Information and Computation 275 (2020) 104644

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

A parametric framework for reversible π-calculi

Doriana Medić [a], Claudio Antares Mezzina [b,*], Iain Phillips [c], Nobuko Yoshida [c]

[a] *Focus Team, University of Bologna/Inria, France*
[b] *Dipartimento di Scienze Pure e Applicate, Università di Urbino, Italy*
[c] *Imperial College London, United Kingdom*

ARTICLE INFO

Article history:
Received 6 October 2020
Accepted 30 October 2020
Available online 12 November 2020

Keywords:
Causal semantics
π-Calculus
Causally-consistent reversibility

ABSTRACT

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. Hence, by using a parametric way of bookkeeping the order and the dependencies among extruders it is possible to map different causal semantics into the same framework. Starting from this simple observation, we present a uniform framework for reversible π-calculi that is *parametric* with respect to a data structure that stores information about the 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 prove causal-consistency for the three instances of our framework. Furthermore, we prove a causal correspondence between the appropriate instances of the framework and the Boreale-Sangiorgi semantics and an operational correspondence with the reversible π-calculus causal semantics.

1. Introduction

Starting from the 1970s [1] reversible computing has attracted interest in different fields, from thermodynamical physics [2] to systems biology [3,4], system debugging [5–7] and quantum computing [8]. Of particular interest is its application to the study of programming abstractions for reliable systems: most fault-tolerant schemes exploiting system recovery techniques [9] rely on some form of *undo*. Examples of how reversibility can be used to model transactions or transactional memory can be found in [10–12]. An example of how controlled reversibility in π-calculus [11] can be used to model checkpoint/rollback schemas is given in [13].

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, since due to concurrency there might be different actions happening at the same time. 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 [14]. 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 [15], 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

* Corresponding author.
E-mail address: claudio.mezzina@uniurb.it (C.A. Mezzina).

https://doi.org/10.1016/j.ic.2020.104644
0890-5401/

point of view there exists only one reversible CCS. Evidence for this has been given first [16], where an equivalence is shown between the two methods for reversing CCS (namely RCCS [14] and CCSK [17]). Lately in [18] it has been shown that RCCS and CCSK are equivalent in terms of LTS isomorphism.

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 $\nu a\,(\overline{b}a \mid a(z))$ the input action on a depends on the output on b. There are different interpretations of the object dependency which are easily distinguished when the parallel extrusion of the same name and dynamic creation of extruders via name passing is considered. Intuitions about the three approaches will be given using a recurring example throughout the paper. We abstract away from the techniques for keeping track of causality and consider just the order between the actions.

Let us consider a π-calculus process $P = \nu a(\overline{b}a \mid \overline{c}a \mid a(x)) \mid b(y).(\overline{y}d \mid \overline{e}y)$. After the synchronisation on the channel b, we obtain a π-calculus process $P' = \nu a(\overline{c}a \mid a(x) \mid \overline{a}d \mid \overline{e}a)$ where the new extruder $\overline{e}a$ of the name a is enabled. Now, we observe the two actions executing on the channel a. We have three different possibilities.

The classical and the most used approach to causality in the π-calculus is the one where the order of extrusions matters, hence the first output executed by process P', say $\overline{c}a$, will cause both actions $a(x)$ and $\overline{a}d$. Moreover, the first output will cause the action $\overline{e}a$ as well. The actions $a(x)$ and $\overline{a}d$ can communicate with each other without any constraints. Some of the causal semantics representing this idea are [19–21] and all of them are defined for standard (forward-only) π-calculus. In [19] 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 [20,21]. Hence we group these semantics together as a single approach to causality.

Secondly, in [22], the actions $a(x)$ and $\overline{a}d$ depend on one of the extruders, $\overline{c}a$ and $\overline{e}a$ executed concurrently, but there is no need to keep track of which one exactly. Additionally, the action $\overline{a}d$ is structurally dependent on the communication which happened on channel b. This causal semantics is defined for the forward-only π-calculus.

Finally, [23] introduces another interpretation of parallel extrusion by resorting to a compositional causal semantics for the reversible π-calculus in [23]. Concerning the common example, the extrusions $\overline{c}a$ and $\overline{e}a$ are executed concurrently. If the action $a(x)$ synchronises with the environment, it chooses its cause to be one of the extruders. Otherwise, if it communicates with the process $\overline{a}d$ its cause needs to be the action that instantiates the name a in process $\overline{a}d$, which is the very first action, synchronisation on the channel b. The idea behind this is that the causality of the visible actions is an anticipation of the causality of synchronisations they are involved in. This causal semantics enjoys certain correctness properties which are not satisfied by the other semantics.

Some correctness criteria for causal models have been proposed in [24] where it was shown that the approach of [23] satisfies them while those of [19,20] do not. However, there is a question about causality and π-calculus which still has to be solved. In [25] it has been shown that the (forward) causal semantics induced by the reversible higher-order π-calculus [26] coincides with the one of [20] when considering reduction semantics (e.g., considering closed systems). That is, the causality considered by [26] coincides with the structural one. One could conclude that while considering closed systems the causal semantics of [26] and [23] are the same, even if [26] does not keep track of causal information about the instantiators. One interpretation of this fact is that [23] uses more causal information than required. But this is still left to be proven. We believe that a common framework will help us compare them.

The novelty of [23] in handling extrusion for reversible π is to endow an extruded name with a "history", that is, a set of process identifiers which extruded that particular name. In this way it is possible to keep track of all the possible causes and eventually choose one. By starting from this idea, if we had to model the causality of [20], in which the first extruder causes all the subsequent actions, then we could endow the extruded name with an ordered list of extruders, in which naturally the order plays a relevant role. Then by starting from this idea, we present a framework for reversible π-calculi that is parametric with respect to the data structure that stores information about the 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. For the three instances of the framework, we provide full proof of causal-consistency, which was never provided for reversible semantics using the causality of [20, 22]. 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 [27], where some initial ideas were given. Moreover in [27] it was argued that it was necessary to modify the semantics of [20] in order to add information about silent actions. In this work we fully develop the idea behind the framework and leave the semantics of [20] unchanged, apart from using a late semantics, rather than early as originally given.

Contributions. We present a framework for reversible π-calculi which is parametric in the bookkeeping data structure used to keep track of the object dependency. As reversing technique, we will extend the one introduced by CCSK [17], which is limited to calculi defined with GSOS [28] 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 [23]). Depending on the bookkeeping data structure used to instantiate the

framework, we can obtain different causal semantics (i.e., [23,20,22]). We then show that the three corresponding instances enjoy the standard properties for a reversible calculus, namely the loop lemma and causal consistency. In particular, this is a novel result for reversible calculi whose underlying causal semantics is one of [20,22]. Also, we prove causal correspondence between the causal semantics introduced in [20] and the matching instance of our framework. Also, we prove a correspondence in terms of strong back and forth bisimulation between an instance of our framework and [23]. This is due to the fact that we use a different notion of concurrent/conflicting transitions with respect to [23].

The rest of the paper is structured as follows: an informal presentation of the framework is given in Section 2, while its syntax and operational semantics are given in Sections 3 and 4, respectively. The properties of the framework are given in Section 5 (full proofs can be found in Appendix A). In Section 6, we show how by using different data structures we can encompass different causal semantics. The operational correspondence between the matching instance of our framework and the causal semantics of [23] is presented in Section 6.1.1 (full proofs and additional examples can be found in Appendix B), while the correspondence with the semantics of [20] is given in Section 6.2.3 (full proofs are gathered in Appendix C). Section 7 concludes the paper.

This paper is a revised and enhanced version of [29]. In particular:

- Section 2 has been revised and expanded;
- in Sections 3 and 4 the framework has been extended with recursion. Examples 1 and 2 have been revised and more explanations have been added;
- Section 5 has been extended with Examples 3–6, which are not present in [29]. Moreover, a novel notion of concurrent/conflicting transitions is presented, which differs from the one of [23];
- Section 6 has been totally revised: more commentary on the various semantics has been added and Examples 8, 13 and 18 have been revised;
- Section 6.1.1 is new. It presents an encoding from our framework to $R\pi$ [23] and then shows a tight correspondence between a term of our framework (instantiated with the causality of $R\pi$) and its encoding in $R\pi$;
- the proofs in Section 6.2.3 have been revised and Examples 14–17 are added for a better understanding of proofs ideas;
- Appendix A, Appendix B and Appendix C include full proofs.

2. Informal presentation

A general approach to deriving a reversible variant of a CCS-like calculi defined through SOS rules is given in [17]. The main ideas behind this approach are to make each operator of a calculus static and to use communication keys to uniquely identify the actions. While dynamic operators as choice or prefix are forgetful operators, in the static one, there is no loss of information. For instance, if we consider a CCS process $P = a.b \mid \overline{a}.c$ the synchronisation between parallel components of the process P is:

$$a.b \mid \overline{a}.c \xrightarrow{\tau} b \mid c$$

As we can see, the information about the synchronising prefixes is lost. This is due to the fact that the prefixing operator is *dynamic*. By following the approach of [17] we can make the prefixing operator reversible in this way:

$$a.b \mid \overline{a}.c \xrightarrow{\tau[i]} a[i].b \mid \overline{a}[i].c$$

As we can notice, prefixes a and $\overline{a}$ are not discarded but annotated with a communication key i. Decorated prefixes are used only for the backward steps, hence the resulting process can continue forward computing behaving as process $b \mid c$. Hence we could have that the process can for example produce the label c:

$$a[i].b \mid \overline{a}[i].c \xrightarrow{c[j]} a[i].b \mid \overline{a}[i].c[j]$$

or it can revert the synchronisation to get back to the original process:

$$a[i].b \mid \overline{a}[i].c \xrightarrow{\tau[i]} a.b \mid \overline{a}.c$$

We expand this idea in the more complex setting of the π-calculus, where the possibility of creating new channel names and treating channels as sent values is enabled. For instance, by adapting the process P to the π-calculus, we have the computation:

$$a(x).Q_1 \mid \overline{a}b.Q_2 \xrightarrow{i:\tau} a(x)[i].Q_1\{^{b^i}/_x\} \mid \overline{a}b[i].Q_2$$

In the substitution $\{^{b^i}/_x\}$, variable x is substituted by the name b decorated with the key i. Decorations on the names keep track of the substitutions occurring during the communications. In the example, it means that variable x is substituted with the name b in the synchronisation identified by the key i. In the example above, there is a distinction between the *past* prefix $\overline{a}b[i]$ and the instantiated name b^i. The former indicates that the action $\overline{a}b$ has been executed and given a communication key i, while the second implies that the name b was resolved via the communication indicated by i.

$$
\begin{array}{rl}
P,\ Q ::= & \mathbf{0} \mid \overline{b}a.P \mid b(x).P \mid\ P \mid Q\ \mid \nu a(P) \mid A \\
\textit{Recursion} & A \triangleq P_A \\
\mathbf{P},\ \mathbf{Q} ::= & \mathbf{0} \mid \overline{b^j}a^{j_1}.\mathbf{P} \mid b^j(x).\mathbf{P} \mid\ \mathbf{P} \mid \mathbf{Q} \mid \nu a_{\mathtt{init}_\Delta}(\mathbf{P})\ \mid \mathbf{A} \\
X,\ Y ::= & \mathbf{P} \mid\ \overline{b}^j a^{j_1}[i, K].X\ \mid\ b^j(x)[i, K].X\ \mid X \mid Y\ \mid \nu a_\Delta(X)
\end{array}
$$

Fig. 1. Syntax of the framework.

