms of $P_l = \texttt{Trig}\ Y\ a\ l$, i.e., $P_l = \nu\texttt{t}.\,\overline{\texttt{t}} \mid (a(X,h)|\texttt{t} \triangleright R) \mid (\texttt{KillT}\ Y\ \texttt{t}\ l\ a)$ where $R = \nu k, c.\, (Y\ X\ c) \mid (c(Z) \triangleright (Z\ k)) \mid (\texttt{Mem}\ Y\ a\ X\ h\ k\ l)$ and $Y = (X\ c)c\langle (\!|Q|\!)\rangle$. In this case $\mathbb{E}[\nu l.\, P_l \mid S]$ can perform several reductions. If the KillT is applied then the thesis banally follows. If the KillT is in its applied form then it can interact with $\overline{\texttt{t}}$:

$$\mathbb{E}[\nu l, \texttt{t}.\,\overline{\texttt{t}} \mid (a(X,h)|\texttt{t} \triangleright_f R) \mid (\texttt{t} \triangleright l\langle (h)\texttt{Trig}\ Y\ a\ h\rangle) \mid \texttt{Rew}\ l) \mid S] \hookrightarrow$$
$$\mathbb{E}[\nu l, \texttt{t}.\, (a(X,h)|\texttt{t} \triangleright_f R) \mid l\langle (h)\texttt{Trig}\ Y\ a\ h\rangle \mid \texttt{Rew}\ l \mid S] \equiv$$
$$\mathbb{E}_1[\nu l.\, l\langle (h)\texttt{Trig}\ Y\ a\ h\rangle \mid \texttt{Rew}\ l \mid S]$$

as desired since $l\langle (h)\texttt{Trig}\ Y\ a\ h\rangle \mid \texttt{Rew}\ l \in \texttt{addG}((\!|a(X) \triangleright Q|\!))$. The only other possibility is that the trigger reads a message from the context. Thanks to Lemma 57 the message should be of the form $a\langle (\!|P|\!), l_1\rangle$ for some P and some l_1. Therefore,

$$\mathbb{E}[\nu l, \texttt{t}.\,\overline{\texttt{t}} \mid (a(X,h)|\texttt{t} \triangleright_f R) \mid \texttt{KillT}\ Y\ \texttt{t}\ l\ a \mid S] \equiv$$
$$\mathbb{E}_1[\nu l, \texttt{t}.\,\overline{\texttt{t}} \mid a\langle (\!|P|\!), l_1\rangle \mid (a(X,h)|\texttt{t} \triangleright_f R) \mid \texttt{KillT}\ Y\ \texttt{t}\ l\ a \mid S] \twoheadrightarrow$$
$$\mathbb{E}_1[\nu l, k, \texttt{t}, c.\, (((X\ c)c\langle (\!|Q|\!)\rangle)\ (\!|P|\!)\ c) \mid c(Z) \triangleright Z\ k \mid (\texttt{Mem}\ Y\ a\ (\!|P|\!)\ l_1\ k\ l) \mid S] \equiv$$
$$\mathbb{E}_2[\nu k.\, (((X\ c)c\langle (\!|Q|\!)\rangle)\ (\!|P|\!)\ c) \mid c(Z) \triangleright Z\ k \mid \nu l.\, (\texttt{Mem}\ Y\ a\ (\!|P|\!)\ l_1\ k\ l) \mid S] \equiv$$
$$\mathbb{E}_3[\nu l.\, (\texttt{Mem}\ Y\ a\ (\!|P|\!)\ l_1\ k\ l) \mid S]$$

where condition (1) holds for name l. Note that condition (1) holds also for the new name k.

If $P_l = \mathtt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ l$ then we have that $\mathbb{E}[\nu l.\,(\mathtt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ l) \mid S] \hookrightarrow \mathbb{E}[\nu l, h, k.\, (\!|P|\!)h \mid (\!|Q|\!)k \mid (\mathtt{KillP}\ h\ k\ l)]$, and we have that condition (1) is still satisfied by name l and also by the new names h and k.

If $P_l = \mathtt{KillP}\ h\ k\ l$ and it reduces alone then it is applied, and condition (1) still holds. P_l may interact with the context only if it is in its applied form and there are two messages, one on h and one on k. Thanks to Lemma 55 they should contain translations of processes, thus:

$$\begin{aligned}
&\mathbb{E}[\nu l.\, h(W)|k(Z) \triangleright l\langle (h)\mathtt{Par}\ W\ Z\ h\rangle] \equiv\\
&\mathbb{E}_1[\nu l.\, h(W)|k(Z) \triangleright l\langle (h)\mathtt{Par}\ W\ Z\ h\rangle \mid h\langle (\!|P|\!)\rangle \mid k\langle (\!|Q|\!)\rangle] \hookrightarrow\\
&\mathbb{E}[\nu l.\, l\langle (h)\mathtt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ h\rangle \mid \mathtt{Rew}\ l]
\end{aligned}$$

and condition (1) still holds.

If $P_l = \nu u.\, (\!|P|\!)l$ then we can move the restriction into the context and reduce to the case $P_l = (\!|P|\!)l$.

If $P_l = \nu c.\,(Y\ \ (\!|P|\!)\ \ c) \mid (c(Z) \triangleright Z\ \ l) \mid (\mathtt{Mem}\ Y\ \ a\ \ (\!|P|\!)\ \ l_1\ \ l\ \ l_2)]$ then $\mathbb{E}[\nu l.\,(P_l \mid S)]$ can perform several reductions. If the $\mathtt{Mem}$ process is applied then condition (1) is still satisfied. Since c and l are restricted, the only other possible reduction is the application of $Y = (X\ c)c\langle (\!|Q|\!)\rangle$. Thus:

$$\begin{aligned}
&\mathbb{E}[\nu l, c.\,(Y\ (\!|P|\!)\ c) \mid (c(Z) \triangleright Z\ l) \mid (\mathtt{Mem}\ Y\ a\ (\!|P|\!)\ l_1\ l\ l_2)] \rightarrowtail\\
&\mathbb{E}[\nu l, c.\, c\langle (\!|Q|\!)\{^{(\!|P|\!)}/_X\}\rangle \mid (c(Z) \triangleright Z\ l) \mid (\mathtt{Mem}\ Y\ a\ (\!|P|\!)\ l_1\ l\ l_2)]
\end{aligned}$$

and condition (1) still holds. The case where the memory is in its applied form is analogous.

If $P_l = \nu c.\, c\langle (\!|P|\!)\rangle \mid (c(Z) \triangleright Z\ l) \mid (\mathtt{Mem}\ Y\ a\ (\!|P|\!)\ l_1\ l\ l_2)$ and the $\mathtt{Mem}$ process is applied then condition (1) holds. Since c and l are restricted, the only possible communication is the internal one along c:

$$\begin{aligned}
&\mathbb{E}[\nu l, c.\, c\langle (\!|P|\!)\rangle \mid (c(Z) \triangleright Z\ l) \mid (\mathtt{Mem}\ Y\ a\ (\!|P|\!)\ l_1\ l\ l_2)] \hookrightarrow\\
&\mathbb{E}[\nu l, c.\, (\!|P|\!)l \mid (\mathtt{Mem}\ Y\ a\ (\!|P|\!)\ l_1\ l\ l_2)] \equiv\\
&\mathbb{E}[\nu l.\,(P_l \mid S)]
\end{aligned}$$

where condition (1) still holds. The case where the memory is in its applied form is analogous.

If $P_l = \texttt{Mem}\, Y\ a\ (\!|P|\!)\ l\ k\ h$ and the Mem is applied, then the thesis banally follows. If the Mem process is already in its applied form, then it may only interact with the context via a message on k. Hence, we have

$$\begin{aligned}&\mathbb{E}[\nu l.\, k(Z) \triangleright_b (\texttt{Msg}\ a\ (\!|P|\!)\ l) \mid (\texttt{Trig}\, Y\ a\ h)] \equiv\\ &\mathbb{E}_1[\nu l.\, k\langle R\rangle \mid k(Z) \triangleright_b (\texttt{Msg}\ a\ (\!|P|\!)\ l) \mid (\texttt{Trig}\, Y\ a\ h)] \rightsquigarrow\\ &\mathbb{E}_1[(\nu l.\, (\texttt{Msg}\ a\ (\!|P|\!)\ l)) \mid \texttt{Trig}\, Y\ a\ h] \equiv\\ &\mathbb{E}_2[\nu l.\, (\texttt{Msg}\ a\ (\!|P|\!)\ l))]\end{aligned}$$

and condition (1) still holds.

The case $P_l = (\texttt{Mem}\, Y\ a\ (\!|P|\!)\ h\ k\ l)$ is similar.

Since $\mathbf{0}$ has no reduction, in the case $P_l = \mathbf{0}$ there is nothing to prove.

Let us consider conditions (2), (3) and (4). If the context evolves by itself, then they are still satisfied. For the hole to contribute, the only possibility is that the context is an active context. For condition (2), we have a case analysis on the form of $\mathbb{C}'$. If $\mathbb{C}' = \bullet$, then $\mathbb{C}[\nu h.\, ((l)P_l)h] \twoheadrightarrow \mathbb{C}[\nu h.\, P_h]$, the name l disappears and the thesis trivially holds. If $\mathbb{C}' = (l')\texttt{Msg}\ a\ \mathbb{C}''[\bullet]\ l'$ then we have $\mathbb{C}[\nu h.\, ((l')\texttt{Msg}\ a\ \mathbb{C}''[(l)P_l]\ l')h] \twoheadrightarrow \mathbb{C}[\nu h.\, \texttt{Msg}\ a\ \mathbb{C}''[(l)P_l]\ h]$. Name l disappears, thus the thesis holds trivially. Note that condition (2) holds for name h. The other cases for contexts in non applied form are analogous.

