A Parametric Framework for Reversible π-Calculi*

Doriana Medic
IMT School for Advanced Studies Lucca, Italy
doriana.medic@imtlucca.it

Claudio Antares Mezzina
IMT School for Advanced Studies Lucca, Italy
claudio.mezzina@imtlucca.it

Iain Phillips
Imperial College London, UK
i.phillips@imperial.ac.uk

Nobuko Yoshida
Imperial College London, UK
n.yoshida@imperial.ac.uk

This paper presents a study of causality in a reversible, concurrent setting. There exist various notions of causality in π-calculus, which differ in the treatment of parallel extrusions of the same name. In this paper we present a uniform framework for reversible π-calculi that is *parametric* with respect to a data structure that stores information about an extrusion of a name. Different data structures yield different approaches to the parallel extrusion problem. We map three well-known causal semantics into our framework. We show that the (parametric) reversibility induced by our framework is causally-consistent and prove a causal correspondence between an appropriate instance of the framework and Boreale and Sangiorgi's causal semantics.

1 Introduction

Starting from the 1970s [5] reversible computing has attracted interest in different fields, from thermodynamical physics [3], to systems biology [14, 28], system debugging [30, 16] and quantum computing [17]. Of particular interest is its application to the study of programming abstractions for reliable systems: most fault-tolerant schemes exploiting system recovery techniques [2] rely on some form of *undo*. Examples of how reversibility can be used to model transactions exist in CCS [13] and higher-order π-calculus [19].

A reversible system is able to execute both in the forward (normal) direction and in the backward one. In a sequential setting, there is just one order of reversing a computation: one has just to undo the computation by starting from the last action. In a concurrent system there is no clear notion of last action. A good approximation of what is the last action in a concurrent system is given by *causally-consistent* reversibility, introduced by Danos and Krivine for reversible CCS [12]. Causally-consistent reversibility relates causality and reversibility of a concurrent system in the following way: an action can be reversed, and hence considered as a last one, provided all its consequences have been reversed.

In CCS [25], there exists just one notion of causality: so-called *structural* causality, which is induced by the prefixing '.' operator and by synchronisations. As a consequence, there is only one way of reversing a CCS trace, and from an abstract point of view there exists only one reversible CCS. Evidence for this has been given in [22], where an equivalence is shown between the two methods for reversing CCS (namely RCCS [12] and CCSK [27]).

When moving to more expressive calculi with name creation and value passing like the π-calculus, matters are more complex. As in CCS, structural causality in the π-calculus is determined by the nesting of the prefixes; for example, in process $\overline{b}a.\overline{c}e$ the output on channel c structurally depends on the output on b. Extruding (or opening) a name generates an *object* dependency; for example, in process $\nu a\,(\overline{b}a \mid a(z))$

*This work was partially supported by COST Action IC1405 "Reversible computation – extending horizons of computing", EPSRC EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1 and EP/N028201/1.

J.A. Pérez and S. Tini (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2018).
EPTCS 276, 2018, pp. 87–103, doi:10.4204/EPTCS.276.8

the input action on a depends on the output on b. In the case of parallel extrusions of the same name, for example $\nu a\,(\overline{b}a \mid \overline{c}a \mid a(z))$, there exist different interpretations of which extrusion will cause the action $a(z)$. In what follows, we consider three approaches.

The classical and the most used approach to causality in the π-calculus is the one where the order of extrusions matters and the first one of them is the cause of the action $a(z)$. Some of the causal semantics representing this idea are [15, 6, 8] and all of them are defined for standard (forward-only) π-calculus. In [15] the authors claim that, after abstracting away from the technique used to record causal dependences, the final order between the actions in their semantics coincides with the ones introduced in [6, 8]. Hence we group these semantics together as a single approach to causality.

Secondly, in [9], action $a(z)$ in the example above depends on one of the extruders, but there is no need to keep track of which one exactly. This causal semantics is defined for the forward-only π-calculus.

Finally, the first compositional causal semantics for the reversible π-calculus is introduced in [10]. In the above example, parallel extrusions are concurrent and the action $a(z)$ will record dependence on one of them (exactly which one is decided by the context). This causal semantics enjoys certain correctness properties which are not satisfied by other semantics.

Here we present a framework for reversible π-calculus that is parametric with respect to the data structure that stores information about an extrusion of a name. Different data structures will lead to different approaches to the parallel extrusion problem, including the three described above. Our framework allows us to add reversibility to semantics where it was not previously defined. By varying the parameters, different orderings of the causally-consistent backward steps are allowed. Our intention is to develop a causal behavioural theory for the framework, in order to better understand different interpretations of reversibility in the π-calculus, and to use this understanding for causal analysis of concurrent programs.

A preliminary discussion of the framework appeared in [23], where some initial ideas were given. Moreover in [23] it was argued that it was necessary to modify the semantics of [6] in order to add information about silent actions. In this work we fully develop the idea behind the framework and leave the semantics of [6] unchanged, apart from using a late semantics, rather than early as originally given.

Contributions. We present a framework for reversible π-calculus which is parametric in the bookkeeping data structure used to keep track of object dependency. As reversing technique, we will extend the one introduced by CCSK [27], which is limited to calculi defined with GSOS [1] inference rules (e.g., CCS, CSP), to work with more expressive calculi featuring name passing and binders. This choice allows us to have a compositional semantics which does not rely on any congruence rule (in particular the splitting rule used by [10]). Depending on the bookkeeping data structure used to instantiate the framework, we can obtain different causal semantics (i.e., [10, 6, 9]). We then show that our framework enjoys the standard properties for a reversible calculus, namely the loop lemma and causal consistency, regardless of the notion of causality which is used. We prove causal correspondence between the causal semantics introduced in [6] and the matching instance of our framework.

The rest of the paper is as follows: syntax and operational semantics of the framework are given in Section 2. In Section 3, we show how by using different data structures we can encompass different causal semantics. The main results are given in Section 4, and Section 5 concludes the paper. Proofs are omitted for space reasons; they can be found in the extended version [24].

2 The Framework

We present the syntax and operational semantics of our parametric framework, after an informal introduction.

2.1 Informal presentation

In [27] a general technique to reverse any CCS-like calculus is given. The key ideas are to use communication keys to identify events, and to make static all the operators of the calculus, since dynamic operators such as choice and prefix are *forgetful* operators. For example, if we take a CCS process $a.P \mid \overline{a}.Q$ a possible computation is:

$$a.P \mid \overline{a}.Q \xrightarrow{\tau[i]} a[i].P \mid \overline{a}[i].Q$$

As one can see, prefixes are not destroyed but *decorated* with a communication key. The obtained process acts like $P \mid Q$, since decorated prefixes are just used for backward steps. We bring this idea to the π-calculus. For example by lifting this process into π-calculus we have something like

$$a(x).P \mid \overline{a}b.Q \xrightarrow{\tau:i} a(x)[i].P\{b^i/x\} \mid \overline{a}b[i].Q$$

In the substitution $\{b^i/x\}$, name b is decorated with the key i to record that it was substituted for variable x in the synchronisation identified by the communication key i. The key i is also recorded in the memories.

By choosing to adapt the ideas of [27] to work with the π-calculus, we avoid using the splitting rule of $R\pi$ [10]. In $R\pi$ each process is monitored by a memory, $m \triangleright P$, which is in charge of recording all past events of the process. In this way, the past of the process is not recorded directly in the process. One drawback of this approach is that one needs to resort to a splitting rule of the form

$$m \triangleright (P \mid Q) \equiv \langle\uparrow\rangle \cdot m \triangleright P \mid \langle\uparrow\rangle \cdot m \triangleright Q$$

to let both P and Q execute. This rule is not associative and moreover, as shown in [20], introduces some undesired non-determinism, since equivalent processes performing the same action may become non-equivalent processes.

The framework has to remember extrusions, and in particular who was the extruder of a certain name, and what is the *contextual cause* for an action. For example in

$$\nu a\,(\overline{b}a \mid a(x).P) \xrightarrow{\overline{b}\langle\nu a\rangle:i} \nu a_{\{i\}}(\overline{b}a[i] \mid a(x).P) \xrightarrow{a(x):j} \nu a_{\{i\}}(\overline{b}a[i] \mid a(x)[j,i].P)$$

we have that after the extrusion, the restriction νa does not disappear as in standard π-calculus, but remains where it was, becoming the memory $\nu a_{\{i\}}$ (introduced in [10]). This memory records the fact that name a was extruded because of transition i. Moreover, since it is no longer a restriction but just a decoration, the following transition using name a can take place. Transition j uses i as its contextual cause, indicating that the input action can happen on a because it was extruded by i, and this is recorded in the process $a(x)[j,i].P$.