Remark 1. We could keep track of the substitution of the name differently. For example, we could record in the memory, along with the key i the process before the substitution, as is done in [25,26]. By applying this approach to the π-calculus example above, we would have the computation:

$$a(x).Q_1 \mid \overline{a}b.Q_2 \xrightarrow{i:\tau} a(x)[i, Q_1].Q_1\{^b/_x\} \mid \overline{a}b[i].Q_2$$

As we can notice, the input prefix is annotated with both key and the process Q_1 before the substitution. This approach is memory consuming but it allows one to revert substitution (which is not a bijection) in a simple way by just restoring the past process. By just keeping track of substitutions via decorations, we avoid keeping a copy of each process before an input action.

In π-calculus, by adopting the static way of bookkeeping the past actions that processes performed as in [17], we avoid using the structural rule of $R\pi$ [23] to split a parallel composition. In what follows we will refer to this rule as the Split rule. In $R\pi$, processes with a computational history are called *monitored* processes. A monitored process has the form $m \triangleright P$. One drawback of this approach is the need for the split rule to enable a parallel composition of processes sharing the same memory to perform a forward action. In the structural rule

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

memory is duplicated and annotated with $\langle\uparrow\rangle$ recording the fact that in the past a process split in two parallel ones. This rule is not associative and, as shown in [25], brings an undesired feature in which equivalent processes performing the same action may become non-equivalent processes.

Besides keeping track of the actions executed in the past of the process, the framework has to remember also the actions which extruded a certain name. Following the idea introduced in [23], this can be done by using the contextual cause of an action. For example, in the computation

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

we can notice that after the extrusion of the name a (the first action performed), the restriction νa is not discarded as in the standard π-calculus, but transformed into the memory $\nu a_{\{i\}}$. In this way, the fact that name a is extruded by the action i, is recorded. The memory $\nu a_{\{i\}}$ does not behave as a restriction operator anymore; hence name a is free and the input action on the channel a can be executed. The contextual cause of the action $a(x)$ is i and this is recorded in the process $a(x)[j, i].P$.

3. Syntax

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 variables. Moreover, we let $\mathcal{K}_* = \mathcal{K} \cup \{*\}$ where $*$ is a special key. We let a, b, c range over $\mathcal{N}$; x, y range over $\mathcal{V}$; i and its decorated variants range over $\mathcal{K}$, j and its decorated variants range over $\mathcal{K}_*$.

The syntax of the framework is depicted in Fig. 1. *Processes*, given by the P, Q productions, are the standard processes of the π-calculus [30]: $\mathbf{0}$ represents the idle process; $\overline{b}a.P$ is the *output*-prefixed process indicating the act of sending name a over a 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. The symbol A represents a process constant defined as $A \triangleq P_A$, where P_A is a π-calculus process. It is assumed that each process identifier A has a *single* defining equation of the form $A \triangleq P$.

Reversible processes, given by the X, Y productions, are defined on top of π-calculus processes. Differently from the standard π-calculus, performed actions are not discarded, but annotated and kept in the structure of a process, representing its *history*. A reversible process $\mathbf{P}$ behaves as a standard π-calculus process P, only decorated with instantiators. The instantiators are used to keep track of the substitutions. The prefix $\overline{b}^j a^{j_1}[i, K]$ is called a *past output* and it records the fact that in the past, the process performed an output action identified by key i and that the contextual cause set of the executed actions was $K \subseteq \mathcal{K}_*$. Prefix $b^j(x)[i, K].X$ represents a *past input* recording the fact that the executed action was the input action identified by key i and its contextual cause set was K. Parallel composition of two reversible processes X and Y is

represented with $X \mid Y$. Inspired by [23], the restriction operator νa_Δ is decorated with the memory Δ which keeps track of the extruders of the name a. When Δ is empty ($\texttt{empty}(\Delta) = \texttt{true}$—we will give the precise definition below), νa_Δ will act as the classical restriction operator νa of the π-calculus.

In the framework we use π-calculus processes P decorated with instantiators, written as $\mathbf{P}$; we do the same for process constants A. Then, we define $\mathbf{A}$ as $\mathbf{A} \triangleq \mathbf{P_A}$.

Process definition is instrumental in adding (reversible) recursion in a neat way, as shown in [31], as it does not require us to keep track of the substituted process variable. We denote the set of reversible processes with $\mathcal{X}$.

Remark 2. If the prefix of the process is not relevant, we will denote it with π. Hence, we have $\pi = \overline{b}^j a^{j_1}$ or $\pi = b^j(x)$. We shall use notation b^* in the subject or the object position to specify that name b has no instantiator. We will use this technicality in Definition 8 when we define the notion of an initial process in the framework. We denote with $K = \{*\}$ the fact that there is no action that caused the executing action, while $K = \{*, i\}$ or $K = \{i\}$ symbolises the fact that the executing action is caused by the action identified with the key i. Later in this section and in Section 4, we shall see that $K = \{*, i\}$ and $K = \{i\}$ have the same meaning, but the difference appears as a consequence of the fact that different definitions for the data structure Δ use different machinery to record the causes (Definitions 14–16).

Remark 3. The framework can be easily equipped with the choice operator $(+)$ by making the operator static as in [17]. Adding the choice operator would result just in a more complex syntax, but all the results will still hold as the choice operator does not affect the notion of causality.

To simplify the manipulation of reversible processes, we shall define *history* and *general* contexts. A history context represents the executed prefixes of a process. For instance, the process $X = \overline{b}^* a^*[i, K].\overline{c}^* a^*[i', K'].\mathbf{P}$ can be written as $X = \mathtt{H}[\mathbf{P}]$, where $\mathtt{H} = \overline{b}^* a^*[i, K].\overline{c}^* a^*[i', K'].\bullet$. On top of the history context, we define a general context by adding parallel and restriction operators. For instance, process $Z \mid Y \mid X$ can be written as $C[X]$ where $C = Z \mid Y \mid \bullet$. Formally, we have:

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:

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

Definition 2 *(Set of free names).* The set of free names of a given process X, written as $\mathtt{fn}(X)$, is defined as:

$$\mathtt{fn}(X \mid Y) = \mathtt{fn}(X) \cup \mathtt{fn}(Y)$$
$$\mathtt{fn}(\nu a_\Delta(X)) = \mathtt{fn}(X) \setminus \{a\} \text{ if } \mathtt{empty}(\Delta) = \mathtt{true}$$
$$\mathtt{fn}(\nu a_\Delta(X)) = \mathtt{fn}(X) \cup \{a\} \text{ if } \mathtt{empty}(\Delta) \neq \mathtt{true}$$
$$\mathtt{fn}(\overline{b}^j a^{j'}.X) = \mathtt{fn}(X) \cup \{b, a\}$$
$$\mathtt{fn}(a^j(x).X) = \mathtt{fn}(X) \cup \{a\}$$

The set of bound names of the process X is defined as $\mathtt{bn}(X) = \mathtt{n}(X) \setminus \mathtt{fn}(X)$ where $\mathtt{n}(X)$ denotes the set of all names appearing in process X.

Definition 3 *(Set of free variables).* The set of free variables of the given process X, written as $\mathtt{fv}(X)$, is defined as:

$$\mathtt{fv}(X \mid Y) = \mathtt{fv}(X) \cup \mathtt{fv}(Y) \qquad \mathtt{fv}(\overline{b}^j a^{j'}.X) = \mathtt{fv}(X)$$
$$\mathtt{fv}(\nu a_\Delta(X)) = \mathtt{fv}(X) \qquad \mathtt{fv}(a^j(x).X) = \mathtt{fv}(X) \setminus \{x\}$$

The set of bound variables of the process X is defined as $\mathtt{bv}(X) = \mathtt{v}(X) \setminus \mathtt{fv}(X)$ where $\mathtt{v}(X)$ denotes the set of all variables appearing in process X.

Given a process X, we represent the set of its free names and variables, written $\mathtt{Fnv}(X)$ as $\mathtt{Fnv}(X) = \mathtt{fn}(X) \cup \mathtt{fv}(X)$. Similarly, the set of bound names and variables is written as $\mathtt{Bnv}(X)$.

3.1. Abstract data types

The framework is parametric with respect to the data structure Δ which we define as an interface (in the style of a Java interface) by giving the operations that it has to offer.

Definition 4. The symbol Δ represents a data structure defined with the following operations:

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

(ii) $\texttt{empty} : \Delta \to \texttt{bool}$ predicate telling whether Δ is empty
(iii) $+ : \Delta \times \mathcal{K} \to \Delta$ operation adding a key to Δ
(iv) $_{\#i} : \Delta \times \mathcal{K} \to \Delta$ operation removing a key from Δ
(v) $\in : \mathcal{K} \times \Delta \to \texttt{bool}$ predicate telling whether a key belongs to Δ

In what follows, we give a description and definitions of operations defined on the three instances of the data structure Δ: set, set indexed with an element and set indexed with a set. As we shall see, these three instances will give rise to three different notions of causality.

Set. The data structure is a set Γ containing keys (i.e. $\Gamma \subset \mathcal{K}_*$) of all the actions that extruded name a. The idea behind the memory νa_Γ is that any of the keys contained in Γ can be a contextual cause for some action having in the subject position name a. For example, in the process $\nu a_{\{i_1,i_2\}}(Y \mid a(x))$, the contextual cause of the action $a(x)$ can be chosen from the set $\Gamma = \{i_1, i_2\}$.