Let us now consider contexts where the toplevel application has been performed. Again we have a case analysis on the form of the context. If the context has the form $\mathbb{C}[\nu h.\, \texttt{Msg}\ a\ \mathbb{C}'[(l)P_l]\ h]$, then we have that $\mathbb{C}[\nu h.\, \texttt{Msg}\ a\ \mathbb{C}'[(l)P_l]\ h] \twoheadrightarrow \mathbb{C}[\nu h.\, a\langle \mathbb{C}'[(l)P_l], h\rangle \mid \texttt{KillM}\ a\ h]$ and condition (2) still holds. The other cases are similar.

If condition (3) holds, we have that $\mathbb{C}[\nu l.\, ((h)P_h)l] \twoheadrightarrow \mathbb{C}[\nu l.\, P_l]$, and condition (1) holds for name l.

If condition (4) holds, we have that:

$$\begin{aligned}&\mathbb{C}[\nu l, c.\, (((X\ c)c\langle (\!|P|\!)\rangle)\ (\!|Q|\!)\ c) \mid (c(Z) \triangleright Z\ l) \mid (\texttt{Mem}\ (X\ c)c\langle (\!|P|\!)\rangle)\ a\ (\!|Q|\!)\ h\ l\ k)] \twoheadrightarrow\\ &\mathbb{C}[\nu l, c.\, c\langle (\!|P|\!)\rangle\{^{(\!|Q|\!)}/_X\}] \mid (c(Z) \triangleright Z\ l) \mid (\texttt{Mem}\ (X\ c)c\langle (\!|P|\!)\rangle)\ a\ (\!|Q|\!)\ h\ l\ k)] =\\ &\mathbb{C}[\nu l, c.\, c\langle (\!|P|\!)\{^Q/_X\}\rangle] \mid (c(Z) \triangleright Z\ l) \mid (\texttt{Mem}\ (X\ c)c\langle (\!|P|\!)\rangle)\ a\ (\!|Q|\!)\ h\ l\ k)]\end{aligned}$$

by applying Lemma 21. Condition (1) holds for the name l. If the reduction involves the application of the Mem process, then condition (1) for name l is still satisfied. □

Lemma 28. *For any consistent configuration M, if $(\!|M|\!) \Rightarrow P$ and $P \hookrightarrow^* Q$ then there exist Q' and P' such that $Q \hookrightarrow^* Q'$, $P' \in \texttt{addG}(P)$ with $\texttt{nf}(P') \equiv_{Ex} \texttt{nf}(Q')$.*

Proof. By induction on the number n of steps in $P \hookrightarrow^* Q$. In the base case ($n = 0$) the thesis banally follows. For the inductive case, consider the first step $P \hookrightarrow Q_1$ of $P \hookrightarrow^* Q$. There are two cases to distinguish: whether $\hookrightarrow$ is an application $\rightharpoondown$ or a non labelled communication $\rightarrow$. Let us consider the first case. We have that $P \rightharpoondown Q_1$ and $Q_1 \hookrightarrow^* Q$. By inductive hypothesis there exist Q', Q'_1 such that $Q \hookrightarrow^* Q'$, $Q'_1 \in \texttt{addG}(Q_1)$ and $\texttt{nf}(Q'_1) \equiv_{Ex} \texttt{nf}(Q')$. Since the added garbage does not forbid reductions we have that there exists $P' \in \texttt{addG}(P)$ such that $P' \rightharpoondown Q'_1$. Thus $\texttt{nf}(P') = \texttt{nf}(Q'_1) \equiv_{Ex} \texttt{nf}(Q')$ as desired.

If the step is a non labelled communication $\rightarrow$ then we have three cases, corresponding to the three kinds of non labelled trigger processes: a Rew process, a killer process and the trigger in the translation of the continuation of a $\mathsf{rho}\pi$ trigger.

Let us consider the case of a Rew process. We have that $P \equiv \mathbb{E}[l\langle (\!|P_1|\!) \rangle \mid l(Z) \triangleright Z\ l] \rightarrow \mathbb{E}[(\!|P_1|\!) l]$. By using Lemma 23 we have that $(\!|P_1|\!) l \hookrightarrow^* l\langle (\!|P_2|\!) \rangle \mid \texttt{Rew}\ l \mid S$ with S a parallel composition of garbage processes. Thanks to Lemma 55, P_1 has no toplevel restrictions (since it was argument of a message), thus $P_2 = P_1$. So, we have that $\mathbb{E}[(\!|P_1|\!) l] \hookrightarrow^* \mathbb{E}[l\langle (\!|P_1|\!) \rangle \mid \texttt{Rew}\ l \mid S] \in \texttt{addG}(P)$. Now from $\mathbb{E}[(\!|P_1|\!) l] \hookrightarrow^* Q$ we have that by inductive hypothesis $Q \hookrightarrow^* Q'$ and there is $P' \in \texttt{addG}(\mathbb{E}[(\!|P_1|\!) l])$ such that $\texttt{nf}(P') \equiv_{Ex} \texttt{nf}(Q')$. Since the garbage does not forbid reductions we have that there exists $P'' \in \texttt{addG}(\mathbb{E}[l\langle (\!|P_1|\!) \rangle \mid \texttt{Rew}\ l \mid S])$ such that $P' \hookrightarrow^* P''$. From Lemma 27 $\texttt{nf}(P') \hookrightarrow^* \texttt{nf}(P'')$. Thanks to Lemma 25 from $\texttt{nf}(P') \equiv_{Ex} \texttt{nf}(Q')$ and $P' \hookrightarrow^* P''$ we have that $\texttt{nf}(Q') \hookrightarrow^* \texttt{nf}(Q'')$ with $P''' \in \texttt{addG}(P'')$ and $\texttt{nf}(P''') \equiv_{Ex} \texttt{nf}(Q'')$. By composing garbage we also have $P''' \in \texttt{addG}(P)$. The thesis follows.

Let us consider the case of a killer process. We have a few subcases, corresponding to the killer processes KillM, KillP and KillT. The main idea here is that the communication is undone by a Rew. Let us consider a KillM process. We have that $P \equiv \mathbb{E}[a\langle (\!|P_1|\!), l \rangle \mid (a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h \rangle \mid \texttt{Rew}\ l])$ and $Q_1 = \mathbb{E}[\langle (h)\texttt{Msg}\ a\ (\!|P_1|\!)\ h) \rangle \mid \texttt{Rew}\ l]$. Then we have:

$$\begin{aligned} Q_1 \rightharpoondown & \mathbb{E}[l\langle (h)\texttt{Msg}\ a\ (\!|P_1|\!)\ h) \rangle \mid l(Z) \triangleright Z\ l] \hookrightarrow \\ & \quad \mathbb{E}[((h)\texttt{Msg}\ a\ (\!|P_1|\!)\ h) l] \rightharpoondown^* \mathbb{E}[a\langle (\!|P_1|\!), l \rangle \mid (\texttt{KillM}\ a\ l)] = Q' \end{aligned}$$

with $\texttt{nf}(Q') = \texttt{nf}(P)$. Using the same approach used in the case of Rew, we can compose this result with the inductive hypothesis to get the thesis. Let us consider a KillP process. We have that $P \equiv \mathbb{E}[h\langle (\!|P_1|\!) \rangle \mid l\langle (\!|Q|\!) \rangle \mid$

$(h(W)|l(Z) \triangleright k\langle (h)\mathtt{Par}W\ Z\ h\rangle \mid \mathtt{Rew}\ k)]$ and $Q_1 = \mathbb{E}[k\langle (h)\mathtt{Par}(\!|P_1|\!)\ (\!|Q|\!)\ h\rangle \mid \mathtt{Rew}\ k]$. Then we have:

$$Q_1 \twoheadrightarrow \mathbb{E}[k\langle (h)\mathtt{Par}(\!|P_1|\!)\ (\!|Q|\!)\ h\rangle \mid k(Z) \triangleright Z\ k] \to$$
$$\mathbb{E}[((h)\mathtt{Par}\ (\!|P_1|\!)\ (\!|Q|\!)\ h)k] \twoheadrightarrow \mathbb{E}[\nu h, l.\, (\!|P_1|\!)l \mid (\!|Q|\!)h \mid (\mathtt{KillP}\ l\ h\ k)]$$

and by using Lemma 23 and Lemma 55 (to ensure that processes P_1 and Q have no toplevel restrictions) we have:

$$\mathbb{E}[\nu h, l.\, (\!|P_1|\!)l \mid (\!|Q|\!)h \mid (\mathtt{KillP}\ l\ h\ k)] \hookrightarrow^*$$
$$\mathbb{E}[\nu h, l.\, l\langle (\!|P_1|\!)\rangle \mid \mathtt{Rew}\ l \mid S_1 \mid h\langle (\!|Q|\!)\rangle \mid \mathtt{Rew}\ h \mid S_2 \mid (\mathtt{KillP}\ l\ h\ k)] = Q'$$

To have a correspondence with P, we have to show that names h and l were restricted also in P, and with the same scope. By using Lemma 58 we have that $P \equiv \mathbb{E}_1[\nu l, h.\, \mathtt{addG}(h\langle (\!|P_1|\!)\rangle \mid \mathtt{Rew}\ h \mid l\langle (\!|Q|\!)\rangle \mid \mathtt{Rew}\ l \mid S)]$ (where S includes $(\mathtt{KillP}\ l\ h\ k)$), and

$$\mathtt{nf}(\mathbb{E}_1[\nu l, h.\, \mathtt{addG}(h\langle (\!|P_1|\!)\rangle \mid \mathtt{Rew}\ h \mid l\langle (\!|Q|\!)\rangle \mid \mathtt{Rew}\ l \mid S)]) \equiv$$
$$\mathtt{nf}(\mathtt{addG}(Q'))$$

as desired.

