

Reversing Higher-Order Pi

Ivan Lanese¹, Claudio Antares Mezzina², and Jean-Bernard Stefani²

¹ Focus Team, University of Bologna/INRIA, Italy

² INRIA Grenoble-Rhône-Alpes, 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 ($\text{HO}\pi$). We prove that reversibility in our calculus is causally consistent and that one can encode faithfully reversible $\text{HO}\pi$ into a variant of $\text{HO}\pi$.

1 Introduction

Motivation and contributions. The notion of reversible computation already has a long history [2]. Nowadays, it is attracting increasing interest because of its applications in diverse fields, including hardware design, biological modelling, program debugging and testing, and quantum computing. Of particular interest is its application to the study of programming abstractions for reliable systems. The work of Danos and Krivine on reversible CCS (RCCS) [5,6] provides a good example: they show how notions of reversible and irreversible actions in a process calculus can model a primitive form of transaction, an abstraction that has been found useful, in different guises, in building reliable distributed systems, including database and workflow management systems.

In this paper, we continue the study undertaken by Danos and Krivine by tackling two questions which were already anticipated in the conclusion of [5]: (i) how can we introduce reversible actions in a higher-order concurrent calculus, specifically, the asynchronous higher-order π -calculus ($\text{HO}\pi$)? (ii) does the introduction of reversible actions augment the expressive power of $\text{HO}\pi$? The first question finds its motivation in the pursuit of a suitable programming model for the construction of reliable and adaptive systems, and hence in the need to study the combination of reliable programming abstractions with modular dynamicity constructs enabling dynamic software update and on-line reconfiguration. The second question is motivated by language design issues: understanding which primitives bring expressive power and which do not, allows a language designer to decide which primitives to keep in a core language definition, and which to provide as derived abstractions or as part of a language library.

In response to the first question, we define a reversible variant of the higher-order π -calculus ($\text{HO}\pi$) [11]. A general method for reversing process calculi has been proposed by Phillips and Ulidowski in [10]. Unfortunately, it is only given

for calculi whose operational semantics can be defined using SOS rules conforming to the *path* format, which is not the case for HO π [9]. We therefore adopt an approach inspired by that of Danos and Krivine, but with significant differences. In particular, in their RCCS approach, the usual congruence laws associated with the parallel operator do not hold. Our first contribution is thus a simple syntax and reduction semantics for a reversible HO π calculus called $\rho\pi$, with a novel way to define reversible actions while preserving the usual structural congruence laws of HO π , notably the associativity and commutativity of the parallel operator.

Concerning the second question, we settle it in the case of $\rho\pi$ by showing that, surprisingly enough, reversibility can be obtained essentially as syntactic sugar on top of HO π . More precisely, our second contribution is a faithful compositional encoding of the $\rho\pi$ calculus into a variant of HO π .

Outline. The paper is organized as follows. Section 2 defines the $\rho\pi$ calculus. We explain the main constructions of the calculus and we contrast our way of handling reversibility with that of Danos and Krivine. Section 3 is devoted to the proof of our first main result, namely that reverse or *backward* computations in $\rho\pi$ are causally consistent, i.e. that they proceed through configurations that are causally equivalent with configurations arising from normal or *forward* computations. Section 4 presents a compositional encoding of the $\rho\pi$ calculus into a variant of HO π . We prove that the translation is faithful, i.e. that a $\rho\pi$ process and its encoding are weakly barbed bisimilar. Section 5 concludes the paper with a discussion of related work.

2 The $\rho\pi$ Calculus

2.1 Informal Presentation

Building a reversible variant of a process calculus involves devising appropriate syntactic representations for computation histories. In general, since a process calculus is not confluent and processes are non-deterministic, reversing a (forward) computation history means undoing the history not in a deterministic way but in a *causally consistent* fashion, 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 to evolve into P'_1 and P'_2 , respectively, then the parallel composition of monitored processes $m_1 : (P_1 + Q_1)$ and $m_2 : (P_2 + Q_2)$ can evolve as follows:

$$m_1 : (P_1 + Q_1) \mid m_2 : (P_2 + Q_2) \rightarrow \langle m_2, a, Q_1 \rangle \cdot m_1 : P'_1 \mid \langle m_1, a, Q_2 \rangle \cdot m_2 : P'_2$$

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 of the form $R_1 + R_2$, gets its own unique identity. Unfortunately, this rule 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 below).