Remark 4. In what follows we will abuse notation and write $X_{\#i}$, that is, we apply the operator $_{\#i}$ to a process instead of a data structure Δ. The meaning of $X_{\#i}$ is that the operator will be applied *only* on the data structures Δ contained in X.

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

(i) $\texttt{init}(\Gamma) = \emptyset$
(ii) $\texttt{empty}(\Gamma) = \texttt{true}$, when $\Gamma = \emptyset$
(iii) $+$ is the addition of elements to a set
(iv) $_{\#i}$ is defined as the identity, that is $\Gamma_{\#i} = \Gamma$
(v) $i \in \Gamma$ means that the key i belongs to the set Γ

The data structure is initialised when $\Gamma = \emptyset$ and it implies that $\texttt{empty}(\Gamma) = \texttt{true}$, as expected. The operation $_{\#i}$ is defined as the identity, while $+$ and $i \in \Gamma$ are classical operations defined on sets.

Indexed set. The data structure is an indexed set Γ_w, where set Γ is a set containing keys ($\Gamma \subset \mathcal{K}_*$) of all the actions that extruded name a and w is the key of the very first action that extruded name a. In this case the contextual cause for name a is exactly w. For instance in the process $\nu a_{\{i_1,i_2\}_{i_1}}(Y \mid a(x))$ the contextual cause of the action $a(x)$ is i_1. We shall write $w = *$ if there is no action that extruded name a.

Definition 6 *(Operations on an indexed set).* The operations on an indexed set Γ_w are defined as:

(i) $\texttt{init}(\Gamma_w) = \emptyset_*$
(ii) $\texttt{empty}(\Gamma_w) = \texttt{true}$, when $\Gamma = \emptyset \wedge w = *$
(iii) $+$ 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) $_{\#i}$ is defined as: $(\Gamma_w)_{\#i} = \begin{cases} (\Gamma_w)_{\#i} = \Gamma_*, & \textit{when } i = w \\ (\Gamma_w)_{\#i} = \Gamma_w, & \textit{otherwise} \end{cases}$
(v) $i \in \Gamma_w$ means that the key i belongs to the set Γ, regardless of w (e.g. $i \in \{i\}_*$)

The data structure is initialised when $\texttt{init}(\Gamma_w) = \emptyset_*$ and it implies that $\texttt{empty}(\Gamma_w) = \texttt{true}$, as expected. The $+$ operator is defined as: the key added to Γ_w will be added to the set Γ and in the place of w if $w = *$, otherwise it will be added just to the set Γ. For instance, after adding key i_3 to $\{i_1, i_2\}_*$ we obtain $\{i_1, i_2, i_3\}_{i_3}$. Operation $_{\#i}$ substitutes the value of w in Γ_w with element $*$, when $w = i$. For example, the result of applying operation $_{\#i}$ on $\{i, i_1\}_i$ is $\{i, i_1\}_*$. The operation of belonging is defined on the set Γ in the classical way, regardless of the index w. For instance, the key i belongs to the indexed set $\{i_1, i\}_{i_1}$.

Set indexed with a set. The data structure is a set indexed with a set Γ_Ω, where Γ is a set containing keys of all the actions that extruded name a and $\Omega \in \mathcal{K}_*$ is a set containing keys of the extruders of name a which are not part of the communication. The idea behind νa_{Γ_Ω} is that the contextual cause for the name a is the set Ω. For example, in the process $\nu a_{\{i_1,i_2,i_3\}_{\{*,i_1,i_3\}}}(Y \mid a(x))$ the contextual cause of the action $a(x)$ is $\Omega = \{*, i_1, i_3\}$. We write $\Omega = \{*\}$ when there is no action that extruded name a which is not part of the synchronisation.

Definition 7 *(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) $\texttt{empty}(\Gamma_\Omega) = \texttt{true}$, when $\Gamma = \emptyset \wedge \Omega = \{*\}$
(iii) $+$ is defined as: $(\Gamma_\Omega) + i = (\Gamma \cup \{i\})_{(\Omega \cup \{i\})}$
(iv) $_{\#i}$ is defined as: $(\Gamma_\Omega)_{\#i} = \Gamma_{\Omega \setminus \{i\}}$
(v) $i \in \Gamma_\Omega$ means that the key i belongs to the set Γ, regardless of Ω (e.g. $i \in \{i\}_{\{*\}}$)

The data structure is initialised when $\texttt{init}(\Gamma_\Omega) = \emptyset_{\{*\}}$ and it implies that $\texttt{empty}(\Gamma_{\{*\}}) = \texttt{true}$. The $+$ operator is defined by adding the key into both sets. For instance, the result of adding key i into the data structure $\{i_1, i_2\}_{\{*,i_2\}}$ is $\{i_1, i_2, i\}_{\{*,i_2,i\}}$. The operation $_{\#i}$ removes the key i from the set Ω. For example, if we apply operation $_{\#i}$ to the data structure $\{i_1, i_2, i\}_{\{*,i_2,i\}}$, we obtain $\{i_1, i_2, i\}_{\{*,i_2\}}$ (the key i is deleted from the set $\Omega = \{*, i_2, i\}$). The operation of belonging is defined on the set Γ in the classical way, regardless of the set Ω. For instance, the key i belongs to the data structure $\{i_1, i\}_{\{*,i_1\}}$.

In the following we define the initial process in the framework.

Definition 8 *(Initial process).* A reversible process $X = \mathbf{P}$ is *initial* if it is derived from a π-calculus process P where all the restriction operators are initialised (i.e. restrictions are decorated with $\texttt{init}(\Delta)$) and in every prefix, names are decorated with a distinguished symbol $*$.

4. Operational semantics of the framework

The grammar of the labels defined on the transition $t : X \xrightarrow{\mu} Y$ is:

$$\mu ::= (i, K, j) : \alpha \qquad \alpha ::= \overline{b}a \mid b(x) \mid \overline{b}\langle \nu a_\Delta \rangle \mid \tau$$

The triple (i, K, j) contains the key i that identifies the action α, the contextual cause set $K \subseteq \mathcal{K}_*$ and the instantiator $j \in \mathcal{K}_*$ of the action α. The action α can be:

- standard input and output on the channel b, symbolised with $b(x)$ and $\overline{b}a$, respectively;
- the silent action; or
- action $\overline{b}\langle \nu a_\Delta \rangle$ that represents the classical bound output from the π-calculus, when Δ is empty ($\texttt{empty}(\Delta) = \texttt{true}$), otherwise stands for free output decorated with a memory Δ.

We say that name b is in the subject position of the action α, written as $\texttt{subj}(\alpha) = b$ if $\alpha \in \{\overline{b}a, b(x), \overline{b}\langle \nu a_\Delta \rangle\}$. Similarly, name b is in the object position, written as $\texttt{obj}(\alpha) = b$ if $\alpha \in \{\overline{a}b, \overline{a}\langle \nu b_\Delta \rangle\}$. We write $\texttt{bn}$ and $\texttt{bv}$ for the set of bound names and variables of the action α:

$$\texttt{bn}(\overline{b}a) = \emptyset \qquad \texttt{bv}(\overline{b}a) = \emptyset$$
$$\texttt{bn}(a(x)) = \emptyset \qquad \texttt{bv}(a(x)) = \{x\}$$
$$\texttt{bn}(\overline{b}\langle \nu a_\Delta \rangle) = \{a\} \text{ when } \texttt{empty}(\Delta) = \texttt{true} \qquad \texttt{bv}(\overline{b}\langle \nu a_\Delta \rangle) = \emptyset$$
$$\texttt{bn}(\overline{b}\langle \nu a_\Delta \rangle) = \emptyset \text{ when } \texttt{empty}(\Delta) \neq \texttt{true}$$

With $\texttt{Bnv}(\alpha) = \texttt{bn}(\alpha) \cup \texttt{bv}(\alpha)$ we denote the set of bound names and variables in the action α.

Now we define the function $\texttt{key}(\cdot)$ which computes the set of keys in a given process and we specify the notion of freshness for a key.

Definition 9 *(Process keys).* The set of *communication keys* of a process X, written $\texttt{key}(X)$, is inductively defined as follows:

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

Definition 10. A key i is *fresh* in a process X, written $\texttt{fresh}(i, X)$ if $i \notin \texttt{key}(X)$.

With the next definition we define the relation between the cause set and the instantiators, which is necessary to define the communication rules.

Definition 11. Given a cause set K and an instantiator j, we have

$$K \bowtie j \quad \text{if} \quad K = \{j\} \quad \text{or} \quad * \in K$$

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

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

$$(\textsc{Par})\ \frac{X \xrightarrow{(i,K,j):\alpha} X' \quad i \notin Y \quad \texttt{Bnv}(\alpha) \cap \texttt{Fnv}(Y) = \emptyset}{X \mid Y \xrightarrow{(i,K,j):\alpha} X' \mid Y}$$

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

$$(\textsc{Rec})\ \frac{\mathbf{P_A} \xrightarrow{(i,K,j):\alpha} X' \quad \mathbf{A} \triangleq \mathbf{P_A}}{\mathbf{A} \xrightarrow{(i,K,j):\alpha} X'}$$

Fig. 2. Common rules for all instances of the framework.

For instance if $K = \{*, i_1, i_2\}$ and $j = i_3$ the relation $K \bowtie j$ holds since we have $* \in K$. We will come back to this relation when explaining the rule (Com).

We let $\mathcal{A}$ be the set of all actions ranged over by α. The set of all possible labels is defined as $\mathcal{L} = \mathcal{K} \times \mathcal{P}(\mathcal{K}_*) \times \mathcal{K}_* \times \mathcal{A}$, where $\mathcal{P}(\mathcal{K}_*)$ denotes the power set of $\mathcal{K}_*$. In the following definition, we give the operational semantics of the framework.

Definition 12 *(Operational semantics).* The operational semantics of the reversible framework is given as a pair of LTSs defined on the same set of reversible processes $\mathcal{X}$ and set of labels $\mathcal{L}$: a forward LTS $(\mathcal{X}, \mathcal{L}, \twoheadrightarrow)$ and a backward LTS $(\mathcal{X}, \mathcal{L}, \rightsquigarrow)$. We define $\rightarrow = \twoheadrightarrow \cup \rightsquigarrow$, where $\twoheadrightarrow$ is the least transition relation induced by the rules in Figs. 2 and 3 (pages 8 and 10 respectively); and $\rightsquigarrow$ is the least transition relation induced by the rules in Fig. 4 (page 11).