2.2 Syntax

We assume the existence of the following *denumerable* infinite mutually disjoint sets: the set $\mathscr{N}$ of names, the set $\mathscr{K}$ of keys, and the set $\mathscr{V}$ of variables. Moreover, we let $\mathscr{K}_* = \mathscr{K} \cup \{*\}$ where $*$ is a special key. We let a,b,c range over $\mathscr{N}$; x,y range over $\mathscr{V}$ and i,j,k range over $\mathscr{K}$.

The syntax of the framework is depicted in Figure 1. *Processes*, given by the P,Q productions, are the standard processes of the π-calculus [29]: $\mathbf{0}$ represents the idle process; $\overline{b}c.P$ is the *output*-prefixed process indicating the act of sending name c over channel b; $b(x).P$ is the *input*-prefixed process indicating the act of receiving a value (which will be bound to the variable x) on channel b. Process $P \mid Q$ represents the *parallel* composition of two processes, while $\nu a(P)$ represents the fact that name a is *restricted* in P.

Reversibility is defined on the top of the π-calculus. Unlike in the standard π-calculus, executed actions are not discarded. Each of them, followed by the memory, becomes a part of the process that we shall call the *history*. *Reversible processes* are given by X,Y productions. A reversible process $\mathbf{P}$ is a

$$X, Y ::= \quad \mathbf{P} \mid \overline{b}^{j} a^{j_1}[i,K].X \mid b^{j}(x)[i,K].X \mid X \mid Y \mid \nu a_{\Delta}(X)$$
$$P, Q ::= \quad \mathbf{0} \mid \overline{b}c.P \mid b(x).P \mid P \mid Q \mid \nu a(P)$$

Figure 1: Syntax.

standard π-calculus process P where channels are decorated with instantiators. As we shall see later on, instantiators are used to keep track of substitutions. In a prefix of the form $\overline{b}a$ or $b(x)$ we say that name b is used in *subject position*, while name a and variable x are in *object position*. We shall use operators $sub(\cdot)$ and $obj(\cdot)$ to get respectively the subject and the object of a prefix. The prefix $\overline{b}^{j} a^{j_1}[i,K].X$ represents a *past output* recording the fact that in the past the process X performed an output identified by key i and that its contextual cause set was $K \subseteq \mathscr{K}_*$. Prefix $b^{j}(x)[i,K].X$ represents a *past input* recording the fact that the input was identified by key i and its contextual cause set was K. If it is not relevant whether the prefix in the process is an input or an output, we shall denote it with α ($\alpha = \overline{b}^{j} a^{j_1}$ or $\alpha = b^{j}(x)$).

Following [10] the restriction operator νa_{Δ} is decorated with the memory Δ which keeps track of the extruders of a name a. As we shall see later on, we shall abstract away from the form of Δ, as different data structures lead to different notions of causality. When $\texttt{empty}(\Delta) = true$, the data structure is initialised and νa_{Δ} will act as the usual restriction operator νa of the π-calculus. The set of reversible processes is denoted with $\mathscr{X}$.

To simplify manipulation with reversible processes, we shall define history and general context. History context represents the reversible process X made of executed prefixes. For example, we can express the process $X = \overline{b}^{*} a^{*}[i,K].\overline{c}^{*} a^{*}[i',K'].\mathbf{P}$ as $X = \mathtt{H}[\mathbf{P}]$ with $\mathtt{H}[\bullet] = \overline{b}^{*} a^{*}[i,K].\overline{c}^{*} a^{*}[i',K'].\bullet$. General context is defined on the top of the history context by adding parallel and restriction operators on it. For example, the process $Z \mid Y \mid X$ can be written as $C[X]$ if the only relevant element is X. Formally:

Definition 1 (History and General context)**.** *History contexts* $\mathtt{H}$ *and general contexts C are reversible processes with a hole* $\bullet$*, defined by the following grammar:*

$$\mathtt{H} ::= \bullet \mid \alpha[i,K].\bullet \qquad C ::= \mathtt{H}[\bullet] \mid X \mid \bullet \mid \nu a_{\Delta}(\bullet)$$

Free names and free variables. Notions of free names and free variables in our framework are standard. It suffices to note that constructs with binders are of the forms: $\nu a_{\Delta}(X)$ when $\texttt{empty}(\Delta)$ holds, which binds the name a with scope X; and $b(x).P$, which binds the variable x with scope P. We denote with $\texttt{fn}(P)$ and $\texttt{fn}(X)$ the set of free names of P and of X respectively.

Remark 1. Annotation b^{*} to a name b, used either in the subject or in the object position, indicates that name b has no instantiators.

Since the framework will be parametric in the data structure Δ, we specify it as an interface (in the style of a Java interface) by defining the operations that it has to offer.

Definition 2. Δ *is a data structure with the following defined operations:*

(i) $\texttt{init} : \Delta \to \Delta$ *initialises the data structure*

(ii) $\texttt{empty} : \Delta \to \texttt{bool}$ *predicate telling whether* Δ *is empty*

(iii) $+ : \Delta \times \mathscr{K} \to \Delta$ *operation adding a key to* Δ

(iv) $\#i : \Delta \times \mathscr{K} \to \Delta$ *operation removing a key from* Δ

(v) $\in : \Delta \times \mathscr{K} \to \texttt{bool}$ *predicate telling whether a key belongs to* Δ

We now define three instances of Δ: sets, sets indexed with an element and sets indexed with a set. As we shall see, these three instances will give rise to three different notions of object causality.

Set. Γ is a set containing keys (i.e. $\Gamma \subseteq \mathscr{K}$). The intuition of νa_Γ is that **any** of the elements contained in Γ can be a contextual cause for a (i.e., the reason why a is known to the context).

Definition 3 (Operations on a set)**.** *The operations on a set* Γ *are defined as:*

(i) $\texttt{init}(\Gamma) = \emptyset$

(ii) $\texttt{empty}(\Gamma) = true$, *when* $\Gamma = \emptyset$

(iii) $+$ *is the classical addition of elements to a set*

(iv) $\#i$ *is defined as the identity, that is* $\Delta_{\#i} = \Gamma_{\#i} = \Gamma$.

(v) $i \in \Gamma$ *the key* i *belongs to the set* Γ

Indexed set. Γ_w is an indexed set containing keys and w is the key of the action which extruded a name a. In this case the contextual cause for name a can be **just** w. If there is no cause, then we shall set $w = *$.

Definition 4 (Operations on an indexed set)**.** *The operations on an indexed set* Γ_w *are defined as:*

(i) $\texttt{init}(\Gamma_w) = \emptyset_*$

(ii) $\texttt{empty}(\Gamma_w) = true$, *when* $\Gamma = \emptyset \wedge w = *$

(iii) *operation* $+$ *is defined as:* $\Gamma_w + i = \begin{cases} (\Gamma \cup \{i\})_i, \text{ when } w = * \\ (\Gamma \cup \{i\})_w, \text{ when } w \neq * \end{cases}$

(iv) *operation* $\#i$ *is defined inductively as:*

$$(X \mid Y)_{\#i} = X_{\#i} \mid Y_{\#i} \qquad (\texttt{H}[X])_{\#i} = \texttt{H}[X_{\#i}] \qquad (\mathbf{P})_{\#i} = \mathbf{P}$$
$$(\nu a_{\Gamma_i} X)_{\#i} = \nu a_{\Gamma_*} X_{\#i} \qquad (\nu a_{\Gamma_w} X)_{\#i} = \nu a_{\Gamma_w} X_{\#i}$$

(v) $i \in \Gamma_w$ *the key* i *belongs to the set* Γ*, regardless of* w *(e.g.* $i \in \{i\}_*$*)*

Set indexed with a set. Γ_Ω is a set containing keys indexed with a set $\Omega \in \mathscr{K}_*$. Extruders of name a which are not part of the communication, will be saved in the set Ω. In this case the contextual cause for name a is a set Ω. If there is no cause, then we shall set $\Omega = \{*\}$.