If the communication is an internal communication in a trigger, we have that

$$P \equiv \mathbb{E}[\nu c.\, c\langle (\!|P_1|\!)\rangle \mid c(Z) \triangleright Z\ k] \hookrightarrow$$
$$\mathbb{E}[\nu c.\, (\!|P_1|\!)k] \equiv_{Ex} \mathbb{E}[\nu c.\, c\langle (\!|P_1|\!)\rangle \mid c(Z) \triangleright Z\ k]$$

as desired. □

Appendix C.3. Proofs of Section 4.5

Before proving Lemma 29 we show that barbs are preserved by the encoding.

Lemma 59. *For each consistent configuration M, if $M \downarrow_a$ then* $\mathtt{nf}((\!|M|\!)) \downarrow_a$

Proof. Easy, by definition of barbs and of the encoding. □

Lemma 29. *If $M \downarrow_a$ and $(\!|M|\!) \hookrightarrow^* Q$ then $Q \hookrightarrow^* \downarrow_a$.*

$$\begin{tikzcd}
(\!|M|\!) \arrow[r, hook, "*"] \arrow[d, "*"] \arrow[dr, phantom, "27"] & Q \arrow[d, "*"] \\
\mathtt{nf}((\!|M|\!)) \arrow[r, hook, "*"] \arrow[d, phantom, "\mathtt{addG}"] & Q' \arrow[d, hook, "*"] \\
P' \arrow[d, "*"] \arrow[r, phantom, "28"] & Q'' \arrow[d, "*"] \\
\mathtt{nf}(P') \arrow[r, phantom, "\equiv_{Ex}"] & \mathtt{nf}(Q'')
\end{tikzcd}$$

Figure C.12: Correspondence schema of barbs (numbers refer to Lemmas).

Proof. By Lemma 59 we have that if $M \downarrow_a$ then $\mathtt{nf}((\!|M|\!)) \downarrow_a$. By definition of normal form we have that $(\!|M|\!) \rightharpoondown^* \mathtt{nf}((\!|M|\!))$, and by hypothesis we have that $(\!|M|\!) \hookrightarrow^* Q$. Using Lemma 27 we have that $\mathtt{nf}(M) \hookrightarrow^* Q'$ and $Q \rightharpoondown^* Q'$. Moreover, by Lemma 28 there exist Q'' and P' such that $Q' \hookrightarrow^* Q''$, $P' \in \mathtt{addG}(\mathtt{nf}((\!|M|\!)))$ and $\mathtt{nf}(P') \equiv_{Ex} \mathtt{nf}(Q'')$. Since addG and $\equiv_{Ex}$ do not remove barbs we have $\mathtt{nf}(Q'') \downarrow_a$ as desired. The proof is graphically depicted in Figure C.12. $\square$

Before proving Lemma 30 below we prove some auxiliary results.

Applications never remove barbs.

Lemma 60. *For any HOπ^+ process P, if $P \downarrow_a$ and $P \rightharpoondown Q$, then also $Q \downarrow_a$.*

Proof. From the definition of $\downarrow_a$ and of $\rightharpoondown$. $\square$

Administrative steps do not add barbs.

Lemma 61. *For each consistent configuration M such that $\nu k.\, k : P \Rrightarrow M$, if $(\!|M|\!) \hookrightarrow^* Q$ and $Q \downarrow_a$, then $\mathtt{nf}((\!|M|\!)) \downarrow_a$.*

Proof. By hypothesis $(\!|M|\!) \hookrightarrow^* Q$ with $Q \downarrow_a$, and by definition $(\!|M|\!) \rightharpoondown^* \mathtt{nf}((\!|M|\!))$. By applying Lemma 27 we get $Q \rightharpoondown^* Q'$ and $\mathtt{nf}((\!|M|\!)) \hookrightarrow^* Q'$. Since $Q \downarrow_a$, and since applications ($\rightharpoondown$) do not remove barbs (by Lemma 60), we also have that $Q' \downarrow_a$. The above reasoning is depicted in Figure C.13.

We have to show that if Q' has a barb then $\mathtt{nf}((\!|M|\!))$ has the same barb. Since $\nu k.\, k : P \Rrightarrow M$ then $\mathtt{fn}(M) \cap \mathcal{K} = \emptyset$ and also $\mathtt{fn}(Q) \cap \mathcal{K} = \emptyset$ (by Lemma 22). Thus we have no need to consider barbs in $\mathcal{K}$.

$$\begin{tikzcd}
(\!|M|\!) \arrow[r, hook, "*"] \arrow[d, "*"] \arrow[dr, phantom, "27"] & Q \downarrow_a \arrow[d, "*"] \\
\mathtt{nf}((\!|M|\!)) \arrow[r, hook, "*"] & Q' \downarrow_a
\end{tikzcd}$$

Figure C.13: Barbs with respect to administrative steps (numbers refer to Lemmas).

We proceed by case analysis on the reduction $\hookrightarrow$. Since Q' is generated from $\mathtt{nf}((\!|M|\!))$ via administrative steps, then applications of the form $(((X\ c)c\langle(\!|P|\!)\rangle)\ (\!|Q|\!)\ c)$ or communications of the form $c\langle(\!|P|\!)\rangle \mid (c(Z) \triangleright Z\ l)$ or with a trigger of the form $(k(Z) \triangleright (\mathtt{Msg}\ a\ (\!|P|\!)\ l_1) \mid (\mathtt{Trig}\ ((X\ c)c\langle(\!|P|\!)\rangle)\ a\ l_2))$ are never enabled. Thus, only communications involving a killer or a $\mathtt{Rew}$ process may happen. A communication involving a killer does not add barbs (since $\mathtt{fn}(Q') \cap \mathcal{K} = \emptyset$). A communication involving a $\mathtt{Rew}$ process does not add barbs since it produces an application. Similarly, all the applications involving killer, $\mathtt{Rew}$ and $\mathtt{Mem}$ processes do not add barbs. Nevertheless, other applications, such as the application of a $\mathtt{Msg}$ process, may create barbs. However, no such application is enabled in $\mathtt{nf}((\!|M|\!))$. The only way they may become enabled is via a kill followed by a $\mathtt{Rew}$. However, the created barb was already present in $\mathtt{nf}((\!|M|\!))$ before the kill. This completes the proof. □

We now show that barbs of $\mathtt{nf}((\!|M|\!))$ come from barbs of M.

Lemma 62. *For each consistent configuration M, if $\mathtt{nf}((\!|M|\!)) \downarrow_a$ then $M \downarrow_a$.*

Proof. By structural induction on M. Note that only rhoπ names may be free, since all the names coming from rhoπ keys are bound (by Lemma 22). Since sub-terms of consistent configurations are not consistent in general, we have to consider in the induction both consistent configurations and their sub-terms. If $M = \kappa : P$, we proceed by structural induction on P and by case analysis on κ. We will consider just the case in which $\kappa = k$, the other case with $\kappa = \langle h_i, \tilde{h}\rangle \cdot k$ is similar. If $P = \mathbf{0}$ then we have that $\mathtt{nf}((\!|k : \mathbf{0}|\!)) = k\langle\mathtt{Nil}\rangle \mid \mathtt{Rew}\ k$, but since $k \notin \mathcal{N}$ the process $\mathtt{nf}((\!|k : \mathbf{0}|\!))$ does not show any relevant barb. If $P = a\langle Q\rangle$ then we have that $\mathtt{nf}((\!|k : a\langle Q\rangle|\!)) = a\langle(\!|Q|\!), k\rangle \mid (a(X, \backslash k) \triangleright k\langle(h)\mathtt{Msg}\ a\ (\!|Q|\!)\ h\rangle \mid \mathtt{Rew}\ k)$, which shows a barb on a. Since also $M \downarrow_a$, we are done. If $P = a(X) \triangleright Q$,we have that $\mathtt{nf}((\!|k : a(X) \triangleright Q|\!)) = \nu\mathtt{t}.\,\overline{\mathtt{t}} \mid (a(X,h)|\mathtt{t} \triangleright R) \mid (\mathtt{t} \triangleright S)$ (for some R and S). Since $\mathtt{t}$ is restricted then the entire process does not show any barb, and we are done. If $P = Q_1 \mid Q_2$, the tag κ has to be a key, since we are dealing with consistent configurations. So, we have that $\texttt{nf}(\llparenthesis k : (Q_1 \mid Q_2)\rrparenthesis) = \nu h, l.\, \texttt{nf}(\llparenthesis Q_1 \rrparenthesis h) \mid \texttt{nf}(\llparenthesis Q_2 \rrparenthesis l) \mid (h(W)|l(Z) \triangleright S)$. The process may show a barb because of either $\texttt{nf}(\llparenthesis Q_1 \rrparenthesis h)$ or $\texttt{nf}(\llparenthesis Q_2 \rrparenthesis l)$ (or both). Let us suppose that it is because of $\texttt{nf}(\llparenthesis Q_1 \rrparenthesis h)$, that is $\texttt{nf}(\llparenthesis Q_1 \rrparenthesis h) \downarrow_a$. By definition of $\llparenthesis \_ \rrparenthesis$ we have that $\texttt{nf}(\llparenthesis Q_1 \rrparenthesis h) = \texttt{nf}(\llparenthesis h : Q_1 \rrparenthesis)$ and hence $\texttt{nf}(\llparenthesis h : Q_1 \rrparenthesis) \downarrow_a$. Now, by applying the inductive hypothesis we have that $(h : Q_1) \downarrow_a$ and then also $k : (Q_1 \mid Q_2) \downarrow_a$, as desired. The other cases are similar.