In what follows, we give the forward and backward rules of our framework defined with a *late* semantics for inputs. Forward rules are divided into two groups, depending on whether they are common to all the instances of the framework (rules which are independent of the data structure) or they are parametric with respect to Δ.

Common rules are given in Fig. 2, where $\texttt{H}$ is a history context (Definition 1). As we can notice, in the rules (Out1) and (In1), executed actions remain in the structure of the process and they are annotated with the memory $[i, K]$, where i is the fresh key bound to the executing action and K is the cause set of the action. The label of the transition except the key i and cause set K contains also the instantiator j of the name b. A prefixed process $\texttt{H}[X]$ can perform a forward step if process X can execute it. This is depicted by the rules (Out2) and (In2). The rule for parallel composition (Par) allows a process to execute the action α, under the condition that the key i is not used by another process in parallel ($i \notin Y$). This condition guarantees uniqueness of the action keys.

Rule (Com) allows synchronisation between two processes in parallel which satisfy the condition $K \bowtie j' \wedge K' \bowtie j$. The idea behind the condition is the following one: having two premises $p_1 : X \xrightarrow{(i,K,j):\overline{b}a} X'$ and $p_2 : Y \xrightarrow{(i,K',j'):b(x)} Y'$, if name b in the premise p_1 is instantiated with the action j, then in the premise p_2, there are two possibilities for the cause set K' of the action $b(x)$ so that condition $K' \bowtie j$ holds, namely $* \in K'$ or $K' = \{j\}$. This condition is necessary when we want to capture the semantics of [23].[1] As we will see in Section 6.1, to capture it, the set Γ will be used as the data structure and the cause set K will be the singleton. Then the condition $K' \bowtie j$ is satisfied if $* \in \{K', j\}$ or $K' = \{j\}$. For the other semantics that we considered, K is the set obtained by adding the index of the used data structure (w or Ω) to it, every time when a certain restriction is passed (we will give more details in Section 6). Therefore, since in the initial process $w = *$ and $\Omega = \{*\}$, we have $* \in K$, which makes the condition on the rule (Com) always true.

Additionally, in the rule (Com), the necessary substitution is applied to the continuation of the input process in the following way: every occurrence of variable $x \in \texttt{Fnv}(Y')$ is substituted with the name a decorated with the key i of the executed action. In this way, future computations of the process $Y'\{^{a^i}/_x\}$ will be aware that variable x was substituted with the name a during the synchronisation identified by i. In a^i, the key i is called the *instantiator* and used only to track the substitution, not to define a name. For instance, in the reversible process $\overline{b}^{j}a^{*}.\mathbf{P} \mid b^{j'}(x).\mathbf{P}'$, the communication between $\overline{b}^{j}a^{*}.\mathbf{P}$ and $b^{j'}(x).\mathbf{P}'$ is allowed even if they do not have the same instantiators on the channel b.

Rule (Rec) allows recursive computations. Following [17] we use process constants to define recursive processes. This is simpler than dealing with the recursion operator **rec** $X.P$ since there is no need of keeping track of the substitution of the

[1] In the semantics of [23], the condition on the communication rule is $k =_* j' \wedge k' =_* j$, where $k =_* j$ is defined as $* \in \{k, j\}$ or $k = j$. We adapt this definition to our framework.

process variable. For example, if we take the usual rule for the recursive process:

$$(\textsc{Rec1})\ \frac{P\{^{\mathbf{rec}\,X.P}/_{X}\} \xrightarrow{\alpha} P'}{\mathbf{rec}\,X.P \xrightarrow{\alpha} P'}$$

in our framework, we should have the following forward and backward rules

$$(\textsc{Rec1})\ \frac{P\{^{\mathbf{rec}\,X.P}/_{X}\} \xrightarrow{(i,K,j):\alpha} P'}{\mathbf{rec}\,X.P \xrightarrow{\alpha} \mathbf{rec}\,X[i,P].P'} \qquad (\textsc{Rec1}^{\bullet})\ \frac{P' \overset{(i,K,j):\alpha}{\rightsquigarrow} P''}{\mathbf{rec}\,X[i,P].P' \overset{(i,K,j):\alpha}{\rightsquigarrow} \mathbf{rec}\,X.P}$$

which keep track of the fact that the process variable X has been substituted in P by the recursive definition, because of the action identified by i. For the same reason, we also do not consider constants which are parametric to a set of names, that is $A(x_1,\ldots,x_n) \triangleq P$. By resorting to process constants, we avoid keeping track of such information.

For example, let $\mathbf{A} \triangleq \overline{b}^*a^*.\mathbf{A} \mid \overline{c}^*d^*$ be a process constant. The execution of the output action on the channel b is:

$$\overline{b}^*a^*.\mathbf{A} \mid \overline{c}^*d^* \xrightarrow{(i_1,\{*\},*):\overline{b}a} \overline{b}^*a^*[i_1,\{*\}].\mathbf{A} \mid \overline{c}^*d^*$$

Now we can substitute $\mathbf{A}$ with $\overline{b}^*a^*.\mathbf{A} \mid \overline{c}^*d^*$ and execute the output action on the channel b again:

$$\overline{b}^*a^*[i_1,\{*\}].(\overline{b}^*a^*.\mathbf{A} \mid \overline{c}^*d^*) \mid \overline{c}^*d^* \xrightarrow{(i_2,\{*\},*):\overline{b}a} \overline{b}^*a^*[i_1,\{*\}].(\overline{b}^*a^*[i_2,\{*\}].\mathbf{A} \mid \overline{c}^*d^*) \mid \overline{c}^*d^*$$

In order to have better intuition about the rules presented above, we give the following example.

Example 1. Let $X = \overline{b}^*a^*.\mathbf{0} \mid b^*(x).\overline{x}c^*$ be a reversible process. Process X has two possibilities of executing forward actions: (i) either the two prefixes synchronise with the environment, (ii) or they synchronise with each other.

(i) An output action $\overline{b}a$ and an input action $b(x)$ can be performed. In this case actions synchronise with the 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'.

(ii) Two parallel components of the process X can synchronise over the channel b, and we have:

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

We can notice that during the synchronisation identified with key i, variable x was substituted with the received name a decorated with the key i. In this way substitution of a name is recorded.

Parametric rules are represented in Fig. 3. Depending on the data structure used, the mechanism for choosing a contextual cause differs. For this reason, we introduce two new predicates `Cause`$(\cdot)$ and `Update`$(\cdot)$. The intuition behind the predicates is that they define how the contextual cause set is chosen from the memory Δ. In the following, we give the definitions of the mentioned predicates for three data structures that we will use.

To be able to define the predicate `Cause`$(\cdot)$ when data structure $\Delta = \Gamma$ is used, we first need to adapt the definition of the instantiation relation ([23, Definition 2.2]) to our framework and define it on the past prefixes. For instance in the process

$$b^*(x)[i_1,K_1].\overline{a}^{i_1}c^*[i_2,K_2].Y$$

actions i_1 and i_2 are in instantiation relation, since action i_1 instantiated the name a. We can see it in the past prefix $\overline{a}^{i_1}c^*[i_2,K_2]$ where name a is decorated with the key i_1. Formally, the instantiation relation on the prefixes is defined as follows.

Definition 13 *(Instantiation relation on the framework).* Two keys i_1 and i_2 such that $i_1, i_2 \in \mathtt{key}(X)$ and $X{=}C[b^{j_1}(x)[i_1,K_1].Y]$ with $Y = C'[\pi[i_2,K_2].Z]$ where $j_2 \in \pi$, are in *instantiation relation*, denoted with $i_1 \rightsquigarrow_X i_2$, if $j_2 = i_1$. If $i_1 \rightsquigarrow_X i_2$ holds, we will write $K_1 \rightsquigarrow_X K_2$.

Now we can give the definition of the predicates.

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

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

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

$$(\textsc{Res})\ \frac{X \xrightarrow{(i,K,j):\alpha} X' \quad a \notin \alpha}{\nu a_\Delta(X) \xrightarrow{(i,K,j):\alpha} \nu a_\Delta(X')}$$

Fig. 3. Parametric rules.

Definition 14 *(Predicates when $\Delta = \Gamma$).* If the data structure Δ is instantiated with the set Γ, the predicates from Fig. 3 are defined as:

1. $\mathtt{Cause}(\Gamma, K, K', X)$ stands for $K' = K$ if $K \subseteq \Gamma$ or $K' \subseteq \Gamma$ such that $K \rightsquigarrow_X K'$;
2. $\mathtt{Update}(\Gamma, K, K')$ stands for $K' = K$.

The predicate $\mathtt{Update}(\Gamma, K, K')$, used in rule (Open) means that the new cause K' is the same as the old one, denoted with K. The first predicate defined above, $\mathtt{Cause}(\Gamma, K, K', X)$, is used in the rule (Cause Ref) and it defines how the new contextual cause K' is chosen. In particular, in the rule (Cause Ref) when passing a restriction νa_Γ, the contextual cause set K might be preserved (i.e. $K' = K$) if $K \subseteq \Gamma$ or it can be replaced by the new cause set $K' \subseteq \Gamma$ such that $K \rightsquigarrow_X K'$. We illustrate this with examples in Section 6.1.

Definition 15 *(Predicates when $\Delta = \Gamma_w$).* If the data structure Δ is instantiated with indexed set Γ_w, the predicates from Fig. 3 are defined as:

1. $\mathtt{Cause}(\Gamma_w, K, K', X)$ stands for $K' = K \cup \{w\}$
2. $\mathtt{Update}(\Gamma_w, K, K')$ stands for $K' = K \cup \{w\}$

We can notice that the predicates $\mathtt{Cause}(\cdot)$ and $\mathtt{Update}(\cdot)$ define the new cause set K' by adding the value of w to the old set of causes K.

Definition 16 *(Predicates when $\Delta = \Gamma_\Omega$).* If an indexed set Γ_Ω is chosen as a data structure Δ, the predicates are defined as:

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

In the definition above we can notice that the new cause K' in the predicate $\mathtt{Cause}(\Gamma_\Omega, K, K', X)$, gathers all extrusions of a restricted name executed previously which are not part of the synchronisations.