We adopt for $\rho\pi$ 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 $\rho\pi$ (noted with arrow \rightarrow) consists in the receipt of a message ($\rho\pi$ 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 $\rho\pi$ the following form:

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

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 ($\rho\pi$ is a higher-order calculus). The two side effects are: (i) the tagging of the newly created process $Q\{P/x\}$ by a fresh name k (operator 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 name: when a memory $[M; k]$ is put in presence of a process tagged with k , a *backward* reduction (noted with the arrow \rightsquigarrow) can take place which kills the process tagged with k and reinstates the configuration M :

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

We thus have:

$$M \rightarrow \nu k. k : Q\{P/x\} \mid [M; k] \rightsquigarrow \nu k. M$$

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

Remark 1. Following Danos and Krivine [6], 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 $\rho\pi$ 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 $\rho\pi$. We do not consider implementation and related space efficiency issues in this paper.

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

Fig. 1. Syntax of $\rho\pi$

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*. We note \mathbb{N} 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 note \tilde{u} a finite set of identifiers $\{u_1, \dots, u_n\}$.

Syntax. The syntax of the $\rho\pi$ calculus is given in Figure 1 (in writing $\rho\pi$ terms, we freely add balanced parenthesis around terms to disambiguate them). *Processes* of the $\rho\pi$ calculus, given by the P, Q productions in Figure 1, are the standard processes of the asynchronous higher-order π -calculus. A receiver process (or *trigger*) in $\rho\pi$ 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 $\rho\pi$ cannot directly execute, only *configurations* can. *Configurations* in $\rho\pi$ 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.

A *memory* is a process of the form $[\mu; k]$, which keeps track of the fact that a configuration M 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 involved in the message receipt $a(X) \triangleright Q$, together with their respective thread tags κ_1, κ_2 .

We note \mathcal{P} the set of $\rho\pi$ processes, and \mathcal{C} the set of $\rho\pi$ 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 $\rho\pi$: these can be easily encoded in $\rho\pi$ (in fact in asynchronous HO π).

Free names and free variables. Notions of free identifiers and free (process) variables in $\rho\pi$ 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 note $\text{fn}(P)$, $\text{fn}(M)$ and $\text{fn}(\kappa)$ 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 $\text{fn}(\kappa : P) = \text{fn}(\kappa) \cup \text{fn}(P)$, $\text{fn}(k) = \{k\}$ and $\text{fn}(\langle h, \tilde{h} \rangle \cdot k) = \tilde{h} \cup \{k\}$. We say that a process P or a configuration M is closed if it has no free (process) variable. We note \mathcal{P}^\bullet 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, \dots, 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 $[M; k]$, tags occurring in the configuration part M 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 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 $[N; k]$ (i.e. $[N; k]$ 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 $[N; k]$ 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 static semantic constraints. We defer the formal definition of consistent configurations to Section 2.3.

2.3 Operational Semantics

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

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

General contexts \mathbb{C} are just processes or configurations with a hole. 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 $M \mathcal{R} N \implies \mathbb{C}[M] \mathcal{R} \mathbb{C}[N]$.

$$\begin{array}{lll}
(\text{E.PARC}) \ A \mid B \equiv B \mid A & (\text{E.PARA}) \ A \mid (B \mid C) \equiv (A \mid B) \mid C \\
(\text{E.NILM}) \ A \mid \mathbf{0} \equiv A & (\text{E.NEWN}) \ \nu u. \mathbf{0} \equiv \mathbf{0} & (\text{E.NEWC}) \ \nu u. \nu v. A \equiv \nu v. \nu u. A \\
(\text{E.NEWP}) \ (\nu u. A) \mid B \equiv \nu u. (A \mid B) & & (\text{E.}\alpha) \ A =_{\alpha} B \implies A \equiv B \\
& (\text{E.TAGN}) \ \kappa : \nu a. P \equiv \nu a. \kappa : P \\
& (\text{E.TAGP}) \ k : \prod_{i=1}^n \tau_i \equiv \nu \tilde{h}. \prod_{i=1}^n (\langle h_i, \tilde{h} \rangle \cdot k : \tau_i) \quad \tilde{h} = \{h_1, \dots, h_n\}
\end{array}$$