Definition 5 (Operations on a set indexed with a set)**.** *The operations on a set indexed with a set* Γ_Ω *are defined as:*

(i) $\texttt{init}(\Gamma_\Omega) = \emptyset_{\{*\}}$

(ii) $\texttt{empty}(\Gamma_\Omega) = true$, *when* $\Gamma = \emptyset \wedge \Omega = \{*\}$

(iii) *operation* $+$ *is defined as:* $(\Gamma_\Omega) + i = (\Gamma \cup \{i\})_{(\Omega \cup \{i\})}$

(iv) *operation* $_{\#i}$ *is defined inductively as:*

$$(X \mid Y)_{\#i} = X_{\#i} \mid Y_{\#i} \qquad (\nu a_{\Gamma_\Omega} X)_{\#i} = \nu a_{\Gamma_{\Omega \setminus \{i\}}} X_{\#i} \qquad (\texttt{H}(X))_{\#i} = \texttt{H}[X_{\#i}] \qquad (\mathbf{P})_{\#i} = \mathbf{P}$$

(v) $i \in \Gamma_\Omega$ *the key* i *belongs to the set* Γ*, regardless* Ω *(e.g.* $i \in \{i\}_{\{*\}}$*)*

$$(\textsc{Out1})\ \overline{b}^{j}a^{j_1}.\mathbf{P} \xrightarrow{(i,K,j):\overline{b}a} \overline{b}^{j}a^{j_1}[i,K].\mathbf{P} \qquad (\textsc{Out2})\ \frac{X \xrightarrow{(i,K,j):\overline{b}a} X' \quad \mathtt{fresh}(i,\mathtt{H}[X])}{\mathtt{H}[X] \xrightarrow{(i,K,j):\overline{b}a} \mathtt{H}[X']}$$

$$(\textsc{In1})\ b^{j}(x).\mathbf{P} \xrightarrow{(i,K,j):b(x)} b^{j}(x)[i,K].\mathbf{P} \qquad (\textsc{In2})\ \frac{X \xrightarrow{(i,K,j):b(x)} X' \quad \mathtt{fresh}(i,\mathtt{H}[X])}{\mathtt{H}[X] \xrightarrow{(i,K,j):b(x)} \mathtt{H}[X']}$$

$$(\textsc{Par})\ \frac{X \xrightarrow{(i,K,j):\pi} X' \quad i \notin Y}{X \mid Y \xrightarrow{(i,K,j):\pi} X' \mid Y} \qquad (\textsc{Res})\ \frac{X \xrightarrow{(i,K,j):\pi} X' \quad a \notin \pi}{\nu a_{\Delta}(X) \xrightarrow{(i,K,j):\pi} \nu a_{\Delta}(X')}$$

$$(\textsc{Com})\ \frac{X \xrightarrow{(i,K,j):\overline{b}a} X' \quad Y \xrightarrow{(i,K',j'):b(x)} Y' \quad K =_{*} j' \wedge K' =_{*} j}{X \mid Y \xrightarrow{(i,*,*):\tau} X' \mid Y'\{a^{i}/x\}}$$

Figure 2: Rules that are common to all instances of the framework.

2.3 Operational Semantics

The grammar of the labels generated by the framework is:

$$\mu ::= (i,K,j) : \pi \qquad \pi ::= \overline{b}c \mid b(x) \mid \overline{b}\langle \nu c_{\Delta} \rangle \mid \tau$$

where i is the key, and $K \subseteq \mathscr{K}_{*}$, $j \in \mathscr{K}_{*}$ are the set of contextual causes and an instantiator of i, respectively. If there is no action which caused and/or instantiated i, we denote this with $K = \{*\}$, $j = *$, respectively. The set $\mathscr{L}$ of all possible labels generated by the framework is defined as $\mathscr{L} = \mathscr{K} \times \mathscr{K}_{*} \times \mathscr{K}_{*} \times \mathscr{A}$, where $\mathscr{A}$ is a set of actions ranged over by π. We extend $sub(\cdot)$ and $obj(\cdot)$ to apply also to labels.

The operational semantics of the reversible framework is given in terms of a labelled transition system (LTS) $(\mathscr{X}, \mathscr{L}, \rightarrow)$, where $\mathscr{X}$ is the set of reversible processes; $\rightarrow = \twoheadrightarrow \cup \rightsquigarrow$ where $\twoheadrightarrow$ is the least transition relation induced by the rules in Figures 2 and 3; and $\rightsquigarrow$ is the least transition relation induced by the rules in Figure 4.

Definition 6 (Process keys)**.** *The set of* communication keys *of a process X, written* $\mathtt{key}(X)$*, is inductively defined as follows:*

$$\mathtt{key}(X \mid Y) = \mathtt{key}(X) \cup \mathtt{key}(Y) \qquad \mathtt{key}(\alpha[i,K].X) = \{i\} \cup \mathtt{key}(X)$$
$$\mathtt{key}(\nu a_{\Delta}(X)) = \mathtt{key}(X) \qquad \mathtt{key}(\mathbf{P}) = \emptyset$$

Definition 7. *A key i is* fresh *in a process X, written* $\mathtt{fresh}(i,X)$ *if* $i \notin \mathtt{key}(X)$.

The forward rules of a framework are divided into two *groups*, depending on whether they are parametric with respect to Δ or they are common to all the instances of the framework.

Common rules are given in Figure 2. Rules Out1 and In1 generate a fresh new key i which is bound to the action. Rules Out2 and In2 inductively allow a prefixed process $\mathtt{H}[X]$ to execute if X can execute. Condition $i \notin Y$ in rule Par ensures that action keys are unique. Rule Res is defined in the usual way. Two processes can synchronise through the rule Com if the additional condition is satisfied ($K =_{*} j$ means $* \in K$ or $j = *$ or $K = j$). After the communication, necessary substitution is applied to the rest of the input process. In the process $Y'\{a^{i}/x\}$ every occurrence of variable $x \in \mathtt{fn}(Y')$ is substituted with the name a^{i}, that is, the name a decorated with the key i of the action which was executed. In the further

$$(\textsc{Cause Ref})\ \frac{X \xrightarrow{(i,K,j):\pi} X' \quad a \in sub(\pi) \qquad \texttt{empty}(\Delta) \neq true \quad \texttt{Cause}(\Delta, K, K')}{\nu a_\Delta(X) \xrightarrow{(i,K',j):\pi} \nu a_\Delta(X'_{[K'/K]\&i})}$$

$$(\textsc{Open})\ \frac{X \xrightarrow{(i,K,j):\pi} X' \quad \pi = \overline{b}a \vee \pi = \overline{b}\langle \nu a_{\Delta'} \rangle \quad \texttt{Update}(\Delta, K, K')}{\nu a_\Delta(X) \xrightarrow{(i,K',j):\overline{b}\langle \nu a_\Delta \rangle} \nu a_{\Delta+i}(X'_{[K'/K]\&i})}$$

$$(\textsc{Close})\ \frac{X \xrightarrow{(i,K,j):\overline{b}\langle \nu a_\Delta \rangle} X' \quad Y \xrightarrow{(i,K',j'):b(x)} Y' \quad K =_* j' \wedge K' =_* j}{X \mid Y \xrightarrow{(i,*,*):\tau} \nu a_\Delta(X'_{\#i} \mid Y'\{a^i/x\})}$$

Figure 3: Parametric rules

actions of a process $Y'\{a^i/x\}$, the key i will be called the *instantiator*. The instantiators are used just to keep track of the substitution, not to define a name. For example, the two processes $\overline{b}^j a^*.\mathbf{P}$ and $b^{j'}(x).\mathbf{P}'$ can communicate, even if the instantiators of the name b are not the same. Let us note that we use a *late* semantics, since substitution happens in the rule COM. In order to understand how the basic rules work let us consider the following example.

Example 1. *Let $X = \overline{b}^* a^*.\mathbf{0} \mid b^*(x).\overline{x}c^*$. There are two possibilities for the process X:*

- *process X can preform an output and an input action on the channel b while synchronising with environment:*

$$\overline{b}^* a^*.\mathbf{0} \mid b^*(x).\overline{x}c^* \xrightarrow{(i,*,*):\overline{b}a} \overline{b}^* a^*[i,*].\mathbf{0} \mid b^*(x).\overline{x}c^* \xrightarrow{(i',*,*):b(x)} \overline{b}^* a^*[i,*].\mathbf{0} \mid b^*(x)[i',*].\overline{x}c^* = Y_1$$

  As we can notice, the output action $\overline{b}a$ is identified by key i, while the input action is identified by key i'.

- *The synchronisation can happen inside of the process X:*

$$\overline{b}^* a^*.\mathbf{0} \mid b^*(x).\overline{x}c^* \xrightarrow{(i,*,*):\tau} \overline{b}^* a^*[i,*].\mathbf{0} \mid b^*(x)[i,*].\overline{a}^i c^* = Y_2$$

  We can notice that τ action is identified with key i and during the synchronisation variable x is substituted with a received name a decorated with the key i of the executed action. In this way we keep track of the substitution of a name.