If $M = \mathbf{0}$, we have that $\llparenthesis \mathbf{0} \rrparenthesis = \mathbf{0}$ and the thesis banally follows. If $M = M_1 \mid M_2$ we have that $\texttt{nf}(\llparenthesis M_1 \mid M_2 \rrparenthesis) = \texttt{nf}(\llparenthesis M_1 \rrparenthesis \mid \llparenthesis M_2 \rrparenthesis) = \texttt{nf}(\llparenthesis M_1 \rrparenthesis) \mid \texttt{nf}(\llparenthesis M_2 \rrparenthesis)$ and we can conclude by applying the inductive hypothesis on $\texttt{nf}(\llparenthesis M_1 \rrparenthesis)$ and $\texttt{nf}(\llparenthesis M_2 \rrparenthesis)$. If $M = \nu u.\, M_1$ we have that $\texttt{nf}(\llparenthesis \nu u.\, M_1 \rrparenthesis) = \nu u.\, \llparenthesis \texttt{nf}(M_1) \rrparenthesis$ and we can conclude by applying the inductive hypothesis on $\llparenthesis \texttt{nf}(M_1) \rrparenthesis$. If $M = [\kappa_1 : a\langle P \rangle \mid \kappa_2 : a(X) \triangleright Q; k]$, then $\texttt{nf}(\llparenthesis M \rrparenthesis) = k(Z) \triangleright R$ (for some R), that shows no barbs and we can conclude. □

Lemma 30. *If $\llparenthesis M \rrparenthesis \hookrightarrow^* \downarrow_a$ then $M \downarrow_a$.*

Proof. By concatenating Lemma 61 and Lemma 62. □

Before proving Proposition 1 below, we prove some auxiliary results.

Each of the lemmas below shows that one of the axioms in $\equiv_{Ex}$ is correct with respect to weak bf barbed bisimulation (actually, the statement for axiom Ex.P is slightly weaker). We consider together each axiom $L \equiv_{Ax} R$ from $\equiv_{Ax}$ and its normal form $\texttt{nf}(L) \equiv_{Ex} \texttt{nf}(R)$, since this is obtained via applications. Below, we denote a multi-holes context as $\mathbb{C}$.

Lemma 63. *Axiom Ex.C and its normal form are correct with respect to weak bf barbed bisimulation.*

Proof. We show that the relation $\mathcal{R}$ below is a weak bf barbed bisimulation.

$$\begin{aligned}
\mathcal{R}_1 =& \{((\texttt{KillP}\ l\ h\ k), (\texttt{KillP}\ h\ l\ k)) \mid h, l, k \in \mathcal{K}\} \\
\mathcal{R}_2 =& \{((a(X)|b(Y) \triangleright R), (b(Y)|a(X) \triangleright R)) \mid a, b \in \mathcal{N} \wedge X, Y \in \mathcal{V} \wedge R \in \mathcal{P}\} \\
\mathcal{R}' =& \mathcal{R}_1 \cup \mathcal{R}_2 \\
\mathcal{R} =& \{(\mathbb{C}[P_1, .., P_n], \mathbb{C}[Q_1, .., Q_n]) \mid n \in \mathbb{N} \wedge \forall i \in \{1 \ldots n\}.(P_i, Q_i) \in \mathcal{R}'\}
\end{aligned}$$

Let us consider the barbs. Since the context is the same on both the sides, and the processes in the holes show no barbs, the barbs coincide.

Let us consider reductions. We consider only challenges from the left, since the reasoning is analogous for challenges from the right. If the process $\mathbb{C}[P_1, .., P_n]$ does a reduction, it is because either the context evolves by itself, or one of the hole processes reduces by itself, or because of an interaction between the context and one hole (no interaction between holes is possible).

If the context performs a reduction by itself, that is $\mathbb{C}[P_1, .., P_n] \to \mathbb{C}'[P_1, .., P_m]$, then also $\mathbb{C}[Q_1, .., Q_n] \to \mathbb{C}'[Q_1, .., Q_m]$. Note that the number of holes may change because of the reduction.

Let us consider reductions in one hole. If $\mathbb{C}[P_1, .., (\texttt{KillP}\ l\ h\ k), .., P_n] \rightharpoondown \mathbb{C}[P_1, .., (h(W)|l(Z) \triangleright R), .., P_n]$ (for some R) then also $\mathbb{C}[Q_1, .., (\texttt{KillP}\ h\ l\ k), .., Q_n] \rightharpoondown \mathbb{C}[Q_1, .., (l(Z)|h(W) \triangleright R), .., Q_n]$ and we are still in the same relation.

Let us consider interactions between the context and one hole. For $\mathbb{C}[P_1, .., a(X)|b(Y) \triangleright R, .., P_n]$ to reduce, we need in the context two messages of the form $a\langle S_1\rangle$ and $b\langle S_2\rangle$. If so, we have that $\mathbb{C}[P_1, .., a(X)|b(Y) \triangleright R, .., P_n] \to \mathbb{C}'[P_1, .., R\{^{S_1,S_2}/_{X,Y}\}, .., P_m]$ and on the other side $\mathbb{C}[Q_1, .., b(Y)|a(X) \triangleright R, .., Q_n] \to \mathbb{C}'[Q_1, .., R\{^{S_2',S_1'}/_{Y,X}\}, .., Q_m]$. Since identity is included in $\mathcal{R}$ (it suffices to consider a 0-ary context), and since $(S_i, S_i') \in \mathcal{R}$ (since they are subterms) we have $(R\{^{S_1,S_2}/_{X,Y}\}, R\{^{S_2',S_1'}/_{Y,X}\}) \in \mathcal{R}$, and also $(\mathbb{C}'[P_1, .., R\{^{S_1,S_2}/_{X,Y}\}, .., P_m], \mathbb{C}'[Q_1, .., R\{^{S_2',S_1'}/_{Y,X}\}, .., Q_m]) \in \mathcal{R}$, as desired. □

Before proving the lemma concerning axiom $Ex.P$ we prove as auxiliary result that the function `addG` does not introduce barbs.

Lemma 64. *For any HOπ^+ process P, if `addG`$(P) \downarrow_a$ then $P \downarrow_a$.*

Proof. Easy, by looking at the definition of `addG` (Definition 15). □

Lemma 65. *Applications of axiom $Ex.P$ and of its normal form to well formed HOπ^+ processes are correct with respect to weak bf barbed bisimulation.*

Proof. Note that the axiom $Ex.P$ alone is not correct, since the left term has barbs at l_1 and l_2 which are not matched by the right term. However, in well formed processes keys are always bound. We show below that applications of the axiom and of its normal form to well formed processes are always correct.

Thanks to Lemma 58 restrictions on l_1 and l_2 may occur only in processes of some forms. In particular, the only possibility is that for both l_1 and l_2 case (1) applies. Thus the process to which the axiom is applied is of the form $\mathbb{E}[\nu l_1, l_2 .\, R']$ with $R' \in \mathtt{addG}(R'')$ where R'' is obtained via zero or more applications from $l_1\langle (\!| P |\!) \rangle \mid l_2\langle (\!| Q |\!) \rangle \mid (\mathtt{KillP}\ l_1\ l_2\ l_3)$. On the right side we can assume l_1 and l_2 do not occur.

Let $S(l_1, l_2, l_3)$ denote a process of one of the following forms:

$$S(l_1, l_2, l_3) = (\mathtt{KillP}\ l_1\ l_2\ l_3)$$
$$S(l_1, l_2, l_3) = (l_1(Z)|l_2(W) \triangleright l_3\langle (h)\mathtt{Par}\ Z\ W\ h\rangle \mid \mathtt{Rew}\ l)$$

Let T_l be either a process of the form $T_l = \mathtt{Rew}\ l$ or of the form $T_l = (l(Z) \triangleright Z\ l)$. Let us write $\mathtt{addG}(P)$ for any process $Q \in \mathtt{addG}(P)$. Let

$$\begin{aligned}\mathcal{R}' = \{&(\nu l_1, l_2 .\, \mathtt{addG}(l_1\langle (\!| P |\!) \rangle \mid l_2\langle (\!| Q |\!) \rangle \mid S(l_1, l_2, l))),\\ &(\nu l_1, l_2 .\, \mathtt{addG}(l\langle (h)\mathtt{Par}\ (\!| P |\!)\ (\!| Q |\!)\ h\rangle \mid T_l))\}\\ \mathcal{R} = \{&(\mathbb{C}[P_1, \ldots, P_n], \mathbb{C}[Q_1, \ldots, Q_n]) \mid (P_i, Q_i) \in \mathcal{R}' \wedge\\ &\mathbb{C}[P_1, \ldots, P_n], \mathbb{C}[Q_1, \ldots, Q_n]\ \mathit{well\ formed}\}\end{aligned}$$

We now prove that the relation $\mathcal{R}$ is a weak bf barbed bisimulation. The thesis will follow since in well formed processes all the contexts where the axiom can be applied have this form.