Fig. 2. Structural congruence for $\rho\pi$

The relation \equiv is defined as the smallest congruence on processes and configurations that satisfies the rules in Figure 2. We note $t =_{\alpha} t'$ when terms t, t' are equal modulo α -conversion. If $\tilde{u} = \{u_1, \dots, u_n\}$, then $\nu \tilde{u}. A$ stands for $\nu u_1. \dots. \nu u_n. A$. We note $\prod_{i=1}^n A_i$ for $A_1 \mid \dots \mid A_n$ (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. 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 $u \notin \text{fn}(B)$. 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 configuration M , we have*

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

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

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

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

Unfortunately using E.TAGP $^{\bullet}$ 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 $M = k : a(Q) \mid (h : a(X) \triangleright X)$. We have: $M \rightarrow M' = \nu l. (l : Q) \mid [M; l]$ Now, assuming E.TAGP $^{\bullet}$, we would have

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

$$\begin{aligned}
 (\text{R.Fw}) \quad & (\kappa_1 : a\langle P \rangle) \mid (\kappa_2 : a(X) \triangleright Q) \rightarrow\!\!\! \rightarrow \nu k. (k : Q\{^P/x\}) \mid [(\kappa_1 : a\langle P \rangle) \mid (\kappa_2 : a(X) \triangleright Q); k] \\
 (\text{R.Bw}) \quad & (k : P) \mid [M; k] \rightsquigarrow M
 \end{aligned}$$

Fig. 3. Reduction rules for $\rho\pi$

Let $M_1 = (h_1 \cdot k : a\langle Q \rangle) \mid (h : a(X) \triangleright X)$. We would then have: $M \rightarrow M''$, 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 $\rho\pi$, which fails if we replace rule E.TAGP with rule E.TAGP $^\bullet$. Extend the grammar of $\rho\pi$ with marked primitive thread processes of the form τ^\bullet . This extended calculus has exactly the same structural congruence and reduction rules than $\rho\pi$, but with possibly marked primitive thread processes. Now call *primed* a closed configuration M with exactly two marked processes of the form $a\langle P \rangle^\bullet$ and $(a(X) \triangleright Q)^\bullet$. Anticipating on the definition of the reduction relation \rightarrow below (with \equiv and \rightarrow trivially extended to marked processes), we have:

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

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

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

The reduction relation \rightarrow is defined as the union of two relations, the *forward* reduction relation $\rightarrow\!\!\! \rightarrow$ and the *backward* reduction relation \rightsquigarrow : $\rightarrow = \rightarrow\!\!\! \rightarrow \cup \rightsquigarrow$. Relations $\rightarrow\!\!\! \rightarrow$ 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 can now formally define the notion of *consistent configuration*.

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

1. For all $j \in J$, $k_j \neq \delta_j$, $k_j \neq \gamma_j$ and $\delta_j \neq \gamma_j$
2. For all $i_1, i_2 \in I$, $i_1 \neq i_2 \implies \kappa_{i_1} \neq \kappa_{i_2}$
3. For all $i \in I, j \in J$, $\kappa_i \neq \delta_j$ and $\kappa_i \neq \gamma_j$
4. For all $j \in J$, there exist $E \subseteq I$, $D \subseteq J$, $G \subseteq J$, such that:

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

Consistent configurations are preserved by reduction:

Lemma 3. Let M be a consistent configuration. If $M \rightarrow N$ then N is a consistent configuration.

Proof. By case analysis on the derivation of $M_t \rightarrow N$, where $M \equiv M_t$ and M_t is in thread normal form. \square

Barbed bisimulation. The operational semantics of the $\rho\pi$ calculus is completed classically by 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 note \Rightarrow the reflexive and transitive closure of \rightarrow . The following definitions are classical:

Definition 2 (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. $N \Rightarrow \downarrow_a$)
- $M \rightarrow M'$ implies $N \rightarrow N'$, 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 note \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 noted \sim (resp. \approx).