We now define the operation $X_{[K'/K]\&i}$, which updates the contextual cause K of an action identified by i with the new cause K'. Contextual cause update will be used in the the parametric rules of Figure 3 (OPEN and CAUSE REF). Formally:

Definition 8 (Contextual Cause Update)**.** *The* contextual cause update *of a process, written $X_{[K'/K]\&i}$ is defined as follows:*

$$(X \mid Y)_{[K'/K]\&i} = X_{[K'/K]\&i} \mid Y_{[K'/K]\&i} \qquad \texttt{H}[\alpha[i,K].X]_{[K'/K]\&i} = \texttt{H}[\alpha[i,K'].X]$$
$$(\nu a_\Delta(X))_{[K'/K]\&i} = \nu a_\Delta(X)_{[K'/K]\&i} \qquad \texttt{H}[\alpha[j,K].X]_{[K'/K]\&i} = \texttt{H}[\alpha[j,K].X]$$

Parametric rules are given in Figure 3. Depending on the underlying causal semantics the way a contextual cause is chosen differs. This is why we need to define two predicates: $\texttt{Cause}(\cdot)$ and $\texttt{Update}(\cdot)$.

$$(\text{OUT1}^\bullet)\ \overline{b}^j a[i,K].\mathbf{P} \overset{(i,K,j):\overline{b}a}{\rightsquigarrow} \overline{b}^j a.\mathbf{P} \qquad (\text{OUT2}^\bullet)\ \frac{X' \overset{(i,K,j):\overline{b}a}{\rightsquigarrow} X \quad \texttt{fresh}(i,\texttt{H}[X])}{\texttt{H}[X'] \overset{(i,K,j):\overline{b}a}{\rightsquigarrow} \texttt{H}[X]}$$

$$(\text{IN1}^\bullet)\ b^j(x)[i,K].\mathbf{P} \overset{(i,K,j):b(x)}{\rightsquigarrow} b^j(x).\mathbf{P} \qquad (\text{IN2}^\bullet)\ \frac{X' \overset{(i,K,j):b(x)}{\rightsquigarrow} X \quad \texttt{fresh}(i,\texttt{H}[X])}{\texttt{H}[X'] \overset{(i,K,j):b(x)}{\rightsquigarrow} \texttt{H}[X]}$$

$$(\text{PAR}^\bullet)\ \frac{X' \overset{(i,K,j):\pi}{\rightsquigarrow} X \quad i \notin Y}{X' \mid Y \overset{(i,K,j):\pi}{\rightsquigarrow} X \mid Y} \qquad (\text{COM}^\bullet)\ \frac{X' \overset{(i,K,j):\overline{b}a}{\rightsquigarrow} X \quad Y' \overset{(i,K',j'):b(x)}{\rightsquigarrow} Y \quad K =_* j' \wedge K' =_* j}{X' \mid Y' \overset{(i,*,*):\tau}{\rightsquigarrow} X \mid Y\{x/a^i\}}$$

$$(\text{RES}^\bullet)\ \frac{X' \overset{(i,K,j):\pi}{\rightsquigarrow} X \quad a \notin \pi}{\nu a_\Delta(X') \overset{(i,K,j):\pi}{\rightsquigarrow} \nu a_\Delta(X)} \qquad (\text{OPEN}^\bullet)\ \frac{X' \overset{(i,K,j):\pi}{\rightsquigarrow} X \quad \pi = \overline{b}a \vee \pi = \overline{b}\langle \nu a_{\Delta'} \rangle \quad \texttt{Update}(\Delta,K,K')}{\nu a_{\Delta+i}(X') \overset{(i,K',j):\overline{b}\langle \nu a_\Delta \rangle}{\rightsquigarrow} \nu a_\Delta(X)}$$

$$(\text{CAUSE REF}^\bullet)\ \frac{X' \overset{(i,K,j):\pi}{\rightsquigarrow} X \quad a \in sub(\pi) \qquad \texttt{empty}(\Delta) \neq true \quad \texttt{Cause}(\Delta,K,K')}{\nu a_\Delta(X') \overset{(i,K',j):\pi}{\rightsquigarrow} \nu a_\Delta(X)}$$

$$(\text{CLOSE}^\bullet)\ \frac{X' \overset{(i,K,j):\overline{b}\langle \nu a_\Delta \rangle}{\rightsquigarrow} X \quad Y' \overset{(i,K',j'):b(x)}{\rightsquigarrow} Y \quad K =_* j' \wedge K' =_* j}{\nu a_\Delta(X' \mid Y') \overset{(i,*,*):\tau}{\rightsquigarrow} X \mid Y\{x/a^i\}}$$

Figure 4: Backward rules.

When instantiating Δ with a specific data structure, different implementations of such predicates are needed. We shall define them precisely when discussing various causal semantics in the Section 3. Every time a label produced by a process X passes the restriction νa_Δ it needs to check if it is necessary to modify the contextual cause. Depending on whether name a is in the subject or in the object position in the label of an action, rules CAUSE REF or OPEN can be used, respectively. Rule CAUSE REF is used when the subject of a label is an already extruded name and a predicate $\texttt{Cause}(\Delta,K,K')$ tells whether contextual cause K has to be substituted with K'. Rule OPEN deals with the scope extrusion of a restricted name. If the restricted name a is used as object of a label with key i we have to record that i is one of the potential extruders of a. Naturally, if $\texttt{empty}(\Delta) = true$ then the first extruder initialises the data structure. Also in this case it might happen that we have to update the contextual cause of the label i. This is why predicate $\texttt{Update}(\Delta,K,K')$ is used. Two processes can synchronise through the rule CLOSE satisfying the additional condition. In some semantics, silent actions do not bring the causal information on what is a reason to introduce the operator $_{\#i}$, where every time when an extruded name is closed over the context, the key of the closing action is deleted from indexes of Δ_h in all restrictions $\nu a_{\Delta_h} \in X'$.

Backward rules are symmetric to the forward ones; they are presented in Figure 4. The predicates are not necessary for the backward transitions as they are invariant in the history of processes but we keep them to simplify the proofs. In order to better understand the backward rules, we shall consider the following example.

Example 2. *Let us consider the following processes from Example 1:*

- $Y_1 = \overline{b}^* a^*[i,*].\mathbf{0} \mid b^*(x)[i',*].\overline{x}c^*$*; Process* Y_1 *can perform backward actions on the channel b (an*

input action identified with key i' and an output action identified with key i) in any order. For example, let us reverse first the input and then the output action:

$$Y_1 = \overline{b}^* a^*[i,*].\mathbf{0} \mid b^*(x)[i',*].\bar{x}c^* \overset{(i',*,*):b(x)}{\rightsquigarrow} \overline{b}^* a^*[i,*].\mathbf{0} \mid b^*(x).\bar{x}c^* \overset{(i,*,*):\overline{b}a}{\rightsquigarrow} \overline{b}^* a^*.\mathbf{0} \mid b^*(x).\bar{x}c^* = X$$

We notice that all the necessary elements to reverse the action $b(x)$ are saved in the history part of the process Y_1.

- *$Y_2 = \overline{b}^* a^*[i,*].\mathbf{0} \mid b^*(x)[i,*].\overline{a}^i c^*$; Process Y_2 can reverse the communication which happened on the channel b, between its subprocesses. Due to the side condition of the rule* PAR$^\bullet$*, it is impossible to reverse an input or an output action separately:*

$$\overline{b}^* a^*[i,*].\mathbf{0} \mid b^*(x)[i,*].\overline{a}^i c^* \not\overset{(i,*,*):\overline{b}a}{\rightsquigarrow} \overline{b}^* a^*.\mathbf{0} \mid b^*(x)[i,*].\overline{a}^i c^*$$

The backward action above cannot be executed as the key i belongs to the process in parallel ($i \in \mathtt{key}(b^*(x)[i,*].\overline{a}^i c^*)$). The only possible backward step is:

$$\overline{b}^* a^*[i,*].\mathbf{0} \mid b^*(x)[i,*].\overline{a}^i c^* \overset{(i,*,*):\tau}{\rightsquigarrow} \overline{b}^* a^*.\mathbf{0} \mid b^*(x).\bar{x}c^* = X$$

Remark 2. The choice operator $(+)$, can be easily added to the framework by following the approach of [27] and by making the operator static.