Let us comment on the rules in Fig. 3. Every time when action α, with $a \in \alpha$, passes the restriction νa_Δ, it is necessary to check if the contextual cause set needs to be modified. If name a is in the subject position of the label α and $\mathtt{empty}(\Delta) = \mathtt{false}$, then rule (Cause Ref) is used, otherwise, if name a is in the object position, rule (Open) is applied. Rule (Cause Ref) can be used only if name a was already extruded by some other action in the past and in this case predicate $\mathtt{Cause}(\Delta, K, K', X)$ ensures that the contextual cause set K will be substituted with a new cause set K'. Additionally, predicate $\mathtt{Cause}(\Delta, K, K', X)$ gives the definition of the cause set K'. The *contextual cause update* operation defined on the process X, written as $X_{[K'/K]@i}$, updates the contextual cause K of the action identified by i with the new cause K'. Formally:

Definition 17 *(Contextual cause update).* The *contextual cause update* of the process X, written $X_{[K'/K]@i}$ is defined as follows:

$$(X \mid Y)_{[K'/K]@i} = X_{[K'/K]@i} \mid Y_{[K'/K]@i}$$
$$(\nu a_\Delta(X))_{[K'/K]@i} = \nu a_\Delta(X)_{[K'/K]@i}$$
$$(\pi[h, K_1].X)_{[K'/K]@i} = \pi[h, K_1].X_{[K'/K]@i} \text{ if } K_1 \neq K$$
$$(\pi[i, K].\mathbf{P})_{[K'/K]@i} = \pi[i, K'].\mathbf{P}$$

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

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

$$(\textsc{Par}^\bullet)\ \frac{X' \overset{(i,K,j):\alpha}{\rightsquigarrow} X \quad i \notin Y}{X' \mid Y \overset{(i,K,j):\alpha}{\rightsquigarrow} X \mid Y} \qquad (\textsc{Res}^\bullet)\ \frac{X' \overset{(i,K,j):\alpha}{\rightsquigarrow} X \quad a \notin \alpha}{\nu a_\Delta(X') \overset{(i,K,j):\alpha}{\rightsquigarrow} \nu a_\Delta(X)}$$

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

$$(\textsc{Open}^\bullet)\ \frac{X' \overset{(i,K,j):\alpha}{\rightsquigarrow} X \quad \alpha = \overline{b}a \vee \alpha = \overline{b}\langle \nu a_{\Delta'}\rangle \quad \texttt{Update}(\Delta,K,K')}{\nu a_{\Delta+i}(X') \overset{(i,K',j):\overline{b}\langle \nu a_\Delta\rangle}{\rightsquigarrow} \nu a_\Delta(X)}$$

$$(\textsc{Cause Ref}^\bullet)\ \frac{X' \overset{(i,K,j):\alpha}{\rightsquigarrow} X \quad a \in \texttt{subj}(\alpha) \quad \texttt{empty}(\Delta) \neq \texttt{true} \quad \texttt{Cause}(\Delta,K,K',X)}{\nu a_\Delta(X') \overset{(i,K',j):\alpha}{\rightsquigarrow} \nu a_\Delta(X)}$$

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

$$(\textsc{Rec}^\bullet)\ \frac{X' \overset{(i,K,j):\alpha}{\rightsquigarrow} \mathbf{P_A} \quad \mathbf{A} \triangleq \mathbf{P_A}}{X' \overset{(i,K,j):\alpha}{\rightsquigarrow} \mathbf{A}}$$

Fig. 4. Backward rules.

We use this operation in rules (Cause Ref) and (Open). A restricted name can be sent out to the environment (extruded) by applying the rule (Open). In this case, we need to record what was the key of the action that extruded a restricted name. For this reason key i is added into the memory Δ. The predicate $\texttt{Update}(\Delta, K, K')$ in the rule (Open) defines what will be the new contextual cause set K'. Rule (Close) allows synchronisation between two processes when bound output is included. An additional condition on the rule needs to be satisfied. In the resulting process, after execution of the τ-action, we can notice the operator $_{\#i}$. As we saw in Section 3, different data structures require different implementations of it. The operator $_{\#i}$ removes from the data structure Δ, the keys of the actions that extruded a restricted name and are part of synchronisations.[2] Therefore, we need to apply the operation $_{\#i}$ on every memory of the form $\nu a_{\Delta'}$ in the resulting process X'. The operation is not applied on the very first element νa_Δ since by construction of the rules (Close) and (Open), the key i cannot belong to Δ. Rule (Res) is defined in the usual way. Better intuition of how parametric rules work will be given by means of examples in Section 6.

Backward rules are presented in Fig. 4 and they are symmetric to the forward ones. These rules could be made simpler by just dropping all the predicates in the premises, since consistency of backward steps is guaranteed by memories and keys. Nonetheless, to simplify the proofs, we keep the predicates, even if they are not necessary for the backward transition.

Backward rule ($\textsc{Rec}^\bullet$) allows one to syntactically put back a constant identifier $\mathbf{A}$ provided that the process $\mathbf{P_A}$ fully reverted its computation.

In order to have a better understanding of the backward rules, we shall give the following example.

Example 2. Let us consider the process $Y_2 = \overline{b}^* a^*[i,\{*\}].\mathbf{0} \mid b^*(x)[i,\{*\}].\overline{a}^i c^*$. The only possible backward step for process Y_2 is undoing the synchronisation done over the channel b:

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

As we can notice, the substitution is also reversed and name a^i is substituted with variable x.

[2] This is necessary, since in causal semantics [20,22], τ-actions do not bring the causal information.

5. 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 then that it enjoys causal-consistency, a fundamental property for reversible calculi. Most of the terminology and the proof schemas follow those of [14,23], with more complex arguments due to the generality of our framework.

All properties defined on the framework are limited to the reachable processes given with the following definition.

Definition 18 *(Reachable process).* A reversible process is *reachable* if it can be derived from an initial process by using the rules in Figs. 2, 3 and 4.

5.1. 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 $\varphi(X)$. The proof of the correspondence with the π-calculus is simpler than the one given in [23] for $R\pi$. The reason for this is that we do not use explicit substitutions logged in the event labels as [23]; instead we use direct substitution in the continuation of the input process participating in the communication. Let $\mathcal{P}$ be the set of π-calculus processes; then we have:

Definition 19 *(Erasing function).* The function $\varphi : \mathcal{X} \to \mathcal{P}$ that maps reversible processes to the π-calculus, is inductively defined as follows:

$$\varphi(X \mid Y) = \varphi(X) \mid \varphi(Y) \qquad \varphi(\overline{b}^{j} a^{j'}.\mathbf{P}) = \overline{b}a.\varphi(\mathbf{P})$$

$$\varphi(\nu a_{\Delta}(X)) = \varphi(X) \quad \text{if } \mathtt{empty}(\Delta) = \mathtt{false} \qquad \varphi(b^{j}(x).\mathbf{P}) = b(x).\varphi(\mathbf{P})$$

$$\varphi(\nu a_{\Delta}(X)) = \nu a\, \varphi(X) \quad \text{if } \mathtt{empty}(\Delta) = \mathtt{true} \qquad \varphi(\mathbf{0}) = \mathbf{0}$$

$$\varphi(\mathtt{H}[X]) = \varphi(X) \quad \text{if } \mathtt{H} \neq \bullet$$

The erasing function can be extended to labels as:

$$\varphi((i, K, j) : \alpha) = \varphi(\alpha) \qquad \varphi(\overline{b}a) = \overline{b}a$$

$$\varphi(\overline{b}\langle \nu a_{\Delta} \rangle) = \overline{b}\langle \nu a \rangle \quad \text{when } \mathtt{empty}(\Delta) = \mathtt{true} \qquad \varphi(b(x)) = b(x)$$

$$\varphi(\overline{b}\langle \nu a_{\Delta} \rangle) = \overline{b}a \quad \text{when } \mathtt{empty}(\Delta) = \mathtt{false} \qquad \varphi(\tau) = \tau$$

As expected, the erasing function discards the past prefixes and name restriction operators when Δ is 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 $\varphi(X) \xrightarrow{\varphi(\mu)}\!\!\!\!\twoheadrightarrow_{\pi} \varphi(Y)$.*

Proof. The proof is by induction on the derivation tree of the transition $X \xrightarrow{\mu} Y$ with the case analysis on the last applied rule. □

Now we give the converse of Lemma 1.

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

Proof. The proof is by induction on the derivation tree of the transition $P \xrightarrow{\varphi(\mu)}\!\!\!\!\twoheadrightarrow_{\pi} Q$. □

By combining the two previous lemmata we can state that the relation between a process X and its corresponding π term P is a strong bisimulation [30, Definition 2.2.1]. Formally:

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

5.2. The main properties of the framework

In this section we shall prove some properties of our framework which are typical of a reversible process calculus [14, 17,25,23]. Such properties hold for the three instances 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.*

Proof. The proof follows from the symmetry of the forward and the backward rules. □

Using the symmetry of the rules, we define reverse transitions.

Definition 20 *(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 $(t^\bullet)^\bullet = t$.

An important component for reversibility is a *causal relation* between the actions. It is possible to reverse an action α only if all the actions that were caused by α, were reversed previously. In this way, one avoids reaching states that are not consistent. For instance, in the reversible process $\nu a_{\{i_1\}}(\overline{b}a[i_1, \{*\}] \mid a(x)[i_2, \{i_1\}])$, obtained from the π-calculus process $\nu a(\overline{b}a \mid a(x))$, if the action with the key i_1 have been reversed as the first one, we would reach a state that is not consistent (it is impossible to have a state in which action with a bound subject is executed before the restricted name was extruded and became a free name).

In the following, we give a definition of the causality relation on the framework, regardless of the data structure used. It is interpreted as the union of the *structural* and the *object* causalities. We start by defining structural dependences between two past prefixes. For instance, in the reversible process $X = \overline{b}a[i, K].\overline{c}d[i', K']$, the past prefix with the key i is a structural cause of the past prefix with key i'. Formally:

Definition 21 *(Structural cause on the past prefixes).* For every two keys i_1 and i_2 such that $i_1, i_2 \in \mathtt{key}(X)$, we say that the past prefix with the key i_1 is a *structural cause* of the past prefix with the key i_2 in the process X, written as $i_1 \sqsubset_X i_2$ if $X = C[\pi[i_1, K_1].Y]$ and $i_2 \in \mathtt{key}(Y)$.

We can now extend the causal relation to work with transitions.

Definition 22 *(Structural causality).* Transition $t_1 : X \xrightarrow{(i_1, K_1, j_1):\alpha_1} X'$ is a *structural cause* of transition $t_2 : X' \xrightarrow{(i_2, K_2, j_2):\alpha_2} X''$, written $t_1 \sqsubset t_2$, if $i_1 \sqsubset_{X''} i_2$, or $i_2 \sqsubset_X i_1$ if the transitions are backward. *Structural causality*, denoted with $\sqsubseteq$, is obtained as the reflexive and transitive closure of $\sqsubset$.