2.4 Basic Properties of Reduction in $\rho\pi$

Define inductively the *erasing function* $\gamma : \mathcal{C} \rightarrow \mathcal{P}$, which maps a configuration on its underlying HO π process, by the following clauses:

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

The following lemma shows that $\rho\pi$ forward computations are indeed decorations on HO π reductions.

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

Proof. Straightforward, by first proving by induction on the derivation of $M \equiv N$ that $M \equiv N \implies \gamma(M) \equiv \gamma(N)$, and then by reasoning by induction on the derivation of $M \twoheadrightarrow N$. \square

Remark 6. One can lift a closed HO π process P to a closed consistent configuration in $\rho\pi$ by defining $\delta(P) = \nu k. k : P$.

The next lemma shows that forward and backward reductions in $\rho\pi$ are really inverse of one another.

Lemma 5 (Loop lemma). *For all closed consistent configurations M, N , $M \twoheadrightarrow N \iff N \rightsquigarrow M$.*

Proof. By induction on the derivation of $M \twoheadrightarrow N$ for the *if* direction, and on the derivation of $N \rightsquigarrow M$ for the converse. \square

A direct consequence of the Loop Lemma is that a closed consistent configuration M is weakly barbed congruent to any of its descendants, a fact that one can understand as a kind of observational property of reversibility:

Lemma 6 (Observational equivalence). *If $M \Rightarrow N$, then $M \approx N$.*

Proof. By induction on the form of configuration contexts, the base case $M \approx N$ being obtained by applying the Loop Lemma. \square

3 Causality in $\rho\pi$

We now proceed to the analysis of causality in $\rho\pi$, showing that reversibility in $\rho\pi$ is causally consistent. We mostly adapt for the exposition the terminology and arguments of [5].

We call *transition* a triplet of the form $M \xrightarrow{m} M'$, or $M \xrightarrow{m\bullet} M'$, where M, M' are closed consistent configurations, $M \rightarrow M'$, and m is the memory involved in the reduction $M \rightarrow M'$. We say that a memory m is *involved* in a 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 noted $M \xrightarrow{m} M'$. Likewise, we say that a memory $m = [N; k]$ is involved in a 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 noted $M \xrightarrow{m\bullet} M'$.

We say a transition $t : M \xrightarrow{m} M'$ is *name-preserving* if M and M' are in thread normal form and if either (i) $M = \nu \tilde{u}. \prod_{i \in I} (\kappa_i : \rho_i) \mid \prod_{j \in J} [M_j : k_j]$,

$M' = \nu\tilde{u}. \prod_{i \in I'} (\kappa_i : \rho_i) \mid \prod_{j \in J'} [M_j : k_j]$, with $I' \subseteq I, J \subseteq J'$, and $m = [N_j; k_j]$ for some $j \in J'$; or (ii) $M = \nu\tilde{u}. \prod_{i \in I} (\kappa_i : \rho_i) \mid \prod_{j \in J} [M_j : k_j]$, $M' = \nu\tilde{u}. \prod_{i \in I'} (\kappa_i : \rho_i) \mid \prod_{j \in J'} [M_j : k_j]$, with $I \subseteq I', J' \subseteq J$, and $m = [M_j; k_j]$ for some $j \in J$. Intuitively, a name-preserving transition keeps track of the set of restricted names of its configurations, all the names used in the transition (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).

In a transition $M \xrightarrow{\mu} N$, we say that M is the *source* or the transition, N is its *target*, and μ is its label (of the form m or m_\bullet , where m is some memory – we let μ and its decorated variants range over transition labels). If $\mu = m_\bullet$, we set $\mu_\bullet = m$. 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 compositability 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* $\lambda(m)$ of a memory $m = [\kappa_1 : a(P) \mid \kappa_2 : a(X) \triangleright Q; k]$ is defined to be the set $\{\kappa_1, \kappa_2, k\}$; we set $\lambda(m_\bullet) = \lambda(m)$.

Definition 3 (Concurrent transitions). Two coinitial transitions $t_1 = M \xrightarrow{\mu_1} M_1$ and $t_2 = M \xrightarrow{\mu_2} M_2$ are said to be concurrent if $\lambda(\mu_1) \cap \lambda(\mu_2) = \emptyset$.