3 Mapping causal semantics of π into the framework

We now review three notions of causal semantics for π-calculus and show how to map them into our framework by giving the definitions for the side conditions in the rules in Figure 3.

***Rπ*-calculus.** Cristescu et al [10] introduce a compositional semantics for the reversible π-calculus. Information about the past actions is kept in a memory added to every process. A term of the form $m \triangleright P$ represents a reversible process, where memory m is a stack of events and P is the process itself. A memory contains two types of events, one which keeps track of the past action, $\langle i,k,\pi \rangle$, where elements of a triple are the key, the contextual cause and the executed action, respectively; and one which keeps track of the position of the process in the parallel composition, $\langle \uparrow \rangle$. Before executing in parallel, a process splits by duplicating its memory and adding event $\langle \uparrow \rangle$ on the top of each copy. This is achievable with specially defined structural congruence rules. The use in [10] of indexed restriction νa_Γ was the inspiration for our parametric indexed restriction νa_Δ. A key point of the semantics of [10] is that it enjoys certain correctness properties: one of which is that two visible transitions are causally related iff for all contexts the corresponding silent transitions are. Since an action can be caused only through the *subject* of a label we have that contextual cause K will be a singleton. We shall consider one relation between the prefixes into the history. In this way, while changing the cause with the rule CAUSE REF, the condition $\mathtt{Cause}(\cdot)$ needs to keep track of the instantiation of the cause.

Definition 9 (Instantiation relation)**.** *Two keys i_1 and i_2 such that $i_1, i_2 \in \mathtt{key}(X)$ and $X = C[b^*(x)[i_1, K_1].Y]$ with $Y = C'[\alpha^{j_2}[i_2, K_2].Z]$, are in instantiation relation, $i_1 \rightsquigarrow_X i_2$, if $j_2 = i_1$. If $i_1 \rightsquigarrow_X i_2$ holds, we will write $K_1 \rightsquigarrow_X K_2$.*

To obtain $R\pi$ causality in our framework, we need to instantiate the rules of Figure 3 with the following predicates.

Definition 10 ($R\pi$ causality)**.** *If data structure Δ is instantiated with a set Γ, the predicates from Figure 3 are defined as:*

1. $\texttt{Cause}(\Delta,K,K') = \texttt{Cause}(\Gamma,K,K')$ *stands for* $K = K'$ *or* $\exists K' \in \Gamma\ K \rightsquigarrow_X K'$*;*

2. $\texttt{Update}(\Delta,K,K') = \texttt{Update}(\Gamma,K,K')$ *stands for* $K' = K$.

The predicates defined above coincide with the conditions of the semantics introduced in [10]. In the following example we shall give the intuition of the $R\pi$ causality using our framework.

Example 3. *Let us consider the process $X = \nu a_{\emptyset}(\overline{b}^* a^* \mid \overline{c}^* a^* \mid a^*(x))$. By applying rule* OPEN *twice and executing concurrently two extrusions on names b and c, we obtain a process:*

$$\nu a_{\{i,h\}}(\overline{b}^* a^*[i,*] \mid \overline{c}^* a^*[h,*] \mid a^*(x))$$

The rule CAUSE REF *is used for the execution of the third action. By definition of the predicate* $\texttt{Cause}(\cdot)$*, the action $a(x)$ can choose its cause from a set $\{i,h\}$. By choosing h for example, and executing the input action, we get the process:*

$$\nu a_{\{i,h\}}(\overline{b}^* a[i,*] \mid \overline{c}^* a[h,*] \mid a^*(x)[l,h])$$

In the memory $[l,h]$ we can see that the action identified with key l needs to be reversed before the action with key h. Process $\overline{b}^* a[i,*]$ can execute a backward step at any time with the rule OPEN$^\bullet$.

Boreale-Sangiorgi and Degano-Priami causal semantics. A compositional causal semantics for standard (i.e., forward only) π-calculus was introduced by Boreale and Sangiorgi [6]. Later on, Degano and Priami in [15] introduced a causal semantics for π based on localities. While using different approaches to keep track of the dependences in π-calculus, these two approaches impose the same order of the forward actions (as claimed in [15]). Hence, from the reversible point of view we can take it that the causality notions of these two semantics coincide. In what follows we shall concentrate on the Boreale-Sangiorgi causal semantics. To show the correspondence between the mentioned semantics and our framework, we shall consider it in a late (rather than early, as originally given) version. The precise definition is given in [24].

The authors distinguish between two types of causality: *subject* and the *object*. To capture the first one, they introduce a causal term $\texttt{K} :: A$, where K is a set of causes recording that every action performed by A depends on K. The object causality is defined on the run (trace) of a process in such a way that once a bound name been extruded, it causes all the subsequent actions using that name in any position of the label. Since an action can be caused through the *subject* and *object* position of a label, the contextual cause is a set $K \subseteq \mathscr{K}_*$. For example, let us consider a process $\nu a(\nu b(\overline{c}b \mid \overline{d}a \mid \overline{b}a))$ with a trace $\xrightarrow{\overline{c}\langle \nu b\rangle} \xrightarrow{\overline{d}\langle \nu a\rangle} \xrightarrow{\overline{b}a}$. The action $\overline{b}a$ depends on the first action because with it name b was extruded and on the second action because with it name a was extruded. It is important to remark that a silent action does not exhibit causes.

To capture Boreale-Sangiorgi late semantics we need to give definitions for the predicates in Figure 3.

Definition 11 (Boreale-Sangiorgi causal semantics)**.** *If an indexed set Γ_w is chosen as a data structure for a memory Δ, the predicates from Figure 3 are defined as:*

1. $\texttt{Cause}(\Delta,K,K') = \texttt{Cause}(\Gamma_w,K,K')$ *stands for* $K' = K \cup \{w\}$

2. $\texttt{Update}(\Delta,K,K') = \texttt{Update}(\Gamma_w,K,K')$ *stands for* $K' = K \cup \{w\}$

Let us comment on the above definition. After the first extrusion of a name, the cause is fixed and there is no possibility of choosing another cause from the set Γ. To capture this behaviour we use the key of the first extruder, say w, as the index of the set Γ. The following example explains how our framework captures Boreale-Sangiorgi causality. We shall use the same process as in Example 3.

Example 4. *Consider the process* $X = \nu a_{\emptyset_*}(\overline{b}^* a^* \mid \overline{c}^* a^* \mid a^*(x))$. *By applying rule* OPEN *and executing the first extrusion on name b, we obtain the process:*

$$\nu a_{\{i\}_i}(\overline{b}^* a^*[i,*] \mid \overline{c}^* a^* \mid a^*(x))$$

In the memory $\{i\}_i$ *the index i indicates that name a was extruded with the action i. On the process* $\overline{c}^* a^*$, *rule* OPEN *can be applied. By definition of the predicate* $\texttt{Update}(\cdot)$, *the output action is forced to add* $w = i$ *in its cause set. Similar for the process* $a^*(x)$, *by applying the rule* CAUSE REF *and definition of the predicate* $\texttt{Cause}(\cdot)$. *After two executions, we obtain the process:*

$$\nu a_{\{i,h\}_i}(\overline{b}^* a^*[i,*] \mid \overline{c}^* a^*[h,\{i,*\}] \mid a^*(x)[l,\{i,*\}])$$

In the memories $[h,\{i,*\}]$ *and* $[l,\{i,*\}]$ *we see that both executed actions are caused by action i and this is why it needs to be reversed last. The second and the third action can be reversed in any order.*

Crafa, Varacca and Yoshida causal semantics. The authors introduced a compositional event structure semantics for the forward π-calculus [9]. They represent a process as a pair $(E,\mathtt{X})$, where E is a prime event structure and $\mathtt{X}$ is a set of bound names. Disjunctive objective causality is represented in such a way that an action with extruded name in the subject position can happen if at least one extrusion of that name has been executed before. In the case of parallel extrusions of the same name, an action can be caused by any of them, but it is not necessary to remember which one.

Consequently, events do not have a unique causal history. As discussed in [11] this type of disjunctive causality cannot be expressed when we consider processes with a contexts. To adapt this notion of causality to reversible settings we need to keep track of causes; otherwise by going backwards we could reach an undefined state (where the extruder of a name is reversed, but not the action using that name in the subject position).

We consider two possibilities for keeping track of causes: the first one is by choosing one of the possible extruders and the second one is recording all of them. In the first case, we would obtain a notion of causality similar to the one introduced in [10]. In the following we shall concentrate on the second option. The idea is that, since we do not know which extruder really caused the action on an extruded name, we shall record the whole set of extruders that happened previously. In the framework, the set of executed extruders is set Ω. The extrusions which are part of synchronisations will be deleted from Ω with the operation #.