To ease the understanding of the causal relation, which is a crucial key of our framework, we now give some examples.

Example 3. Let us consider the process $X = \overline{b}a.\overline{c}d$ and forward transitions:

$$t_1 : X \xrightarrow{(i,K,j):\overline{b}a} \overline{b}a[i, K].\overline{c}d \quad \text{and}$$
$$t_2 : \overline{b}a[i, K].\overline{c}d \xrightarrow{(i',K',j'):\overline{c}d} \overline{b}a[i, K].\overline{c}d[i', K']$$

In the process $X' = \overline{b}a[i, K].\overline{c}d[i', K']$, by using Definition 21, we have $i \sqsubset_{X'} i'$. Since key i identifies transition t_1 and key i' transition t_2, we have that $t_1 \sqsubset t_2$.

Example 4. Let us take the resulting process from the example above and use it as an initial state for the backward transitions, i.e. we consider process $X = \overline{b}a[i, K].\overline{c}d[i', K']$ and backward transitions:

$$t_1 : X \overset{(i',K',j'):\overline{c}d}{\rightsquigarrow} \overline{b}a[i, K].\overline{c}d \quad \text{and} \quad t_2 : \overline{b}a[i, K].\overline{c}d \overset{(i,K,j):\overline{b}a}{\rightsquigarrow} \overline{b}a.\overline{c}d$$

In the process $X = \overline{b}a[i, K].\overline{c}d[i', K']$, by using Definition 21 for the backward transitions, we have $i \sqsubset_X i'$, which implies that $t_1 \sqsubset t_2$.

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

Definition 23 *(Object causality).* Transition $t_1 : X \xrightarrow{(i_1,K_1,j_1):\alpha_1} X'$ is an *object cause* of transition $t_2 : X' \xrightarrow{(i_2,K_2,j_2):\alpha_2} X''$, 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 obtained as the reflexive and transitive closure of $<$.

Example 5. Consider the process $X = \nu a_\Delta(\overline{b}^* a^* \mid a^*(z))$ and computation:

$$X \xrightarrow{(i_1,K_1,*):\overline{b}\langle \nu a_\Delta\rangle} \xrightarrow{(i_2,K_2,*):a(z)} \nu a_{\Delta'}(\overline{b}^* a^*[i_1,K_1] \mid a^*(z)[i_2,K_2])$$

where $K_1 = \{*\}$ and $K_2 = \{i_1\}$. Then $i_1 \in K_2$ holds, and we have that the first transition is the object cause of the second one.

In what follows we will use the concept of *freed* and *consumed* prefix. A prefix is freed (resp. consumed) when it generates a backward (resp. forward) label. In other words a prefix is freed when it is put back by a backward transition, while it is consumed when it becomes a history prefix.

Definition 24 *(Causality relation).* The *causality relation* $\prec$ is the reflexive and transitive closure of structural and object causality: $\prec = (\sqsubset \cup \ll)^*$.

Now we define the notion of *conflict* between two consecutive transitions. It will consider the reverse transitions and transitions executed on the very same prefix. Formally we have:

Definition 25 *(Conflict relation).* Two consecutive transitions $t_1 : X \xrightarrow{\mu_1} X'$ and $t_2 : X' \xrightarrow{\mu_2} X''$ are in *conflict* if

- t_1 is a backward transition, t_2 is a forward one and transition t_2 consumes a prefix freed by transition t_1 or
- t_1 is a forward transition and $t_2 = t_1^\bullet$.

Using conflict and causality as defined above, we can give the notion of concurrency defined on the framework.

Definition 26 *(Concurrency relation).* Two consecutive transitions are *concurrent* if they are neither causally related nor in conflict.

We illustrate the necessity of the conflict condition between transitions (Definition 25) with the following example. Having a process

$$X = \overline{a}^* b^*[i_1,K_1] \mid \overline{a}^* b^* \mid a^*(x)$$

we execute two consecutive transitions t_1 (backward) and t_2 (forward) as:

$$t_1 : \overline{a}^* b^*[i_1,K_1] \mid \overline{a}^* b^* \mid a^*(x) \xrightarrow{(i_1,K_1,*):\overline{a}b} \overline{a}^* b^* \mid \overline{a}^* b^* \mid a^*(x)$$

$$t_2 : \overline{a}^* b^* \mid \overline{a}^* b^* \mid a^*(x) \xrightarrow{(i_2,*,*):\tau} \overline{a}^* b^*[i_2,K_2] \mid \overline{a}^* b^* \mid a^*(x)[i_2,K_2]$$

Transitions t_1 and t_2 are neither structural causal (Definition 22) nor object causal (Definition 23). If there would not be the notion of conflict (Definition 25), we would have that transitions t_1 and t_2 are concurrent, which is not the case (it is not possible to swap transitions t_1 and t_2). Let us note that according to the definition of causality and concurrency given in [23, Definition 4.1] the two transitions are not taken into account. In particular, the notion of causality of [23, Definition 4.1] is defined only for two backward or two forward transitions; it does not consider mixed transitions.

We now give some additional properties and definitions necessary to prove our main results, the Square Lemma (Lemma 5) and causal-consistency (Theorem 1). We start by showing the following property stating that in a reachable reversible process all restrictions νa_Δ of the same name a are nested.

Lemma 4. *If process $X = C[\nu a_\Delta(Y) \mid Y']$ is reachable, then $\nu a_{\Delta'} \notin Y'$, for all non-empty Δ and Δ'.*

Proof. The proof is by induction on the trace that leads to the process X: $X_1 \to \cdots \to X_n \to X$, where X_1 is an initial reversible process,[3] and the last applied rule on the transition $X_n \to X$. The full proof can be found in Appendix A. □

[3] From Definition 8 we have that the reversible process X is initial when all its names are decorated with $*$ and for all restrictions νa_Δ, for some name a, Δ is empty.

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

Definition 27 *(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 $\Delta, \Delta' \subseteq \mathcal{K}$. (Having an indexed set Γ_w for Δ we disregard index w, and observe $\Gamma \subseteq \mathcal{K}$.)

Label equivalence is necessary since actions bring information about Δ into the labels. By permuting the transitions, the content of Δ is changing. To illustrate this, let us consider the following example.

Example 6. Consider the process $X = \nu a_\emptyset(\overline{b}^* a^* \mid \overline{c}^* a^*)$ and the case when $\Delta = \Gamma$. For instance, process X can first execute output on channel b identified with the key i_1 and then output on channel c with the key i_2, and we have:

$$X \xrightarrow{(i_1,\{*\},*):\overline{b}\langle \nu a_\emptyset \rangle} \nu a_{\{i_1\}}(\overline{b}^* a^*[i_1, \{*\}] \mid \overline{c}^* a^*)$$
$$\xrightarrow{(i_2,\{*\},*):\overline{c}\langle \nu a_{\{i_1\}} \rangle} \nu a_{\{i_1,i_2\}}(\overline{b}^* a^*[i_1, \{*\}] \mid \overline{c}^* a^*[i_2, \{*\}]) = X_1$$

Now, if we execute actions in the opposite order, we have:

$$X \xrightarrow{(i_2,\{*\},*):\overline{c}\langle \nu a_\emptyset \rangle} \nu a_{\{i_2\}}(\overline{b}^* a^* \mid \overline{c}^* a^*[i_2, \{*\}])$$
$$\xrightarrow{(i_1,\{*\},*):\overline{b}\langle \nu a_{\{i_2\}} \rangle} \nu a_{\{i_1,i_2\}}(\overline{b}^* a^*[i_1, \{*\}] \mid \overline{c}^* a^*[i_2, \{*\}]) = X_2$$

We can notice that the resulting processes in both computations are the same, i.e. $X_1 = X_2$. In the labels of the transitions, we can see that $(i_1, \{*\}, *) : \overline{b}\langle \nu a_\emptyset \rangle =_\lambda (i_1, \{*\}, *) : \overline{b}\langle \nu a_{\{i_2\}} \rangle$, since the only difference is in the set Γ. Similarly for the transitions on channel c, we have $(i_2, \{*\}, *) : \overline{c}\langle \nu a_{\{i_1\}} \rangle =_\lambda (i_2, \{*\}, *) : \overline{c}\langle \nu a_\emptyset \rangle$.

Lemma 5 *(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 $\mu_i =_\lambda \mu_i'$.*

Proof. The proof is by case analysis on the form of the transitions t_1 and t_2. We consider four main cases depending on whether transitions t_1 and t_2 are synchronisations or not, and then use induction on the structure of the process while checking all possible combinations of the rules applied to the transitions t_1 and t_2. More details about the proof can be found in Appendix A. □

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 transition. 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 28 *(Equivalence up-to permutation).* Equivalence up-to permutation, $\sim$, is the least equivalence relation on the traces, satisfying:

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

Equivalence up-to permutation introduced in [14] is an adaptation of equivalence between traces introduced in [32,33] 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 *(Causal-consistency). Two traces are coinitial and cofinal if and only if they are equivalent up-to permutation.*

Proof. Let us denote the two traces with s_1 and s_2. If $s_1 \sim s_2$ then from the definition of $\sim$ (Definition 28) we can conclude that they are coinitial and cofinal. Let us suppose that s_1 and s_2 are coinitial and cofinal. We reason by induction on the lengths of s_1, s_2 and on the depth of the very first disagreement between them. The full proof can be found in Appendix A. □

6. Mapping causal semantics

In this section, we show how the causality notions induced by three different approaches [23,20,22] can be mapped in our framework. Moreover, we prove the causal / operational correspondence between matching instance of the framework and causal semantics of [20] and [23], respectively.

A key point of the semantics proposed in [23] is that it satisfies several correctness criteria for causal models [24], while the other causal semantics considered in this paper do not. Two of such criteria in [24] are particularly interesting: (i) two visible transitions are causally related if and only if for all contexts the corresponding reductions are; and (ii) causality is preserved by structural congruence. These two properties make the semantics of [23] compositional, and do not hold for the semantics of [19,20].

Nonetheless, semantics [19,20] deserve still to be studied, and having a common framework in which one can express all these semantics all together can lead to a better comparison among them and increase the understanding of causality in the π-calculus. Besides, our framework adds reversible computation to semantics originally conceived as forward only.