Let us consider barbs. On both sides barbs shown by the contexts $\mathbb{C}$ are banally matched. The process $\nu l_1, l_2 .\, \mathtt{addG}(l_1\langle (\!| P |\!) \rangle \mid l_2\langle (\!| Q |\!) \rangle \mid S(l_1, l_2, l))$ does not show any barb, since $\mathtt{addG}$ does not add any barb thanks to Lemma 64. On the other side, the process $\nu l_1, l_2 .\, \mathtt{addG}(l\langle (h)\mathtt{Par}\ (\!| P |\!)\ (\!| Q |\!)\ h\rangle \mid T_l)$ shows only a barb at l. We have that:

$$\begin{aligned}&\nu l_1, l_2 .\, \mathtt{addG}(l_1\langle (\!| P |\!) \rangle \mid l_2\langle (\!| Q |\!) \rangle \mid S(l_1, l_2, l)) \hookrightarrow^*\\ &\nu l_1, l_2 .\, \mathtt{addG}(l\langle (h)\mathtt{Par}\ (\!| P |\!)\ (\!| Q |\!)\ h\rangle \mid \mathtt{Rew}\ l)\end{aligned}$$

showing a barb at l as well.

Let us consider reductions. If $\mathbb{C}[P_1, \ldots, P_n]$ reduces it is because the context reduces by itself or because of the hole processes. The first case is banally matched by the process $\mathbb{C}[Q_1, .., Q_n]$. In the second case, the process $\nu l_1, l_2 .\, \mathtt{addG}(l_1\langle (\!| P |\!) \rangle \mid l_2\langle (\!| Q |\!) \rangle \mid S(l_1, l_2, l))$ reduces. We have three cases: the processes added by $\mathtt{addG}$ are not involved, the processes added by $\mathtt{addG}$ reduce alone or they interact with the other processes. In the first case, if the reduction is the application of the process $S(l_1, l_2, l)$ then the right process can match the reduction by staying idle. If it is a communication on the channels l_1 and l_2 then we have that:

$$\begin{aligned}&\nu l_1, l_2 .\, \mathtt{addG}(l_1\langle (\!| P |\!) \rangle \mid l_2\langle (\!| Q |\!) \rangle \mid (l_1(W)|l_2(Z) \triangleright l\langle (h)\mathtt{Par}\ (\!| P |\!)\ (\!| Q |\!)\ h\rangle \mid \mathtt{Rew}\ l)) \rightarrow\\ &\nu l_1, l_2 .\, \mathtt{addG}(l\langle (h)\mathtt{Par}\ (\!| P |\!)\ (\!| Q |\!)\ h\rangle \mid \mathtt{Rew}\ l))\end{aligned}$$

This step is matched by the right process by staying idle, since the process in the i-th hole on the two sides become equal, and we can put them in the context since the identity belongs to the relation (actually, processes added by addG may be different, but they have no impact). The case where processes inside addG reduce alone is banally matched. In the third case, since the process $\nu l_1, l_2.\, \texttt{addG}(l_1\langle(\!|P|\!)\rangle \mid l_2\langle(\!|Q|\!)\rangle \mid S(l_1, l_2, l))$ is well formed then the only processes inside $\texttt{addG}(\_)$ able to interact are a $(l_1(Z) \triangleright Z\ l_1)$ and/or a $(l_2(Z) \triangleright Z\ l_2)$. The two cases are similar, so we consider just the first one. Assume

$$\begin{aligned}&\nu l_1, l_2.\, \texttt{addG}((l_1(Z) \triangleright Z\ l_1) \mid l_1\langle(\!|P|\!)\rangle \mid l_2\langle(\!|Q|\!)\rangle \mid S(l_1, l_2, l)) \hookrightarrow\\ &\nu l_1, l_2.\, \texttt{addG}(((\!|P|\!)\ l_1 \mid l_2\langle(\!|Q|\!)\rangle \mid S(l_1, l_2, l))\end{aligned}$$

We have

$$\begin{aligned}&\nu l_1, l_2.\, \texttt{addG}(l\langle(h)\texttt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ h\rangle \mid T_l) \hookrightarrow^*\\ &\nu l_1, l_2.\, \texttt{addG}(\nu h, k.\, (\!|P|\!)\, h \mid (\!|Q|\!)\, k \mid (\texttt{KillP}\ h\ k\ l))\end{aligned}$$

Using Lemma 23:

$$\begin{aligned}&\nu l_1, l_2.\, \texttt{addG}(\nu h, k.\, (\!|P|\!)\, h \mid (\!|Q|\!)\, k \mid (\texttt{KillP}\ h\ k\ l)) \hookrightarrow^*\\ &\nu l_1, l_2.\, \texttt{addG}(\nu h, k.\, (\!|P|\!)\, h \mid k\langle(\!|Q|\!)\rangle \mid (\texttt{KillP}\ h\ k\ l))\end{aligned}$$

since all the garbage can be moved to addG and since Q does not contain restrictions thanks to Lemma 55. Now, using α-conversion to swap names l_1 and l_2 with h and k and exploiting the fact that addG is closed under α-conversion we get $\nu h, k.\, \texttt{addG}(\nu l_1, l_2.\, (\!|P|\!)\, l_1 \mid l_2\langle(\!|Q|\!)\rangle \mid (\texttt{KillP}\ l_1\ l_2\ l))$. Since h, k are just used by the $\texttt{addG}(\_)$ context we can rewrite the process as $\nu l_1, l_2.\, \texttt{addG}((\!|P|\!)\, l_1 \mid l_2\langle(\!|Q|\!)\rangle \mid (\texttt{KillP}\ l_1\ l_2\ l))$, where the restriction on k, h has been moved to the context.

Let us consider now the reductions of $\mathbb{C}[Q_1, \ldots, Q_n]$. We have three cases: the context reduces by itself, the hole reduces by itself or the hole and the context interact. The first case is trivial. In the second case, if the reduction is the application of T_l then the step is matched by $\mathbb{C}[P_1, \ldots, P_n]$ by staying idle, and we are still in the same relation. If T_l is already in its applied form then we have:

$$\begin{aligned}&\nu l_1, l_2.\, \texttt{addG}(l\langle(h)\texttt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ h\rangle \mid l(Z) \triangleright Z\ l) \to\\ &\nu l_1, l_2.\, \texttt{addG}(\texttt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ l)\end{aligned}$$

and on the other side we have that:

$$\nu l_1, l_2.\,\texttt{addG}(l_1\langle(\!|P|\!)\rangle \mid l_2\langle(\!|Q|\!)\rangle \mid S(l_1,l_2,l) \hookrightarrow^* \nu l_1, l_2.\,\texttt{addG}(\texttt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ l)$$

and since the identity is contained in the relation, we are still in the same relation. In the last case, since we only consider well formed processes, thanks to Lemma 24 there are no two `KillP` processes waiting on the same channels. Thus the context may not interact with the hole, and this case will never happen. □

Before proving the lemma concerning axiom $Ex.A$ we prove as auxiliary result that the function `addG` has no impact on weak bf barbed bisimulation. We show the result for each form of garbage, and compose the partial results together in Lemma 68.

Lemma 66. *Let T_l be a process of the form $T_l = \texttt{Rew}\ l$ or $T_l = (l(Z) \triangleright Z\ l)$. The relation $\mathcal{R} = \{(\mathbb{C}[T_l], \mathbb{C}[\mathbf{0}])\}$ is a weak bf barbed bisimulation.*

Proof. Let us start with barbs. Since T_l does not show barbs, the barbs of $\mathbb{C}[T_l]$ and $\mathbb{C}[\mathbf{0}]$ coincide.

Let us consider the reductions. If $\mathbb{C}[T_l]$ reduces then it is either because the context reduces by itself, or because T_l reduces by itself or because of an interaction between the context and T_l. In the first case the reduction is banally matched by the process $\mathbb{C}[\mathbf{0}]$. The second case implies that $T_l = \texttt{Rew}\ l$ and $\mathbb{C}[\texttt{Rew}\ l] \rightharpoondown \mathbb{C}[l(Z) \triangleright Z\ l]$. $\mathbb{C}[\mathbf{0}]$ matches this step by staying idle. The third case implies that $T_l = (l(Z) \triangleright Z\ l)$ and the presence of a message in the context of the form $l\langle(\!|P|\!)\rangle$. Hence, we have that $\mathbb{C}[(l(Z) \triangleright Z\ l)] \equiv \mathbb{C}'[l\langle(\!|P|\!)\rangle \mid (l(Z) \triangleright Z\ l)]$ and $\mathbb{C}[\mathbf{0}] \equiv \mathbb{C}'[l\langle(\!|P|\!)\rangle]$. By Lemma 16 we know $\mathbb{C}'[l\langle(\!|P|\!)\rangle] \equiv \mathbb{C}''[l\langle(\!|P|\!)\rangle \mid T_l]$. Then on the other side we have that $\mathbb{C}'[l\langle(\!|P|\!)\rangle \mid (l(Z) \triangleright Z\ l)] \equiv \mathbb{C}''[l\langle(\!|P|\!)\rangle \mid (l(Z) \triangleright Z\ l) \mid T_l]$. Thus, the reduction $\mathbb{C}''[l\langle(\!|P|\!)\rangle \mid (l(Z) \triangleright Z\ l) \mid T_l] \rightarrow \mathbb{C}''[(\!|P|\!)l \mid T_l]$ can be matched on the right side by $\mathbb{C}''[l\langle(\!|P|\!)\rangle \mid T_l] \hookrightarrow^* \mathbb{C}''[(\!|P|\!)l]$, and we are still in the same relation since $\mathbb{C}''[(\!|P|\!)l \mid T_l] \equiv \mathbb{C}'''[T_l]$ and $\mathbb{C}''[(\!|P|\!)l] \equiv \mathbb{C}'''[\mathbf{0}]$, as desired. □

Lemma 67. *Let $T(l,a)$ be a process of the form $T(l,a) = (\texttt{KillM}\ a\ l)$ or $T(l,a) = (a(X, \backslash l) \triangleright l\langle(h)\texttt{Msg}\ a\ X\ h\rangle \mid \texttt{Rew}\ l)$. The relation $\mathcal{R} = \{(\mathbb{C}[T(l,a)], \mathbb{C}[\mathbf{0}])\}$ is a weak bf barbed bisimulation.*

Proof. Let us consider the barbs. Since a process of the form $(\texttt{KillM}\ a\ l)$ or $(a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h\rangle \mid \texttt{Rew}\ l)$ does not show any barbs, then the barbs of $\mathbb{C}[T(l,a)]$ and $\mathbb{C}[\mathbf{0}]$ coincide.