The predicates from the rules of Figure 3 are defined as follows:

Definition 12 (Disjunctive causality)**.** *If an indexed set* Γ_Ω *is chosen as a data structure for a memory* Δ, *the predicates are defined as:*

1. $\texttt{Cause}(\Delta,K,K') = \texttt{Cause}(\Gamma_\Omega,K,K')$ *stands for* $K' = K \cup \Omega$
2. $\texttt{Update}(\Delta,K,K') = \texttt{Update}(\Gamma_\Omega,K,K')$ *stands for* $K' = K$

In the following example we shall give the intuition of how our framework captures the defined notion of causality.

Example 5. *Let us consider the process* $X = \nu a_{\emptyset_{\{*\}}}(\overline{b}^* a^* \mid \overline{c}^* a^* \mid a^*(x))$. *By applying a rule* OPEN *twice and executing concurrently two extrusions on names b and c, we obtain a process:*

$$\nu a_{\{i,h\}_{\{*,i,h\}}}(\overline{b}^* a^*[i,*] \mid \overline{c}^* a^*[h,*] \mid a^*(x))$$

By definition of the predicate $\texttt{Cause}(\cdot)$*, the third action will take the whole index set* $\{*,i,h\}$ *as a set K and we get the process:*

$$\nu a_{\{i,h\}_{\{*,i,h\}}}(\overline{b}^* a[i,*] \mid \overline{c}^* a[h,*] \mid a^*(x)[l,\{*,i,h\}])$$

In the memory $[l,\{*,i,h\}]$ *we see that the first action to be reversed is the action with key l; the other two actions can be reversed in any order.*

4 Properties

In this section we shall show some properties of our framework. First we shall show that the framework is a conservative extension of standard π-calculus and that it enjoys causal consistency, a fundamental property for reversible calculi. After that, we shall prove causal correspondence between the causality induced by Boreale-Sangiorgi semantics and causality in the framework when $\Delta = \Gamma_w$.

Definition 13 (Initial and Reachable process)**.** *A reversible process X is* initial *if it is derived from a π-calculus process P where all the restricting operators are initialised and in every prefix, names are decorated with a* $*$*. A reversible process is* reachable *if it can be derived from an initial process by using the rules in Figures 2, 3 and 4.*

4.1 Correspondence with the π-calculus

We now show that our framework is a conservative extension of the π-calculus. To do so, we first define an erasing function φ that given a reversible process X, by deleting all the past information, generates a π process. Then we shall show that there is a *forward* operational correspondence between a reversible process X and $\varphi(X)$. Let $\mathscr{P}$ be the set of π-calculus processes; then we have:

Definition 14 (Erasing function)**.** *The function* $\varphi : \mathscr{X} \to \mathscr{P}$ *that maps reversible processes to the π-calculus, is inductively defined as follows:*

$$\varphi(X \mid Y) = \varphi(X) \mid \varphi(Y) \qquad \varphi(\texttt{H}[X]) = \varphi(X) \qquad \varphi(\mathbf{0}) = \mathbf{0}$$
$$\varphi(\nu a_\Delta(X)) = \varphi(X) \quad \textit{if } \texttt{empty}(\Delta) = \mathit{false} \qquad \varphi(b^j(x).\mathbf{P}) = b(x).\varphi(\mathbf{P})$$
$$\varphi(\nu a_\Delta(X)) = \nu a\ \varphi(X) \quad \textit{if } \texttt{empty}(\Delta) = \mathit{true} \qquad \varphi(\overline{b}^j a^{j'}.\mathbf{P}) = \overline{b}a.\varphi(\mathbf{P})$$

The erasing function can be extended to labels as:

$$\varphi((i,K,j) : \pi) = \varphi(\pi) \qquad \varphi(\overline{b}a) = \overline{b}a$$
$$\varphi(\overline{b}\langle \nu a_\Delta \rangle) = \overline{b}\langle \nu a \rangle \quad \textit{when } \texttt{empty}(\Delta) = \mathit{true} \qquad \varphi(b(x)) = b(x)$$
$$\varphi(\overline{b}\langle \nu a_\Delta \rangle) = \overline{b}a \quad \textit{when } \texttt{empty}(\Delta) = \mathit{false} \qquad \varphi(\tau) = \tau$$

As expected the erasing function discards the past prefixes and name restriction operators that are non-empty. Moreover, it deletes all the information about the instantiators.

Every forward move of a reversible process X can be matched in the π-calculus. To this end we use $\twoheadrightarrow_\pi$ to indicate the transition semantics of the π-calculus.

Lemma 1. *If there is a transition $X \xrightarrow{\mu} Y$ then $\varphi(X) \xrightarrow{\varphi(\mu)}_{\pi} \varphi(Y)$.*

We can state the converse of Lemma 1 as follows:

Lemma 2. *If there is a transition $P \xrightarrow{\varphi(\mu)}_{\pi} Q$ then for all reachable X such that $\varphi(X) = P$, there is a transition $X \xrightarrow{\mu} Y$ with $\varphi(Y) = Q$.*

Corollary 1. *The relation given by $(X, \varphi(X))$, for all reachable processes X, is a strong bisimulation.*

4.2 The main properties of the framework

We now prove some properties of our framework which are typical of a reversible process calculus [12, 27, 20, 10]. Most of the terminology and the proof schemas are adapted from [12, 10] with more complex arguments due to the generality of our framework. The first important property is the so-called Loop Lemma, stating that any reduction step can be undone. Formally:

Lemma 3 (Loop Lemma)**.** *For every reachable process X and forward transition $t : X \xrightarrow{\mu} Y$ there exists a backward transition $t' : Y \overset{\mu}{\rightsquigarrow} X$, and conversely.*

Before stating our main theorems we need to define the causality relation. It is defined on the general framework and it is interpreted as the union of *structural* and *object* causality.

Definition 15 (Structural cause on the keys)**.** *For every two keys i_1 and i_2 such that $i_1, i_2 \in \mathtt{key}(X)$, we let $i_1 \sqsubset_X i_2$ if $X = C[\alpha[i_1, K_1].Y]$ and $i_2 \in \mathtt{key}(Y)$.*

Once having defined structural causal relation on keys, we can extend it to transitions.

Definition 16 (Structural causality)**.** *Transition $t_1 : X \xrightarrow{(i_1,K_1,j_1):\pi_1} X'$ is a* structural cause *of transition $t_2 : X'' \xrightarrow{(i_2,K_2,j_2):\pi_2} X'''$, written $t_1 \sqsubset t_2$, if $i_1 \sqsubset_{X'''} i_2$ or $i_2 \sqsubset_X i_1$.* Structural causality*, denoted with $\sqsubseteq$, is the reflexive and transitive closure of $\sqsubset$.*

Object causality is defined directly on the transitions and to keep track of it we use the contextual cause K.

Definition 17 (Reverse transition)**.** *The reverse transition of a transition $t : X \xrightarrow{\mu} Y$, written $t^\bullet$, is the transition with the same label and the opposite direction $t^\bullet : Y \overset{\mu}{\rightsquigarrow} X$, and vice versa. Thus $(t^\bullet)^\bullet = t$.*

Definition 18 (Object causality)**.** *Transition $t_1 : X \xrightarrow{(i_1,K_1,j_1):\pi_1} X'$ is an* object cause *of transition $t_2 : X' \xrightarrow{(i_2,K_2,j_2):\pi_2} X''$, written $t_1 < t_2$, if $i_1 \in K_2$ or $i_2 \in K_1$ (for the backward transition) and $t_1 \neq t_2^\bullet$.* Object causality*, denoted with $\ll$, is the reflexive and transitive closure of $<$.*

Example 6. *Consider a process $X = \nu a_\Delta(\overline{b}^* a^* \mid \overline{c}^* a^* \mid a^*(z))$ and the case when $\Delta = \emptyset_*$, as in Example 4. The executed actions would be $\xrightarrow{(i_1,*,*):\overline{b}\langle \nu a_{\emptyset_*} \rangle} \xrightarrow{(i_2,\{i_1\},*):\overline{c}\langle \nu a_{\{i_1\}_{i_1}} \rangle} \xrightarrow{(i_3,\{i_1\},*):a(z)}$. We can notice that $K_2 = \{i_1\}$ and $K_3 = \{i_1\}$, indicating that the second and the third action are caused by the first one. By choosing a different data structure we can obtain different causal order, as mentioned in Example 3 and Example 5.*

Definition 19 (Causality relation and concurrency)**.** *The causality relation $\prec$ is the reflexive and transitive closure of structural and object cause: $\prec = (\sqsubseteq \cup \ll)^*$. Two transitions are* concurrent *if they are not causally related.*

Concurrent transitions can be permuted, and the commutation of transitions is preserved up to label equivalence.