Even if the causal semantics proposed in [23] is the best one according to the aforementioned criteria, there is an open question about causality and π-calculus which still has to be solved. In [25] it has been shown that the (forward) causal semantics induced by the reversible higher-order π-calculus [26] coincides with the one of [20] while considering reduction semantics (e.g., considering closed systems). That is, the causality considered by [26] coincides with the structural one. The causal semantics of [26] heavily exploits structural laws for extruding a name and due to this fact there is no need to store information about the extruder of a name. The drawback is that still there exists no compositional LTS for such semantics. On the other hand, if we take the semantics of [23] and consider just a closed system, the contextual causality is always shadowed by the structural one. For instance let us consider the π-calculus process $\nu a(\overline{b}a \mid a(x)) \mid b(y).\overline{y}c$. After the synchronisations on the channels b and a, according to the semantics of [23] (where we omit the split elements in the memory), we have:

$$\nu a_{\emptyset}(\nu a_{\{1\}}(\langle 1, *, \overline{b}a\rangle \triangleright \mathbf{0} \mid \langle 2, 1, a[c/x]\rangle \triangleright \mathbf{0}) \mid \langle 2, *, \overline{y}c\rangle \cdot \langle 1, *, b[a/y]\rangle \triangleright \mathbf{0})$$

If we consider two τ-actions we can notice that the action 1 structurally causes the action 2. In Proposition 5.1 of [23], the authors stated that in a closed system, there is a canonical definition of causality, which is the structural one.

Considering the semantics of [26] we start from the configuration

$$\nu a(k_1 : b\langle a\rangle \mid a(x)) \mid k_2 : b(y) \triangleright y\langle c\rangle$$

and when we execute the same transitions, we have:

$$\nu a(\nu k, h_1, h_2, k_3.(k_3 : \mathbf{0}) \mid [M, k] \mid [M', k_3])$$

with $M = k_1 : b\langle a\rangle \mid a(x) \mid k_2 : b(y)$, $M' = (\langle h_1, \tilde{h}\rangle \cdot k : a(x)) \mid (\langle h_2, \tilde{h}\rangle \cdot k : a\langle c\rangle)$ and $\tilde{h} = \{h_1, h_2\}$. According to the definition of causality of [26] we have that the communication identified by k causes the one identified by k_3 (as in [23]).

One could conclude that while considering closed systems the causal semantics of [26] and [23] are the same, even if [26] does not keep track of causal information about the instantiators. One interpretation of this fact is that [23] uses more causal information than required. But this is still left to be proven.

For every semantics, through the common example of process

$$X = \nu a_{\mathtt{init}(\Delta)}(\overline{b}^* a^* \mid \overline{c}^* a^* \mid a^*(x))$$

we comment on the differences in the orders of the causal-consistent backward moves.

6.1. Reversible semantics for π-calculus

Cristescu et al. [23] introduced $R\pi$, a compositional reversible semantics for 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 label of 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 [23] of indexed restriction νa_{Γ} was the inspiration for our parametric indexed restriction νa_{Δ}.

In the following we show how $R\pi$ causality can be captured by the framework.

Definition 29 *($R\pi$ causality).* To capture $R\pi$ causality with the reversible framework, the data structure Δ is instantiated with the *set* Γ.

Note that since the actions in [23] can be caused only through the subject of the label, contextual cause set K is a singleton.

To illustrate how $R\pi$ causality is captured in the framework, we give the following example.

Example 7. Let us consider the process $X = \nu a_{\emptyset}(\overline{b}^* a^* \mid \overline{c}^* a^* \mid a^*(x))$ where we have the parallel extrusion of the same name a. By extruding the name a with the rule (Open) twice, on the channels b and c, we obtain a process:

$$\nu a_{\{i_1,i_2\}}(\overline{b}^* a^*[i_1,\{*\}] \mid \overline{c}^* a^*[i_2,\{*\}] \mid a^*(x))$$

The rule (Cause Ref) is used for the execution of the third action where action $a(x)$ can choose its cause from the set $\{i_1, i_2\}$. By choosing, for example, i_2 as a cause, we obtain the process:

$$\nu a_{\{i_1,i_2\}}(\overline{b}^* a^*[i_1,\{*\}] \mid \overline{c}^* a^*[i_2,\{*\}] \mid a^*(x)[i_3,\{i_2\}])$$

In the memory $[i_3, \{i_2\}]$ we can see that the action identified with key i_3 needs to be reversed before the action with key i_2. Process $\overline{b}^* a^*[i_1, \{*\}]$ can execute a backward step at any time with the rule (Open$^\bullet$).

With the following example we shall show how the instantiation relation is used in the predicate $\texttt{Cause}(\Delta, K, K', X)$ (Definition 14) while choosing the new cause K'.

Example 8. Let us consider the process $X = \nu a_{\emptyset}(\overline{b}^* a^* \mid \overline{c}^* a^* \mid a^*(x))$ from Example 7 in parallel with the process $X' = b^*(y).\overline{d}^* y$ and we are executing the synchronisation on channel b, and extrusions $\overline{c}a$ and $\overline{d}a$, identified with keys i_1, i_2 and i_3, respectively. In particular, we have:

$$\nu a_{\emptyset}(\overline{b}^* a^* \mid \overline{c}^* a^* \mid a^*(x)) \mid b^*(y).\overline{d}^* y \xrightarrow{(i_1,\{*\},*):\tau}$$

$$\nu a_{\emptyset}(\nu a_{\{i_1\}}(\overline{b}^* a^*[i_1,\{*\}] \mid \overline{c}^* a^* \mid a^*(x)) \mid b^*(y)[i_1,\{*\}].\overline{d}^* a^{i_1}) \xrightarrow{(i_2,\{*\},*):\overline{c}a}$$

$$\nu a_{\{i_2\}}(\nu a_{\{i_1,i_2\}}(\overline{b}^* a^*[i_1,\{*\}] \mid \overline{c}^* a^*[i_2,\{*\}] \mid a^*(x)) \mid b^*(y)[i_1,\{*\}].\overline{d}^* a^{i_1}) \xrightarrow{(i_3,\{*\},*):\overline{d}a}$$

$$\nu a_{\{i_2,i_3\}}(\nu a_{\{i_1,i_2\}}(\overline{b}^* a^*[i_1,\{*\}] \mid \overline{c}^* a^*[i_2,\{*\}] \mid a^*(x)) \mid b^*(y)[i_1,\{*\}].\overline{d}^* a^{i_1}[i_3,\{*\}]) = Y$$

For the sake of simplicity, we omit the history part of the process Y and we write it as $\nu a_{\{i_2,i_3\}}(\nu a_{\{i_1,i_2\}}(a^*(x)))$.

Now, let us assume that the action $a(x)$ while passing the restriction $\nu a_{\{i_1,i_2\}}$ chooses as its cause the key i_1. To pass the second restriction, we have the following rule

$$\frac{\nu a_{\{i_1,i_2\}}(a^*(x)) \xrightarrow{(i,\{i_1\},*):a(x)} \nu a_{\{i_1,i_2\}}(a^*(x)[i,\{i_1\}]) \quad \{i_3\} \subseteq \Gamma \quad \{i_1\} \rightsquigarrow_Y \{i_3\}}{\nu a_{\{i_2,i_3\}}(\nu a_{\{i_1,i_2\}}(a^*(x))) \xrightarrow{(i,\{i_3\},*):a(x)} \nu a_{\{i_2,i_3\}}(\nu a_{\{i_1,i_2\}}(a^*(x)[i,\{i_3\}]))}$$

We can notice that in the premise, the contextual cause was $K = \{i_1\}$ and it is saved in the resulting process $\nu a_{\{i_1,i_2\}}(a^*(x)[i,\{i_1\}])$. Passing the restriction $\nu a_{\{i_2,i_3\}}$, action $a(x)$ needs to take another cause, since $i_1 \notin \{i_2, i_3\}$ and the chosen cause is $K' = \{i_3\}$ since $\{i_1\} \rightsquigarrow_Y \{i_3\}$. Contextual cause i_3 is then recorded in the final process $\nu a_{\{i_2,i_3\}}(\nu a_{\{i_1,i_2\}}(a^*(x)[i,\{i_3\}]))$.

6.1.1. Correspondence with $R\pi$ causal semantics

In this section we prove the correspondence between our framework when the data structure Δ is instantiated with the set Γ and the reversible π-calculus given in [23].

The reversing technique used in [23] is an extension of the approach used in [14], where reversibility is added to CCS. To obtain reversibility in the framework, we have extended the approach introduced by CCSK [17], to work with the π-calculus. These two approaches to reverse CCS are different, as [14] employs a *dynamic* way of reversing a process by using memory stacks and splitting over a parallel composition, while the approach of [17] is *static* in the sense that all the computational history is blended directly into the process. Nonetheless, in [16] it has been shown that the two approaches are similar, by means of encodings. Hence, starting from this idea we extend the encoding of RCCS into CCSK, to work with π-calculus terms. The following example will highlight the difference between RCCS and CCSK.

Example 9. Let $P = a \mid b$ a CCS process. A process in order to be executed in RCCS needs to be monitored by a memory. In the following example, $\langle\rangle$ indicate the empty memory.

$$\langle\rangle \triangleright P \equiv \langle\uparrow\rangle \cdot \langle\rangle \triangleright a \mid \langle\uparrow\rangle \cdot \langle\rangle \triangleright b \xrightarrow{a,i} \tag{1}$$

$$\langle a, i, \mathbf{0}\rangle \cdot \langle\uparrow\rangle \cdot \langle\rangle \triangleright \mathbf{0} \mid \langle\uparrow\rangle \cdot \langle\rangle \triangleright b \tag{2}$$

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

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

$$(\textsc{Par})\ \frac{X \xrightarrow{(i,K,j):\alpha} X' \quad i \notin Y \quad \texttt{Bnv}(\alpha) \cap \texttt{Fnv}(Y) = \emptyset}{S :: X \mid Y \xrightarrow{(i,K,j):\alpha} S :: X' \mid Y}$$

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

$$(\textsc{Cause Ref})\ \frac{X \xrightarrow{(i,K,j):\alpha} X' \quad a \in \texttt{subj}(\alpha) \quad K' = K \ \vee\ K' \subseteq \Gamma \ \ K \rightsquigarrow_X K'}{S :: \nu a_\Gamma(X) \xrightarrow{(i,K',j):\alpha} S :: \nu a_\Gamma(X'_{[K'/K]@i})}$$

$$(\textsc{Open})\ \frac{X \xrightarrow{(i,K,j):\alpha} X' \quad \alpha = \overline{b}a \vee \alpha = \overline{b}\langle \nu a_{\Gamma'} \rangle}{S :: \nu a_\Gamma(X) \xrightarrow{(i,K',j):\overline{b}\langle \nu a_\Gamma \rangle} S :: \nu a_{\Gamma+i}(X'_{[K'/K]@i})}$$

$$(\textsc{Close})\ \frac{X \xrightarrow{(i,K,j):\overline{b}\langle \nu a_\Gamma \rangle} X' \quad Y \xrightarrow{(i,K',j'):b(x)} Y' \quad K \bowtie j' \ \wedge\ K' \bowtie j}{S :: X \mid Y \xrightarrow{(i,\{*\},*):\tau} S \cup \{^{a^i}/_x\} :: \nu a_\Gamma(X' \mid Y'\{^{a^i}/_x\})}$$

$$(\textsc{Res})\ \frac{X \xrightarrow{(i,K,j):\alpha} X' \quad a \notin \alpha}{S :: \nu a_\Gamma(X) \xrightarrow{(i,K,j):\alpha} S :: \nu a_\Gamma(X')}$$

Fig. 5. Semantics of the framework with the substitution set S.

where in (1) $\equiv$ is used to split the empty memory along the processes in parallel, so that they can further execute. Once the memory is split, the two processes can run. Let us suppose the left process produces the action a. First, the action is bound with a unique identifier i and then the memory of the process keeps track of the executed action, as shown in (2). If we were to reproduce the above reduction in CCSK, it is simpler as CCSK does not require initial memories:

$$a \mid b \xrightarrow{a,i} a[i] \mid b \tag{3}$$

Also, let us note that in CCSK the history is kept in the process itself, not requiring additional memories.