Remark 7. Note that the stamp of a memory $[M; k]$ include its tag k . This is necessary to take into account possible conflicts between a forward action and a backward action.

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

Lemma 7 (Square lemma). If $t_1 = M \xrightarrow{\mu_1} M_1$ and $t_2 = M \xrightarrow{\mu_2} M_2$ are two coinitial concurrent transitions, then there exist two cofinal transitions $t_2/t_1 = M_1 \xrightarrow{\mu_2} N$ and $t_1/t_2 = M_2 \xrightarrow{\mu_1} N$.

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

We are now in a position to show that reversibility in $\rho\pi$ is causally consistent. We define first the notion of causal equivalence between traces, noted \asymp , as the least equivalence relation between traces closed under composition that obeys the following rules:

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

The proof of causal consistency proceeds along the exact same lines as in [5], with simpler arguments because of the simpler form of our memory stamps.

Lemma 8 (Rearranging Lemma). Let σ be a trace. There exists forward traces σ' and σ'' such that $\sigma \asymp \sigma'_\bullet; \sigma''$.

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

Fig. 4. Syntax of HO π^+

Lemma 9 (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 $\sigma'_1 \asymp \sigma_1$.*

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.*

4 Encoding $\rho\pi$ in HO π^+

We show in this section that $\rho\pi$ can be encoded in a variant of HO π , with bi-adic channels, join patterns, sub-addressing, abstractions and applications, which we call HO π^+ . This particular variant was chosen for convenience, because it simplifies our encoding. All HO π^+ constructs are well understood in terms of expressive power with respect to HO π and π [12,8,4]. The syntax of HO π^+ is given in Figure 4. Channels in HO π^+ carry both a name and an abstraction (bi-adicity); a trigger can receive a message on a given channel, or on a given channel provided the received message carries some given name (sub-addressing). HO π^+ has 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 HO π^+ to be the set $\mathcal{I} \cup \{\star\}$ of $\rho\pi$. The set of (process) variables of HO π^+ is taken to coincide with the set \mathcal{V} of variables of $\rho\pi$.

The structural congruence for HO π^+ , also noted \equiv , obeys the same rules than those of $\rho\pi$, minus the rules E.TAGN and E.TAGP, which are specific to $\rho\pi$. Evaluation contexts in HO π^+ are given by the following grammar:

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

The reduction relation for HO π^+ , also noted \rightarrow , is defined as the least evaluation closed relation (same definition as for $\rho\pi$, with HO π^+ processes instead of configurations) that satisfies the rules in Figure 5. The function `match` in Figure 5 is the partial function which is defined in the cases given by the clauses below, and undefined otherwise:

$$\text{match}(u, v) = \{^u/_v\} \quad \text{match}(u, \setminus u) = \{^u/_u\} \quad \text{match}(F, X) = \{^F/_X\}$$

$$\begin{aligned}
(\text{RED}) \prod_{i=1}^n u_i \langle F_i, v_i \rangle \mid (\prod_{i=1}^n u_i(X_i, \psi_i) \triangleright P) \rightarrow P \{^{F_1 \dots F_n / X_1 \dots X_n} \theta_1 \dots \theta_n \quad \text{match}(v_i, \psi_i) = \theta_i \\
(\text{APP}) ((\psi)F V) \rightarrow F\theta \quad \text{match}(V, \psi) = \theta
\end{aligned}$$

Fig. 5. Reduction rules for HO π^+

$$\begin{array}{ll}
\langle \mathbf{0} \rangle = \text{Nil} & \langle X \rangle = (l)(X l) \\
\langle a(P) \rangle = (l)(\text{Msg } a \langle P \rangle l) & \langle \nu a. P \rangle = (l)\nu a. \langle P \rangle l \\
\langle P \mid Q \rangle = (l)(\text{Par } \langle P \rangle \langle Q \rangle l) & \langle a(X) \triangleright P \rangle = (l)(\text{Trig}_{\langle P \rangle} a l)
\end{array}$$