Definition 20 (Label equivalence)**.** *Label equivalence, $=_\lambda$, is the least equivalence relation satisfying: $(i,K,j) : \overline{b}\langle \nu a_\Delta \rangle =_\lambda (i,K,j) : \overline{b}\langle \nu a_{\Delta'} \rangle$ for all i,j,K,a,b and $\Delta,\Delta' \subseteq \mathscr{K}$. (Having an indexed set Γ_w for Δ we disregard index w, and observe $\Gamma \subseteq \mathscr{K}$.)*

Lemma 4 (Square Lemma)**.** *If $t_1 : X \xrightarrow{\mu_1} Y$ and $t_2 : Y \xrightarrow{\mu_2} Z$ are two concurrent transitions, there exist $t_2' : X \xrightarrow{\mu_2'} Y_1$ and $t_1' : Y_1 \xrightarrow{\mu_1'} Z$ where $\mu_i =_\lambda \mu_i'$.*

We shall follow the standard notation and say that t_2 is a residual of t_2' after t_1, denoted with $t_2 = t_2'/t_1$. Two transitions are *coinitial* if they have the same source, *cofinal* if they have the same target, and *composable* if the target of one is the source of the other. A sequence of pairwise composable transitions is called a *trace*, written as $t_1;t_2$. We denote with ε the *empty* trace. Notions of target, source, composability and reverse extend naturally to traces.

With the next theorem we prove that reversibility in our framework is causally consistent.

Definition 21 (Equivalence up-to permutation)**.** *Equivalence up-to permutation, $\sim$, is the least equivalence relation on the traces, satisfying:*

$$t_1;(t_2/t_1) \sim t_2;(t_1/t_2) \qquad t;t^\bullet \sim \varepsilon$$

Equivalence up-to permutation introduced in [12] is an adaptation of equivalence between traces introduced in [21, 7] that additionally erases from a trace, transitions triggered in both directions. It just states that concurrent actions can be swapped and that a trace made by a transition followed by its inverse is equivalent to the empty trace.

Theorem 1. *Two traces are coinitial and cofinal if and only if they are equivalent up-to permutation.*

4.3 Correspondence with Boreale and Sangiorgi's semantics

We prove causal correspondence between Boreale and Sangiorgi's late semantics (rather than early, as originally given) and the framework when memory Δ is instantiated with Γ_w. The precise definitions and the proofs are given in [24]; here we shall give just a brief presentation of the idea.

To compare semantics, we observe traces (runs) of the processes. Labels in the framework will bring additional information about the multiset of the structural causes (K_F) of the executed action and a trace in the framework will have the following form: $X_1 \xrightarrow[K_{F1}]{\mu_1} \!\!\!\!\rightarrow X_2 \ldots X_{n-1} \xrightarrow[K_{Fn}]{\mu_n} \!\!\!\!\rightarrow X_n$. To simplify notation, we shall write the transition $A \xrightarrow[K;k]{\pi} A_2$ from Boreale and Sangiorgi's semantics as $A \xrightarrow[K_B]{\zeta} A_2$, where $\zeta = k : \pi$.

Focusing on the structural correspondence, the main difference is in the silent actions. In the framework, silent actions are identified with unique keys, while in Boreale and Sangiorgi's semantics, they just merge the cause sets of the actions participating in the communication. Hence, we need to provide a connection between sets of structural causes in these two semantics. Let us briefly explain our method; more details can be found in [24].

Suppose that we have two transition t and t', where $t : X \xrightarrow[K_F]{(i,K,j):\pi} \!\!\!\!\rightarrow X'$ and $t' : A \xrightarrow[K_B]{i:\pi} A'$ where the continuation of the processes X and A is $\pi.P$[1]. We can represent the dependences between the keys in the history of the process X (all executed actions in X) with a directed graph, in the following way: keys of executed actions will be represented as vertices of a graph (actions which are part of a communication and have the same key, will be represented by two vertices with the same name); order between keys will be

[1] By abuse the notation, we shall write π for the prefixes and the labels of the actions in both semantics, since they are essentially the same

represented with directed edges where between the same vertices we shall have edges in both directions. Let as denote this graph $G = (V,E)$, where V is a multiset of vertices and E set of edges.

To show exact correspondence between cause sets K_F and K_B we need to take the history part of the process X involved in the execution of an action π. We can do it by taking all the paths in G in which the target vertex will be key i of the action π and we shall obtain the graph $G(i) = (V(i), E(i))$. The multiset $V(i)$ contains all the keys which cause the action π including i and we can conclude that $K_F = V(i) \setminus \{i\}$. By removing all bidirectional edges from the graph $G(i)$ and replacing vertices that they connect with vertex renamed to τ_l when $l = 1,2,\ldots n$, we shall obtain the graph $G' = (V', E')$. (Renaming is applied also on the other edges containing removed vertices. The operation of bidirectional edge contraction is precisely defined in [24].) The set V' differs from $V(i)$ in having τ_l vertices instead of the pairs of the vertices with the same name (originally belonging to silent moves in the framework). Hence, we can conclude that $K_B = V' \setminus (\{i\} \cup \tau_l)$.

We shall call the algorithm explained above 'Removing Keys from a Set', denoted as `Rem`. We shall write $\texttt{Rem}(K_F) = K_B$, meaning that K_B can be obtained by applying algorithm `Rem` to K_F.

Before stating the theorem, we shall give a definition of the erasing function λ and the function γ that maps labels from the framework into labels from Boreale and Sangiorgi's semantics:

Definition 22. *The function γ that maps label from the framework with a label from Boreale and Sangiorgi's semantics, is inductively defined as follows:*

$$\gamma((i,K,j):\pi) = i:\gamma(\pi) \quad \textit{when } \pi \neq \tau \qquad \gamma((i,*,*):\tau) = \tau$$
$$\gamma(\overline{b}\langle \nu a_\Delta \rangle) = \overline{b}\langle \nu a \rangle \quad \textit{when } \texttt{empty}(\Delta) = \textit{true} \qquad \gamma(b(c)) = b(c)$$
$$\gamma(\overline{b}\langle \nu a_\Delta \rangle) = \overline{b}a \quad \textit{when } \texttt{empty}(\Delta) = \textit{false} \qquad \gamma(\overline{b}a) = \overline{b}a$$

Definition 23. *The erasing function λ that maps causal processes from Boreale and Sangiorgi's semantics to the π-calculus is inductively defined as follows:*

$$\lambda(A \mid A') = \lambda(A) \mid \lambda(A') \qquad \lambda(\mathtt{K} :: A) = \lambda(A) \qquad \lambda(\nu a(A)) = \nu a(\lambda(A)) \qquad \lambda(P) = P$$

Now we have all necessary definitions to state the lemma about structural correspondence between two causal semantics.

Lemma 5 (Structural correspondence)**.** *Starting from initial π-calculus process P, where $P = A_1 = X_1$, we have:*

1. *if $P \xrightarrow[K_{B1}]{\zeta_1} A_2 \ldots A_n \xrightarrow[K_{Bn}]{\zeta_n} A_{n+1}$ then there exists a trace $P \xrightarrow[K_{F1}]{\mu_1} X_2 \ldots X_n \xrightarrow[K_{Fn}]{\mu_n} X_{n+1}$ and K_{Fi}, such that for all i, $\lambda(A_i) = \varphi(X_i)$, $\zeta_i = \gamma(\mu_i)$ and $\texttt{Rem}(K_{Fi}) = K_{Bi}$, for $i = 1,\ldots,n$.*
2. *if $P \xrightarrow[K_{F1}]{\mu_1} X_2 \ldots X_n \xrightarrow[K_{Fn}]{\mu_n} X_{n+1}$ then there exists a trace $P \xrightarrow[K_{B1}]{\zeta_1} A_2 \ldots A_n \xrightarrow[K_{Bn}]{\zeta_n} A_{n+1}$ where for all i, $\lambda(A_i) = \varphi(X_i)$, $\zeta_i = \gamma(\mu_i)$ and $\texttt{Rem}(K_{Fi}) = K_{Bi}$, for $i = 1,\ldots,n$.*

Considering the object dependence we have that the first action which extrudes a bound name will cause all future actions using that name in any position of the label. The main difference is that object dependence induced by input action in Boreale and Sangiorgi's semantics is subject dependence as well.