Let us consider the reductions. If the context evolves by itself, the reduction is banally matched. If $T(l,a) = (\texttt{KillM}\ a\ l)$ then the only possible reduction is the application

$$\mathbb{C}[(\texttt{KillM}\ a\ l)] \rightharpoondown \mathbb{C}[(a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h\rangle \mid \texttt{Rew}\ l)]$$

which is matched by $\mathbb{C}[\mathbf{0}]$ by staying idle. If in $\mathbb{C}[(a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h\rangle \mid \texttt{Rew}\ l)]$ the hole process and the context interact, it is because there is a message of the form $a\langle (\!|P|\!), l\rangle$ in the context. Hence

$$\begin{array}{l}\mathbb{C}[(a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h\rangle \mid \texttt{Rew}\ l)] \equiv \\ \mathbb{C}'[a\langle (\!|P|\!), l\rangle \mid (a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h\rangle \mid \texttt{Rew}\ l)]\end{array}$$

This implies that also $\mathbb{C}[\mathbf{0}] \equiv \mathbb{C}'[a\langle (\!|P|\!), l\rangle]$, and by Lemma 17 we know that $\mathbb{C}'[a\langle (\!|P|\!), l\rangle] \equiv \mathbb{C}''[a\langle (\!|P|\!), l\rangle \mid S]$ with $S = (\texttt{KillM}\ a\ l)$ or $S = (a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h\rangle \mid \texttt{Rew}\ l)$. Then

$$\begin{array}{l}\mathbb{C}'[a\langle (\!|P|\!), l\rangle \mid (a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h\rangle \mid \texttt{Rew}\ l)] \equiv \\ \mathbb{C}''[a\langle (\!|P|\!), l\rangle \mid (a(X, \backslash l) \triangleright l\langle (h)\texttt{Msg}\ a\ X\ h\rangle \mid \texttt{Rew}\ l) \mid S] \hookrightarrow \\ \mathbb{C}''[l\langle (h)\texttt{Msg}\ a\ (\!|P|\!)\ h\rangle \mid \texttt{Rew}\ l) \mid S] \equiv \mathbb{C}'''[S]\end{array}$$

and on the other side we have that

$$\mathbb{C}''[a\langle (\!|P|\!), l\rangle \mid S] \hookrightarrow^* \mathbb{C}''[l\langle (h)\texttt{Msg}\ a\ (\!|P|\!)\ h\rangle \mid \texttt{Rew}\ l] \equiv \mathbb{C}'''[\mathbf{0}]$$

and we remain in the same relation, as desired. □

Lemma 68. *For any $HO\pi^+$ process $P \equiv \nu\tilde{a}.\, P'$, the relation $\mathcal{R} = \{\nu\tilde{a}.\, P'$, $\nu\tilde{a}.\,(P' \mid \nu\tilde{b}.\, Q)\}$ with Q a parallel composition of processes as in Definition 15 is a weak bf barbed bisimulation.*

Proof. By induction on the number of parallel components inside Q. The base case $Q = \mathbf{0}$ reduces to the identity since we can can garbage collect via structural congruence names contained in $\tilde{b}$ (and structural congruence is a weak bf barbed bisimulation by Lemma 32). In the inductive case, we do a case analysis on the last process Q_n of the parallel composition.

$Q_n = \texttt{Rew}\ l$**:** by Lemma 66 we know that $\mathbb{C}[Q_n] \dot{\approx} \mathbb{C}[\mathbf{0}]$. By choosing $\mathbb{C}[\bullet] = \nu\tilde{a}.\,(P' \mid \nu\tilde{b}.\,\bullet \mid \prod_{i=1..n-1} Q_i)$ we have that $\nu\tilde{a}.\,(P' \mid \nu\tilde{b}.\,\prod_{i=1..n} Q_i) \dot{\approx} \nu\tilde{a}.\,(P' \mid \nu\tilde{b}.\,\prod_{i=1..n-1} Q_i)$. By inductive hypothesis we have $\nu\tilde{a}.\,(P' \mid \nu\tilde{b}.\,\prod_{i=1..n-1} Q_i) \dot{\approx} \nu\tilde{a}.\,P'$ and by transitivity $\nu\tilde{a}.\,(P' \mid \nu\tilde{b}.\,\prod_{i=1..n} Q_i) \dot{\approx} \nu\tilde{a}.\,P'$.

$Q_n = \texttt{KillM}\ a\ l$**:** similar to the case above, using Lemma 67 instead of Lemma 66.

$Q_n = \nu\texttt{t}.\,(a(X,k)|\texttt{t} \triangleright Q)$**:** trivial, since the process cannot interact.

$Q_n = \nu c, \texttt{t}.\,(\texttt{KillT}\ ((X)c(\!|P|\!)))\ \texttt{t}\ l\ a)$**:** trivial, since the process reduces to a process that cannot interact. □

Lemma 69. *Axiom Ex.A and its normal form are correct with respect to weak bf barbed bisimulation.*

Proof. Let $S(l_1,l_2,l_3)$ denote either a process of the form $S(l_1,l_2,l_3) = (\texttt{KillP}\ l_1\ l_2\ l_3)$ or of the form $S(l_1,l_2,l_3) = (l_1(Z)|l_2(W) \triangleright l_3\langle (h)\texttt{Par}\ Z\ W\ h\rangle \mid \texttt{Rew}\ l)$. Let T_l denote either a process of the form $T_l = \texttt{Rew}\ l$ or of the form $T_l = (l(Z) \triangleright Z\ l)$. Let $A((\!|P|\!), l)$ denote either a process of the form $A((\!|P|\!), l) = l\langle (\!|P|\!)\rangle$ or of the form $A((\!|P|\!), l) = (\!|P|\!)l$. A weak bf barbed bisimulation containing axiom $Ex.A$ and its normal form is quite large. For simplicity we consider a relation which is closed only under challenges from the left term. Extending the relation and the proof by considering the symmetric cases is a tedious but easy work. The considered relation is $\mathcal{R}$ in Figure C.14.

Let us consider the barbs. The processes of the form T_l, $S(l_1,l_2,l_3)$, $A((\!|P|\!), l)$ have no barbs. Also, function $\texttt{addG}(\_)$ does not add any barb thanks to Lemma 64. Moreover, all the messages on channels $l \in \mathcal{K}$ are on restricted channels. So the only barbs are those shown by the context $\mathbb{C}[\_]$. These barbs are trivially matched.

Let us now consider reductions. All the reductions performed by the context are banally matched. Also, since all the relations are closed under the applications of auxiliary processes such as T_l or $S(l_1,l_2,l_3)$ we will not mention them. Let us consider the different relations.

In $\mathcal{R}_1$ the only possibility is that the process $S(l_1,l_2,l') = (l_1(Z)|l_2(W) \triangleright l'\langle (h)\texttt{Par}\ Z\ W\ h\rangle)$ interacts with two messages on l_1 and l_2 in the context. We can assume they are of the form $l_1\langle (\!|P|\!)\rangle$ and $l_2\langle (\!|Q|\!)\rangle$, thus the reduction leads to $\mathcal{R}_2$.

$$
\begin{aligned}
&\mathcal{R}_1 = \{(\nu l'.\, S(l_1,l_2,l') \mid S(l',l_3,l)), (\nu l'.\, S(l_1,l',l) \mid S(l_2,l_3,l')) \mid l_1,l_2,l,l' \in \mathcal{K}\}\\
&\mathcal{R}_2 = \{(\nu l'.\, l'\langle(h)\texttt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ h\rangle \mid T_{l'} \mid S(l',l_3,l)),\\
&\qquad (\nu l_1,l_2.\, \texttt{addG}(A((\!|P|\!),l_1) \mid A((\!|Q|\!),l_2) \mid \nu l'.\, S(l_1,l',l) \mid S(l_2,l_3,l'))\}\\
&\mathcal{R}_3 = \{(\nu l'.\, ((h)\texttt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ h)l' \mid T_{l'} \mid S(l',l_3,l)),\\
&\qquad (\nu l_1,l_2.\, \texttt{addG}(A((\!|P|\!),l_1) \mid A((\!|Q|\!),l_2) \mid \nu l'.\, S(l_1,l',l) \mid S(l_2,l_3,l')))\}\\
&\mathcal{R}_4 = \{(\nu l'.\, (\texttt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ l') \mid T_{l'} \mid S(l',l_3,l)),\\
&\qquad (\nu l_1,l_2.\, \texttt{addG}(A((\!|P|\!),l_1) \mid A((\!|Q|\!),l_2) \mid \nu l'.\, S(l_1,l',l) \mid S(l_2,l_3,l')))\}\\
&\mathcal{R}_5 = \{(\nu l',h_1,h_2.\, (\!|P|\!)h_1 \mid (\!|Q|\!)h_2 \mid S(h_1,h_2,l') \mid T_{l'} \mid S(l',l_3,l)),\\
&\qquad (\nu l_1,l_2.\, \texttt{addG}(A((\!|P|\!),l_1) \mid A((\!|Q|\!),l_2) \mid \nu l'.\, S(l_1,l',l) \mid S(l_2,l_3,l'))\}\\
&\mathcal{R}_6 = \{(\texttt{Par}\ (\!|P \mid Q|\!)\ (\!|R|\!)\ l), (\texttt{Par}\ (\!|P|\!)\ (\!|Q \mid R|\!)\ l)\}\\
&\mathcal{R}_7 = \{(\nu l_1,l_2.\, (\!|P \mid Q|\!)l_1 \mid (\!|R|\!)l_2 \mid S(l_1,l_2,l)),\\
&\qquad (\nu l_1,l_2.\, (\!|P|\!)l_1 \mid (\!|Q \mid R|\!)l_2 \mid S(l_1,l_2,l))\}\\
&\mathcal{R}_8 = \{(\nu l_1.\, (\!|P \mid Q|\!)l_1 \mid S(l_1,l_2,l)),\\
&\qquad (\nu l_1,l_3,l_4.\, (\!|P|\!)l_1 \mid (\!|Q|\!)l_3 \mid S(l_1,l_4,l) \mid S(l_3,l_2,l_4))\}\\
&\mathcal{R}' = \cup \mathcal{R}_i\\
&\mathcal{R} = \{(\mathbb{C}[P_1,\ldots,P_n], \mathbb{C}[Q_1,\ldots,Q_n]) \mid (P_i,Q_i) \in \mathcal{R}' \wedge\\
&\qquad \mathbb{C}[P_1,\ldots,P_n], \mathbb{C}[Q_1,\ldots,Q_n]\ \mathit{wellformed}\}
\end{aligned}
$$