$$\begin{array}{l}
\text{Nil} = (l)l\langle \text{Nil} \rangle \\
\text{Msg} = (a X l)a\langle X, l \rangle \mid (\text{KillM } a X l) \\
\text{KillM} = (a X l)(a(X, \backslash l) \triangleright l \langle (h)\text{Msg } a X h \rangle \mid \text{Rew } l) \\
\text{Par} = (X Y l)\nu h, k. X h \mid Y k \mid (\text{KillP } h k l) \\
\text{KillP} = (h k l)(h(W) \mid k(Z) \triangleright l \langle (l)\text{Par } W Z l \rangle \mid \text{Rew } l) \\
\text{Trig}_{\langle P \rangle} = (a l)\nu t. t \mid (a(X, h) \mid t \triangleright \nu k. \langle P \rangle k \mid (\text{Mem}_{\langle P \rangle} a X h k l)) \mid (\text{KillT}_{\langle P \rangle} t l a) \\
\text{KillT}_{\langle P \rangle} = (t l a)(t \triangleright l \langle (h)\text{Trig}_{\langle P \rangle} a h \rangle \mid \text{Rew } l) \\
\text{Mem}_{\langle P \rangle} = (a X h k l)k(Z) \triangleright (\text{Msg } a X h) \mid (\text{Trig}_{\langle P \rangle} a l) \\
\text{Rew} = (l)(l(Z) \triangleright Z l)
\end{array}$$

Fig. 6. Encoding $\rho\pi$ processes

Conventions. In writing HO π^+ terms, $u(v)$ abbreviates $u\langle (X)\mathbf{0}, v \rangle$ and $u(F)$ abbreviates $u\langle F, \star \rangle$. Likewise, $a(u) \triangleright P$ abbreviates $a(X, u) \triangleright P$, where $X \notin \text{fv}(P)$, and $u(X) \triangleright P$ abbreviates $a(X, \star) \triangleright P$. We adopt the usual conventions for writing applications and abstractions: $(F V_1 \dots V_n)$ stands for $((F V_1) \dots) V_n$, and $(X_1 \dots X_n)F$ stands for $(X_1) \dots (X_n)F$. When there is no potential ambiguity, we often write FV for $(F V)$. When defining HO π^+ processes, we freely use recursive definitions for these can be encoded using e.g. the Turing fixed point combinator Θ defined as $\Theta = (\mathbf{A} \mathbf{A})$, where $\mathbf{A} = (X F)(F (X X F))$ (cf. [1] p.132).

The encoding $\langle \cdot \rangle : \mathcal{P}_{\rho\pi} \rightarrow \mathcal{P}_{\text{HO}\pi^+}$ of processes of $\rho\pi$ in HO π^+ is defined inductively in Figure 6. It extends to an encoding $\langle \cdot \rangle : \mathcal{C}_{\rho\pi} \rightarrow \mathcal{P}_{\text{HO}\pi^+}$ of configurations of $\rho\pi$ in HO π^+ as given in Figure 7 (note that the encoding for $\mathbf{0}$ in Figure 7 is the encoding for the null configuration). The main idea behind the encoding is simple: a tagged process $l : P$ is interpreted as a process equipped with a special channel l on which to report that it has successfully rolled back. This intuition leads to the encoding of a $\rho\pi$ process as an abstraction which takes this reporting channel l as a parameter. Rolling back a message is simply consuming it and sending it (actually the message encoding itself) on the report channel. Rolling back a trigger is just consuming the special token t that locks it and sending the trigger encoding on the report channel. Rolling back a parallel composition is just rolling back its branches and reporting when all have rolled back. The last

$$\begin{aligned}
\langle \mathbf{0} \rangle &= \mathbf{0} \\
\langle M \mid N \rangle &= \langle M \rangle \mid \langle N \rangle \\
\langle \nu u. M \rangle &= \nu u. \langle M \rangle \\
\langle k : P \rangle &= (\langle P \rangle k) \\
\langle \langle h_i, \tilde{h} \rangle \cdot k : P \rangle &= (\langle P \rangle h_i) \mid \text{Kill}_{\langle h_i, \tilde{h} \rangle \cdot k} \\
\langle [\kappa_1 : a(P) \mid \kappa_2 : a(X) \triangleright Q; k] \rangle &= (\text{Mem}_{(Q)} a \langle P \rangle \langle \kappa_1 \rangle k \langle \kappa_2 \rangle) \mid \text{Kill}_{\kappa_1} \mid \text{Kill}_{\kappa_2} \\
\langle k \rangle &= k \\
\langle \langle h_i, \tilde{h} \rangle \cdot k \rangle &= h_i \\
\text{Kill}_{\langle h_1, \tilde{h} \rangle \cdot k} &= \nu \tilde{l}. (\text{KillP } h_1 h_2 l_1) \mid \dots \mid (\text{KillP } h_{n-1} l_{n-3} l_{n-2}) \mid (\text{KillP } h_n l_{n-2} k) \\
\text{Kill}_\kappa &= \mathbf{0} \quad \text{otherwise}
\end{aligned}$$