The next theorem demonstrates causal correspondence between causality in the framework when memory Δ is instantiated with Γ_w and Boreale and Sangiorgi's late causal semantics.

Theorem 2 (Causal correspondence)**.** *The reflexive and transitive closure of the causality introduced in [6] coincides with the causality of the framework when $\Delta = \Gamma_w$.*

5 Conclusions

In a concurrent setting, causally-consistent reversibility relates causality and reversibility. Several works [10, 15, 6, 9, 8] have addressed causal semantics for π-calculus, differing on how object causality is modelled. Starting from this observation, we have devised a framework for reversible π-calculus which abstracts away from the underlying data structure used to record causes and consequences of an extrusion, and hence from the object causality. Depending on the underlying data structure, we can obtain different causal semantics. We have shown how three different semantics [10, 6, 9] can be derived, and we have proved causal correspondence with the semantics introduced in [6]. Our framework enjoys typical properties for reversible process algebra, such as loop lemma and causal consistence. As a future work we plan to prove causal correspondence with the semantics [10, 9] and to continue working towards a more parametric framework and to compare it with [26, 18]. Moreover it would be interesting to implement our framework in the psi-calculi framework [4], and to develop further the behavioural theory of our framework.

Acknowledgments

We are grateful to the EXPRESS/SOS reviewers for their useful remarks and suggestions which led to substantial improvements.

References

[1] L. Aceto (1994): *GSOS and Finite Labelled Transition Systems*. *Theor. Comput. Sci.* 131(1), pp. 181–195, doi:10.1016/0304-3975(94)90094-9.

[2] A. Avizienis, J.-C. Laprie, B. Randell & C.E. Landwehr (2004): *Basic Concepts and Taxonomy of Dependable and Secure Computing*. *IEEE Trans. Dependable Sec. Comput.* 1(1), pp. 11–33, doi:10.1109/TDSC.2004.2.

[3] G. Bacci, V. Danos & O. Kammar (2011): *On the Statistical Thermodynamics of Reversible Communicating Processes*. In: *CALCO 2011*, *LNCS* 6859, Springer, pp. 1–18, doi:10.1007/978-3-642-22944-2_1.

[4] J. Bengtson, M. Johansson, J. Parrow & B. Victor (2011): *Psi-calculi: a framework for mobile processes with nominal data and logic*. *Logical Methods in Computer Science* 7(1), doi:10.2168/LMCS-7(1:11)2011.

[5] C.H. Bennett (1973): *Logical Reversibility of Computation*. *IBM Journal of Research and Development* 17(6), doi:10.1147/rd.176.0525.

[6] M. Boreale & D. Sangiorgi (1998): *A Fully Abstract Semantics for Causality in the π-Calculus*. *Acta Inf.* 35(5), pp. 353–400, doi:10.1007/s002360050124.

[7] G. Boudol & I. Castellani (1988): *Permutation of transitions: An event structure semantics for CCS and SCCS*. In: *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, *LNCS* 354, Springer, pp. 411–427, doi:10.1007/BFb0013028.

[8] N. Busi & R. Gorrieri (1995): *A Petri Net Semantics for pi-Calculus*. In: *CONCUR Philadelphia, PA, USA, August 21-24, 1995, Proceedings*, pp. 145–159, doi:10.1007/3-540-60218-6_11.

[9] S. Crafa, D. Varacca & N. Yoshida (2012): *Event Structure Semantics of Parallel Extrusion in the Pi-Calculus*. In: *FOSSACS 2012*, *LNCS* 7213, Springer, pp. 225–239, doi:10.1007/978-3-642-28729-9_15.

[10] I.D. Cristescu, J. Krivine & D. Varacca (2013): *A Compositional Semantics for the Reversible π-Calculus*. In: *LICS 2013*, pp. 388–397, doi:10.1109/LICS.2013.45.

[11] I.D. Cristescu, J. Krivine & D. Varacca (2015): *Rigid Families for CCS and the π-calculus*. In: *ICTAC*, *LNCS* 9399, Springer, pp. 223–240, doi:10.1007/978-3-319-25150-9_14.

[12] V. Danos & J. Krivine (2004): *Reversible Communicating Systems*. In: *CONCUR 2004*, *LNCS* 3170, Springer, pp. 292–307, doi:10.1007/978-3-540-28644-8_19.

[13] V. Danos & J. Krivine (2005): *Transactions in RCCS*. In: *CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005*, pp. 398–412, doi:10.1007/11539452_31.

[14] V. Danos & J. Krivine (2007): *Formal Molecular Biology Done in CCS-R*. *Electr. Notes Theor. Comput. Sci.* 180(3), pp. 31–49, doi:10.1016/j.entcs.2004.01.040.

[15] P. Degano & C. Priami (1999): *Non-Interleaving Semantics for Mobile Processes*. *Theor. Comput. Sci.* 216(1-2), pp. 237–270, doi:10.1016/S0304-3975(99)80003-6.

[16] E. Giachino, I. Lanese & C.A. Mezzina (2014): *Causal-Consistent Reversible Debugging*. In: *FASE 2014*, *LNCS* 8411, Springer, pp. 370–384, doi:10.1007/978-3-642-54804-8_26.

[17] J. Grattage (2005): *A Functional Quantum Programming Language*. In: *LICS*, IEEE Computer Society, Washington, DC, USA, pp. 249–258, doi:10.1109/LICS.2005.1.

[18] T.T. Hildebrandt, C. Johansen & H. Normann (2017): *A Stable Non-interleaving Early Operational Semantics for the Pi-Calculus*. In: *LATA*, *LNCS* 10168, pp. 51–63, doi:10.1007/978-3-319-53733-7_3.

[19] I. Lanese, M. Lienhardt, C.A. Mezzina, A. Schmitt & J.-B. Stefani (2013): *Concurrent Flexible Reversibility*. In: *ESOP 2013*, pp. 370–390, doi:10.1007/978-3-642-37036-6_21.

[20] I. Lanese, C.A. Mezzina & J.-B. Stefani (2016): *Reversibility in the higher-order π-calculus*. *Theor. Comput. Sci.* 625, pp. 25–84, doi:10.1016/j.tcs.2016.02.019.

[21] Jean-Jacques Lévy (1976): *An Algebraic Interpretation of the $\lambda\beta$K-Calculus; and an Application of a Labelled λ -Calculus*. *Theor. Comput. Sci.* 2(1), pp. 97–114, doi:10.1016/0304-3975(76)90009-8.

[22] D. Medic & C.A. Mezzina (2016): *Static VS Dynamic Reversibility in CCS*. In: *Reversible Computation RC 2016*, *LNCS* 9720, Springer, pp. 36–51, doi:10.1007/978-3-319-40578-0_3.

[23] D. Medic & C.A. Mezzina (2017): *Towards Parametric Causal Semantics in π-calculus*. In: *Joint Proceedings of the 18th Italian Conference on Theoretical Computer Science and the 32nd Italian Conference on Computational Logic, Naples, Italy, September 26-28.*, pp. 121–125.

[24] D. Medic, C.A. Mezzina, I.C.C. Phillips & N. Yoshida (2018): *A Parametric Framework for Reversible π-Calculi*. *ArXiv e-prints*. Available at `http://arxiv.org/abs/1807.11800`.

[25] R. Milner (1980): *A Calculus of Communicating Systems*. *LNCS* 92, Springer, doi:10.1007/3-540-10235-3.

[26] R. Perera & J. Cheney (2017): *Proof-relevant π-calculus: a constructive account of concurrency and causality*. *Mathematical Structures in Computer Science*, pp. 1–37, doi:10.1017/S096012951700010X.

[27] I.C.C. Phillips & I. Ulidowski (2007): *Reversing algebraic process calculi*. *J. Log. Algebr. Program.* 73(1-2), pp. 70–96, doi:10.1016/j.jlap.2006.11.002.

[28] I.C.C. Phillips, I. Ulidowski & S. Yuen (2013): *Modelling of Bonding with Processes and Events*. In: *Reversible Computation - RC 2013*, *LNCS* 7948, Springer, pp. 141–154, doi:10.1007/978-3-642-38986-3_12.

[29] D. Sangiorgi & D. Walker (2001): *The Pi-Calculus - a Theory of Mobile Processes*. Cambridge Uni. Press.

[30] M.V. Zelkowitz (1973): *Reversible Execution*. *Commun. ACM* 16(9), pp. 566–, doi:10.1145/362342.362360.