Figure C.14: Relation for $Ex.A$

From $\mathcal{R}_2$ two reductions are possible: the process can interact with the context by reading a message on l_3 of the form $l_3\langle(\!|R|\!)\rangle$, or the message on l' may interact with $T_{l'}$. In the first case we obtain $l\langle(h)\texttt{Par}\ (\!|P \mid Q|\!)\ (\!|R|\!)\ h\rangle \mid T_l$. On the right side we obtain $T_l \mid l\langle(h)\texttt{Par}\ (\!|P|\!)\ (\!|Q \mid R|\!)\ h\rangle$. We can move to the context T_l, which occurs on both the sides, and also the message context, the application context and the abstraction context going to $\mathcal{R}_6$. In the second case we go directly to $\mathcal{R}_3$.

In $\mathcal{R}_3$ the only possible reduction is the application of the `Par`, leading to $\mathcal{R}_4$.

Also in $\mathcal{R}_4$ the only possible reduction is an application, leading to $\mathcal{R}_5$.

In $\mathcal{R}_5$ both $(\!|P|\!)h_1$ and $(\!|Q|\!)h_2$ may reduce by means of an application. Note that the right process weakly reduces to the left one. In fact, since contexts are well formed there exist in the term two `Rew` processes, one on l_1

and one on l_2. Hence we have that

$$\begin{aligned}&\nu l_1, l_2.\, \texttt{addG}(l_1\langle(\!|P|\!)\rangle \mid l_2\langle(\!|Q|\!)\rangle \mid (\texttt{Rew}\ l_1) \mid\\ &\qquad (\texttt{Rew}\ l_2) \mid \nu l'.\, S(l_1, l', l) \mid S(l_2, l_3, l')) \hookrightarrow^*\\ &\nu l_1, l_2.\, \texttt{addG}((\!|P|\!)l_1 \mid (\!|Q|\!)l_2 \mid \nu l'.\, S(l_1, l', l) \mid S(l_2, l_3, l'))\end{aligned}$$

and by α-converting l_1, l_2 into h_1, h_2 we obtain $\nu h_1, h_2.\, \texttt{addG}((\!|P|\!)h_1 \mid (\!|Q|\!)h_2 \mid \nu l'.\, S(h_1, l', l) \mid S(h_2, l_3, l'))$. This last process is structural congruent to $\nu h_1, h_2.\, \texttt{addG}((\!|P|\!)h_1 \mid (\!|Q|\!)h_2) \mid \nu l'.\, S(h_1, l', l) \mid S(h_2, l_3, l')$. Since each process R is weak bf barbed bisimilar to $\texttt{addG}(R)$ (by Lemma 68), we conclude by transitivity, noting that by removing the addG function we get back to the relation $\mathcal{R}_1$.

From $\mathcal{R}_6$ we can only move to $\mathcal{R}_7$ by applications.

From $\mathcal{R}_7$ there are two possible reductions, either the application of $(\!|P \mid Q|\!)l_1$ or the application of $(\!|R|\!)l_2$. In the first case we get back to the relation $\mathcal{R}_1$ (by using α-conversion). In the second case we can execute on the right the application $(\!|Q \mid R|\!)l_2$, and obtain a process of the form $(\!|R|\!)l_2$ (using α-conversion) also on the right. We can thus move these processes to the context, going to the relation $\mathcal{R}_8$.

From $\mathcal{R}_8$, the left process can only perform the application on $(\!|P \mid Q|\!)$, and we get back to the relation $\mathcal{R}_1$ (the right process matches this challenge by a 0 steps computation). □

Lemma 70. *Axiom Ex.Unfold and its normal form are correct with respect to weak bf barbed bisimulation.*

Proof. Let T_l denote a process of the form $T_l = \texttt{Rew}\ l$ or $T_l = (l(Z) \triangleright Z\ l)$. Let

$$\begin{aligned}&\mathcal{R}' = \{((\!|P|\!)l, \nu\tilde{u}.\, l\langle(\!|Q|\!)\rangle \mid T_l) \mid P \equiv \nu\tilde{u}.\, Q\} \cup \{((\!|P|\!)l, \nu\tilde{u}.\, (\!|Q|\!)l) \mid P \equiv \nu\tilde{u}.\, Q\}\\ &\mathcal{R} = \{(\mathbb{C}[P_1, .., P_n]\ ,\ \mathbb{C}[Q_1, .., Q_n]) \mid (P_i, Q_i) \in \mathcal{R}'\}\end{aligned}$$

We now show that the relation $\mathcal{R}$ is a weak bf barbed bisimulation.

Let us consider barbs. Since $(\!|P|\!)l$ is an application then the only barbs shown by a process of the form $\mathbb{C}[P_1, .., P_n]$ are those shown by the context. The same barbs are shown also by the process $\mathbb{C}[Q_1, .., Q_n]$. On the other hand, a process of the form $l\langle(\!|Q|\!)\rangle$ shows a barb at l. By using Lemma 23 we have that $(\!|P|\!)l \hookrightarrow^* \nu\tilde{v}.\, l\langle(\!|Q|\!)\rangle \mid \texttt{Rew}\ l \mid S$, with S a parallel composition of garbage processes and $l \notin \tilde{v}$. Thus, $(\!|P|\!)l$ has a weak barb at l too.

Let us consider reductions. On both sides, reductions done by the context are trivially matched. The only possible reduction for a hole process of the form $(\!|P|\!)l$ is an application. Then, we have a case analysis on the form of P. If $P = \nu a.\, P'$, then $\mathbb{C}[P_1, .., (\!|\nu a.\, P'|\!)l, .., P_n] \rightharpoondown^* \mathbb{C}[P_1, .., \nu a.\, (\!|P'|\!)l, .., P_n]$, and this step is mimicked by the process $\mathbb{C}[Q_1, .., \nu\tilde{u}.\, l\langle(\!|Q|\!)\rangle, .., Q_n]$ by staying idle, since we can extract name a from the set $\tilde{u}$. If $P = P' \mid P''$, we have that $P' \mid P'' \equiv \nu\tilde{u}.\, Q$ implies $\nu\tilde{u}.\, Q \equiv \nu\tilde{u}_1.\, Q' \mid \nu\tilde{u}_2.\, Q''$. Hence

$$\begin{aligned}&\mathbb{C}[P_1, .., (\!|P' \mid P''|\!)l, .., P_n] \rightharpoondown\\ &\mathbb{C}[P_1, .., \nu l_1, l_2.\, (\!|P'|\!)l_1 \mid (\!|P''|\!)l_2 \mid (\texttt{KillP}\ l_1\ l_2\ l), .., P_n]\end{aligned}$$

This step can be matched by

$$\begin{aligned}&\mathbb{C}[Q_1, .., \nu\tilde{u}.\, l\langle(\!|Q' \mid Q''|\!)\rangle \mid \texttt{Rew}\ l, .., Q_n] \rightharpoondown\\ &\mathbb{C}[Q_1, .., \nu\tilde{u}.\, l\langle(\!|Q' \mid Q''|\!)\rangle \mid (l(Z) \triangleright Z\ l), .., Q_n] \to\\ &\mathbb{C}[Q_1, .., \nu l_1, l_2.\, \nu\tilde{u}_1.\, (\!|Q'|\!)l_1 \mid \nu\tilde{u}_2.\, (\!|Q''|\!)l_2 \mid (\texttt{KillP}\ l_1\ l_2\ l), .., Q_n]\end{aligned}$$

and we are still in the same relation since:

$$\begin{aligned}&((\!|P'|\!), \nu\tilde{u}_1.\, (\!|Q'|\!)) \in \mathcal{R}'\\ &((\!|P''|\!), \nu\tilde{u}_2.\, (\!|Q''|\!)) \in \mathcal{R}'\end{aligned}$$

The other cases are similar.