Fig. 7. Encoding $\rho\pi$ configurations

part of this section is devoted to prove that the encoding is faithful, i.e. that it preserves the semantics of the original process. More precisely, we will prove the following theorem:

Theorem 2 (Operational correspondance). *For any closed $\rho\pi$ process P , $\nu k. k : P \approx \langle \nu k. k : P \rangle$.*

One cannot simply prove that given a (consistent) configuration M , if $M \rightarrow M'$ then $\langle M \rangle \Rightarrow \langle M' \rangle$. In fact this does not hold, since the translated processes produce some garbage, and since structural congruent processes do not always have structural congruent translations. Thus we need some auxiliary machinery.

Definition 4. *Let \equiv_{Ex} be the smallest congruence satisfying the rules for structural congruence \equiv plus the rules below.*

$$(\text{Ex.NIL}) \quad \nu l_1, l_2. l_1 \langle \text{Nil} \rangle \mid (\langle R \rangle l_2) \mid (\text{KillP } l_1 l_2 l) \equiv_{Ex} (\langle R \rangle l)$$

$$(\text{Ex.A}) \quad \nu l'. (\text{KillP } l_1 l_2 l') \mid (\text{KillP } l' l_3 l) \equiv_{Ex} \nu l'. (\text{KillP } l_1 l' l) \mid (\text{KillP } l_2 l_3 l')$$

$$(\text{Ex.UNFOLD}) \quad \langle P \rangle l \equiv_{Ex} l \langle \langle P \rangle \rangle \mid (\text{Rew } l)$$

Furthermore let \equiv_C be the smallest congruence satisfying rules for structural congruence \equiv plus the first two rules above.

Definition 5. *Let P be a $HO\pi^+$ process. Then $\text{addG}(P)$ is any process obtained from P by applying one or more times the following rules:*

$$\begin{array}{ll}
\mathbb{C}[P'] \mapsto \mathbb{C}[P' \mid (\text{Rew } l)] & \mathbb{C}[P'] \mapsto \mathbb{C}[P' \mid (\text{KillM } a X l)] \\
\mathbb{C}[P'] \mapsto \mathbb{C}[P' \mid (\text{KillP } h k l)] & \mathbb{C}[P'] \mapsto \mathbb{C}[P' \mid \nu t. (a(X, k) \mid t \triangleright Q)]
\end{array}$$

We characterize now the effect of the encoding on structural congruent configurations.

Lemma 10. *Let M, N be configurations. Then $M \equiv N$ implies $(M) \equiv_C (N)$.*

It is easy to see that names in \mathcal{K} are always bound.

Lemma 11. *If $(\nu k. k : P) \Rightarrow P'$ then $\text{fn}(P') \cap \mathcal{K} = \emptyset$.*

The next lemma shows that the encoding of a process can mimic its reductions.

Lemma 12. *For each consistent configuration M , if $M \rightarrow M'$ then $(M) \Rightarrow P$ with $P \equiv_{Ex} \text{addG}((M'))$.*

Proof. By induction on the derivation of $M \rightarrow M'$. The two base cases correspond to \rightsquigarrow and \rightsquigleftarrow . The garbage produced by the translation is managed by function addG . For closure under context, the proof is easy by induction on the structure of the context. Closure under structural congruence is managed by relation \equiv_C . \square

We can now prove our main result.

Proof (of Theorem 2). We have to prove that the following relation is a barbed bisimulation:

$$\mathcal{R} = \{(M, N) \mid \nu k. k : P \Rightarrow M \wedge N \equiv_{Ex} \text{addG}((M))\}$$

For names in \mathcal{K} , the condition on barbs follows from Lemma 11. For names in \mathcal{N} , it is proved by observing that \equiv_{Ex} and $\text{addG}(\bullet)$ do not change those barbs. For terms of the form (M) the condition is proved by structural induction on M .

Then we have to show that each reduction of M is matched by a reduction of N and viceversa. The first direction follows from Lemma 12. It is easy to see that all the processes related to M can simulate its transitions.

For the other direction we have a case analysis according to the channels involved in the reduction $N \rightarrow N'$. Notably, since all reductions are reversible, reductions can not change the set of weak barbs. The case analysis follows:

- reductions involving a message on a name $a \in \text{names}$: these correspond to a transition $M \rightsquigarrow M'$, and it is easy to see that $M' \mathcal{R} N'$;
- reductions involving a memory: these correspond to a transition $M \rightsquigleftarrow M'$, and it is easy to see that $M' \mathcal{R} N'$;
- reductions involving a Kill process: these do not correspond to any transition of M , and it is easy to see that $M \mathcal{R} N'$.

Note that garbage does not add further reductions. This is trivial for triggers with a bound premise. This will hold for other triggers too, since for each k there is at most one message on k at the time, and when such a message is produced the trigger consuming it is produced too. Thus additional triggers are redundant. \square

5 Conclusion and Related Work

We have presented a reversible asynchronous higher-order π -calculus, which we have shown to be causally consistent. The paper gets its inspiration from Danos and Krivine work [5] and makes two 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, compared to the two previous approaches of RCCS [5], that relied on memory stacks as thread tags, and of Phillips and Ulidowski[10], that relied on making the structure of terms in SOS rules static and on keeping track of causality by tagging actions in SOS rules. A further paper by Danos et al. [7] 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 HO π), which is not entirely trivial. The second contribution of the paper is a faithful encoding on our reversible HO π calculus into a variant of HO π , showing that adding reversibility does not change substantially the expressive power of HO π . This result is consistent with, though not reducible to, Boreale and Sangiorgi’s encoding of the π -calculus with causality in the π -calculus itself [3]. Our encoding trades simplicity and weak barbed bisimilarity for divergence in several places. It would be interesting to see whether divergence added by the encoding can be eliminated while still preserving weak bisimilarity, and furthermore whether we can extend our result to obtain full abstraction (proving that weak barbed congruence on $\rho\pi$ configurations corresponds to weak barbed congruence of their encodings).

Acknowledgments. Many thanks to Cosimo Laneve and Alan Schmitt for interesting suggestions, and to anonymous referees for suggested clarifications.

References

1. Barendregt, H.P.: The Lambda Calculus – Its Syntax and Semantics. North-Holland, Amsterdam (1984)
2. Bennett, C.H.: Notes on the history of reversible computation. IBM Journal of Research and Development 32(1) (1988)
3. Boreale, M., Sangiorgi, D.: A fully abstract semantics for causality in the π -calculus. Acta Informatica 35(5) (1998)
4. Carbone, M., Maffeis, S.: On the expressive power of polyadic synchronisation in pi-calculus. Nord. J. Comput. 10(2) (2003)
5. Danos, V., Krivine, J.: Reversible communicating systems. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 292–307. Springer, Heidelberg (2004)
6. Danos, V., Krivine, J.: Transactions in RCCS. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 398–412. Springer, Heidelberg (2005)

7. Danos, V., Krivine, J., Sobociński, P.: General reversibility. *Electr. Notes Theor. Comput. Sci.* 175(3) (2007)
8. Fournet, C., Gonthier, G.: The reflexive chemical abstract machine and the join-calculus. In: 23rd ACM Symp. on Princ. of Prog. Languages, POPL (1996)
9. Mousavi, M.R., Reniers, M.A., Groote, J.F.: SOS formats and meta-theory: 20 years after. *Theor. Comput. Sci.* 373(3) (2007)
10. Phillips, I., Ulidowski, I.: Reversing algebraic process calculi. *J. Log. Algebr. Program.* 73(1-2) (2007)
11. Sangiorgi, D.: Bisimulation for higher-order process calculi. *Information and Computation* 131(2) (1996)
12. Sangiorgi, D., Walker, D.: The π -calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)