If the step is performed by the process on the right, we have three cases. The application of a process $\texttt{Rew}\ l$ can be matched by the left process by staying idle. An interaction between $(l(Z) \triangleright Z\ l)$ and a message on l is matched since $\mathbb{C}[Q_1, .., \nu\tilde{u}.\, l\langle(\!|Q|\!)\rangle \mid l(Z) \triangleright Z\ l, .., Q_n] \to \mathbb{C}[Q_1, .., \nu\tilde{u}.\, (\!|Q|\!)l, .., Q_n]$ and we are still in the same relation. For applications of $(\!|Q|\!)l$ we proceed by case analysis as before and we remain in the same relation, as desired. □

Lemma 71. *Axiom Ex.Adm and its normal form are correct with respect to weak bf barbed bisimulation.*

Proof. Let

$$\begin{aligned}\mathcal{R}' &= \{(\nu c.\, (c\langle P\rangle \mid c(Z) \triangleright Z\ k), (\!|P|\!)k)\}\\ \mathcal{R} &= \{(\mathbb{C}[P_1, .., P_n], \mathbb{C}[Q_1, .., Q_n]) \mid (P_i, Q_i) \in \mathcal{R}'\}\end{aligned}$$

We now show that the relation $\mathcal{R}$ is a weak bf barbed bisimulation.

All the challenges from the right process are easily matched, since the left process reduces to the right one via an administrative reduction.

The only barbs of the left term are in the context, thus they are easily matched. Similarly, reductions of the context are easily matched. Reductions in the hole reduce to the term on the right, thus we are in the same relation by removing one hole. □

Proposition 1. *The relation* $\mathcal{R} = \{(P, Q) \mid P \equiv_{Ex} Q\}$ *where* P, Q *are* $HO\pi^+$ *processes is a weak bf bisimulation.*

Proof. By definition $P \equiv_{Ex} Q$ iff there are $P_1, \dots P_n$ such that $P \equiv_{Ex} P_1 \equiv_{Ex} \dots \equiv_{Ex} P_n \equiv_{Ex} Q$ where each equivalence is obtained by applying just one axiom. The proof is by induction on n. The base case is banally verified. In the inductive case we proceed by case analysis on the last applied axiom.

Let us consider axiom $Ex.C$. By inductive hypothesis we have that $P \equiv_{Ex} P_n$ implies that $P \dot{\approx}_c P_n$ and that $P_n \equiv_{Ex} Q$ using the axiom $Ex.C$. By Lemma 63 we know that $P_n \equiv_{Ex} Q$ implies $P_n \dot{\approx}_c Q$, and by transitivity we have that also $P \dot{\approx}_c Q$.

The other cases are analogous, using Lemma 65 for axiom $Ex.P$, Lemma 69 for axiom $Ex.A$, Lemma 70 for axiom $Ex.Unfold$, Lemma 71 for axiom $Ex.Adm$ and Lemma 32 for axioms in $\equiv_\pi$. □

Appendix C.4. Proofs of Section 4.6

Lemma 33. *If* $\mathtt{nf}(\llparenthesis M \rrparenthesis) \twoheadrightarrow P$ *then* $M \twoheadrightarrow M'$ *with* $P \hookrightarrow^* P'$ *and* $\mathtt{nf}(P') \equiv \mathtt{nf}(Q')$ *and* $Q' \in \mathtt{addG}(\llparenthesis M' \rrparenthesis)$*.*

Proof. By structural induction on M. If M is a simple process such as $\mathbf{0}$, a message or a trigger there is nothing to verify since $\mathtt{nf}(\llparenthesis M \rrparenthesis) \not\twoheadrightarrow$.

In the inductive case, if M is of the form $\nu a.\, M_1$ then by definition of $\mathtt{nf}(\cdot)$ and $\llparenthesis \cdot \rrparenthesis$ we have that $\mathtt{nf}(\llparenthesis M \rrparenthesis) = \nu a.\, \mathtt{nf}(\llparenthesis M_1 \rrparenthesis)$ and by applying the inductive hypothesis on $\mathtt{nf}(\llparenthesis M_1 \rrparenthesis)$ we have that $\mathtt{nf}(\llparenthesis M_1 \rrparenthesis) \twoheadrightarrow P$ implies $M_1 \twoheadrightarrow M_1'$ with $P \hookrightarrow^* P'$ and $\mathtt{nf}(P') = \mathtt{nf}(Q')$ with $Q' \in \mathtt{addG}(\llparenthesis M_1' \rrparenthesis)$. If $\nu a.\, \llparenthesis M_1 \rrparenthesis \twoheadrightarrow \nu a.\, P$ then also $\llparenthesis M_1 \rrparenthesis \twoheadrightarrow P$ and using inductive hypothesis we obtain $\nu a.\, M_1 \twoheadrightarrow \nu a.\, M_1'$ with $\nu a.\, P \hookrightarrow^* \nu a.\, P'$ and since $\mathtt{nf}(P') = \mathtt{nf}(Q')$ with $Q' \in \mathtt{addG}(\llparenthesis M_1' \rrparenthesis)$ we also have that $\nu a.\, \mathtt{nf}(P') = \nu a.\, \mathtt{nf}(Q')$ that is $\mathtt{nf}(\nu a.\, P') = \mathtt{nf}(Q'')$ with $Q'' \in \mathtt{addG}(\llparenthesis \nu a.\, M_1' \rrparenthesis)$, as desired. The case of parallel context $M = M_1 \mid M_2$ also follows by inductive hypothesis if the reduction is done inside either M_1 or M_2. If both M_1 and M_2 contribute to the reduction then we can assume that there is a message in M_1 and a trigger in M_2 able to communicate. For the sake of brevity, we consider just the case in which both message and trigger are tagged by a key. The other cases are similar. We can write $M = M_1' \mid k_1 : a\langle R\rangle \mid k_2 : a(X) \triangleright Q \mid M_2'$. Hence, we have

$$\begin{aligned}&(\!|M_1' \mid k_1 : a\langle R\rangle \mid k_2 : a(X) \triangleright Q \mid M_2'|\!) = \\ &(\!|M_1'|\!) \mid (\!|a\langle R\rangle|\!)_{k_1} \mid (\!|a(X) \triangleright Q|\!)_{k_2} \mid (\!|M_2'|\!)\end{aligned}$$

Let $Y = (X\ c)c\langle(\!|Q|\!)\rangle$, then we have that

$$\begin{aligned}&\texttt{nf}((\!|M|\!)) = \texttt{nf}((\!|M_1'|\!)) \mid a\langle(\!|R|\!), k_1\rangle \mid \texttt{nf}(\texttt{KillM}\ a\ k_1) \mid \\ &\qquad \nu\texttt{t}.\,\texttt{t}|(a(X,l)|\texttt{t} \triangleright_f \nu k, c.\,(Y\ X\ x) \mid (c(Z) \triangleright Z\ k) \mid (\texttt{Mem}\ Y\ a\ X\ l\ k\ k_2)) \mid \\ &\qquad \texttt{nf}(\texttt{KillT}\ Y\ \texttt{t}\ k_2\ a) \mid \texttt{nf}((\!|M_2'|\!)) \twoheadrightarrow \\ &\texttt{nf}((\!|M_1'|\!)) \mid \texttt{nf}(\texttt{KillM}\ a\ k_1) \mid \nu k, c, \texttt{t}.\,(Y\ (\!|R|\!)\ c) \mid (c(Z) \triangleright Z\ k) \mid \\ &(\texttt{Mem}\ Y\ a\ X\ l\ k\ k_2) \mid \texttt{nf}(\texttt{KillT}\ Y\ \texttt{t}\ k_2\ a) \mid \texttt{nf}((\!|M_2'|\!)) \hookrightarrow\hookrightarrow \\ &\texttt{nf}((\!|M_1'|\!)) \mid \texttt{nf}(\texttt{KillM}\ a\ k_1) \mid \nu k, c, \texttt{t}.\,(\!|Q|\!)\{^{(\!|R|\!)}/_X\}k \mid (\texttt{Mem}\ Y\ a\ (\!|R|\!)\ k_1\ k\ k_2) \mid \\ &\texttt{nf}(\texttt{KillT}\ Y\ \texttt{t}\ k_2\ a) \mid \texttt{nf}((\!|M_2'|\!))\end{aligned}$$

By using Lemma 21 (Substitution Lemma) the process above is equal to:

$$\begin{aligned}&\texttt{nf}((\!|M_1'|\!)) \mid \texttt{nf}(\texttt{KillM}\ a\ k_1) \mid \nu k, c, \texttt{t}.\,(\!|Q\{^R/_X\}|\!)k \mid (\texttt{Mem}\ Y\ a\ (\!|R|\!)\ k_1\ k\ k_2) \mid \\ &\texttt{nf}(\texttt{KillT}\ Y\ \texttt{t}\ k_2\ a) \mid \texttt{nf}((\!|M_2'|\!)) = P\end{aligned}$$

We have that $M \twoheadrightarrow M'$ with $M' = M_1' \mid \nu k.\, k : Q\{^R/_X\} \mid M_2'$ and we can easily see that $P \rightharpoondown^* \texttt{nf}((\!|M_1'|\!)) \mid \texttt{nf}(\texttt{KillM}\ a\ k_1) \mid \nu k, c, \texttt{t}.\,(\!|Q\{^R/_X\}|\!)k$ $\mid \texttt{nf}(\texttt{Mem}\ Y\ a\ (\!|R|\!)\ k_1\ k\ k_2) \mid \texttt{nf}(\texttt{KillT}\ Y\ \texttt{t}\ k_2\ a) \mid \texttt{nf}((\!|M_2'|\!)) \equiv \texttt{nf}(Q'')$ with $Q'' \in \texttt{addG}((\!|M'|\!))$, as desired. $\square$