An important difference between the encoding given in [16] and the one that we will present here is that we need to keep track of the substitution of names and encode it. Similarly as for CCS, the difference between $R\pi$ and the framework is in how the information about past actions is stored ($R\pi$ relies on external memories and structural congruence, while in the framework the information about the past actions is spread among the structure of the process). Additionally, there is also a difference in how substitutions are handled. $R\pi$ uses explicit substitutions by relying on memories, while our framework uses implicit substitutions. Furthermore, once a process in $R\pi$ produces a label, a lookup function translates "local" names (e.g., variables) into public ones (e.g., their actual values). That is, the names are resolved and rendered "public" before interacting with the context. To check if synchronisation is possible, processes need to search in their memories for the public name of a channel. This operation is performed with rules for reducing the prefix ((IN+) and (OUT+) of [23]).

In the framework, substitutions are executed directly during the synchronisation in the continuation of the input prefix. For this reason, to simplify the causal correspondence proofs, we instrument our semantics in such a way that it keeps track of all the substitutions that have been made during the computation. Hence, a reversible process from the framework will have the form $S :: X$, where the substitution set S is attached to every reversible process X and when process is initial, we have that $S = \emptyset$. Since we are considering fresh keys and always different bound names there will be no clashes, and so the set S can be a global one. The syntax is formally defined as:

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

The semantics of the framework with substitution set S is given in Fig. 5. We instantiate the data structure Δ with the set Γ and instead of predicates $\texttt{Cause}(\cdot)$ and $\texttt{Update}(\cdot)$ use their definitions to obtain $R\pi$ causality (definition of predicates is given in Definition 14). The set S does not have any influence on the execution of the actions. The semantics

of the framework work in the same way. The only difference is that we add substitution $\{^{a^i}/_x\}$ to the set S when rules (Com) and (Close) are applied.

Remark 5.

- The set S will not have any influence on the execution of the actions in the framework; it will just collect all substitutions that happen during the computation. The framework already takes care of the substitutions; this addition to the existing semantics is just to simplify the translation of a process from the framework into an $R\pi$ process.
- We suppose that both calculi support α-conversion [30] that is a bound name/variable can be freely renamed in order to avoid name capturing.
- In the encoding of the framework into $R\pi$, we do not consider recursion as it does not appear in $R\pi$.

In what follows we will adapt the encoding from [16] to work with the π-calculus. We start by introducing a substitution function σ which will delete the substitutions which happened in the past of a process by bringing back the original names/variables and deleting the instantiators that decorate names. In this way we will be able to match the explicit substitutions used in [23]. Formally:

Definition 30. Given the substitution set S, the function σ is defined as follows:

$$\sigma(S, \mathbf{P}_1 \mid \mathbf{P}_2) = \sigma(S, \mathbf{P}_1) \mid \sigma(S, \mathbf{P}_2)$$

$$\sigma(S, \nu a_\Gamma(\mathbf{P})) = \nu a_\Gamma \sigma(S, \mathbf{P})$$

$$\sigma(b^j(x).\mathbf{P}) = \begin{cases} b(x).\sigma(S,\mathbf{P}) \text{ if for some } y & \{^{b^j}/_y\} \notin S \\ y(x).\sigma(S,\mathbf{P}) \text{ if for some } y & \{^{b^j}/_y\} \in S \end{cases}$$

$$\sigma(S, \overline{b}^j a^{j'}.\mathbf{P}) = \begin{cases} \overline{b}a.\sigma(S,\mathbf{P}) \text{ if for some } x, y & \{^{b^j}/_y\} \notin S \wedge \{^{a^{j'}}/_x\} \notin S \\ \overline{b}x.\sigma(S,\mathbf{P}) \text{ if for some } x, y & \{^{b^j}/_y\} \notin S \wedge \{^{a^{j'}}/_x\} \in S \\ \overline{y}a.\sigma(S,\mathbf{P}) \text{ if for some } x, y & \{^{b^j}/_y\} \in S \wedge \{^{a^{j'}}/_x\} \notin S \\ \overline{y}x.\sigma(S,\mathbf{P}) \text{ if for some } x, y & \{^{b^j}/_y\} \in S \wedge \{^{a^{j'}}/_x\} \in S \end{cases}$$

$$\sigma(S, \mathbf{0}) = \mathbf{0}$$

The function σ can be extended to the labels as

$$\sigma(S, b^j[*/x]) = \begin{cases} b[*/x] \text{ if for some } a^{j'}, y & \{^{b^j}/_y\} \notin S \wedge \{^{a^{j'}}/_x\} \notin S \\ b[a/x] \text{ if for some } a^{j'}, y & \{^{b^j}/_y\} \notin S \wedge \{^{a^{j'}}/_x\} \in S \\ y[*/x] \text{ if for some } a^{j'}, y & \{^{b^j}/_y\} \in S \wedge \{^{a^{j'}}/_x\} \notin S \\ y[a/x] \text{ if for some } a^{j'}, y & \{^{b^j}/_y\} \in S \wedge \{^{a^{j'}}/_x\} \in S \end{cases}$$

$$\sigma(S, \overline{b}^j a^{j'}) = \begin{cases} \overline{b}a \text{ if for some } x, y & \{^{b^j}/_y\} \notin S \wedge \{^{a^{j'}}/_x\} \notin S \\ \overline{b}x \text{ if for some } x, y & \{^{b^j}/_y\} \notin S \wedge \{^{a^{j'}}/_x\} \in S \\ \overline{y}a \text{ if for some } x, y & \{^{b^j}/_y\} \in S \wedge \{^{a^{j'}}/_x\} \notin S \\ \overline{y}x \text{ if for some } x, y & \{^{b^j}/_y\} \in S \wedge \{^{a^{j'}}/_x\} \in S \end{cases}$$

We will use the function σ while applying the encoding, to obtain the correct action label in the memory and to delete the instantiators from the process $\mathbf{P}$ (a process from the framework without past). For instance, if the set of the substitutions is $S = \{^{a^j}/_x\}, \{^{b^{j'}}/_y\}$, then if we apply $\sigma(\cdot)$ on the label of the event $\langle i, *, c^{j''}[*/x]\rangle$, we have $\langle i, *, \sigma(S, c^{j''}[*/x])\rangle = \langle i, *, c[a/x]\rangle$. If function is applied on the process $\mathbf{P} = \overline{b}^{j'} a^j.c^{j''}(z)$, we have $\sigma(S, \overline{b}^{j'} a^j.c^{j''}(z)) = \overline{y}x.\sigma(S, c^{j''}(z)) = \overline{y}x.c(z)$.

Now we can give the encoding function $[\![\cdot]\!]$, which will take two additional parameters: a memory m and the set S (the set of all substitutions applied in the past of the process X). See Fig. 6.

Let us comment on the encoding. The encoding has to go through the structure of the process from the framework in order to build the memory of the $R\pi$ process (similarly as in [16]). The encoding of a decorated π-calculus process $\mathbf{P}$ with some memory m and substitution set S is the reversible process $m \triangleright \sigma(S, \mathbf{P})$, where function σ will deal with the substitutions and delete the instantiators. If the process is not standard, then the encoding is by structural induction. A past prefix $\overline{b}^j a^{j'}[i, K]$ of the process X is encoded into a corresponding memory event $\langle i, K, \sigma(S, \overline{b}^j a^{j'})\rangle$. The parallel and the restriction operators of the framework are mapped to the corresponding operators of $R\pi$. Let us note that, in the case of

$$
\begin{aligned}
[\![S :: X]\!] &= [\![X, \langle\rangle, S]\!] \\
[\![\overline{b}^j a^{j'}[i,K].X, m, S]\!] &= [\![X, \langle i, K, \sigma(S, \overline{b}^j a^{j'})\rangle.m, S]\!] \\
[\![b^j(x)[i,K].X, m, S]\!] &= [\![X, \langle i, K, \sigma(S, b^j[*/x])\rangle.m, S]\!] \\
[\![X \mid Y, m, S]\!] &= [\![X, \langle\uparrow\rangle.m, S]\!] \mid [\![Y, \langle\uparrow\rangle.m, S]\!] \\
[\![\nu a_\Gamma(X), m, S]\!] &= \nu b_\Gamma [\![X\{^b/_a\}, m, S]\!] \\
& \quad \text{if } b \notin \mathtt{Fnv}(m) \wedge (b = a \vee b \notin \mathtt{Fnv}(X)) \\
[\![\mathbf{P}, m, S]\!] &= m \triangleright \sigma(S, \mathbf{P})
\end{aligned}
$$

Fig. 6. Encoding of the framework into $R\pi$.

parallel composition, the memory m and the substitution set S are duplicated into two identical memories $\langle\uparrow\rangle.m$ and sets S, respectively. The rule for restriction needs to avoid capturing free occurrences of a inside m: name capture is avoided by renaming a into a fresh name b. If $a \notin \mathtt{Fnv}(m)$ then one can choose $b = a$. Both the calculi feature α-conversion; hence renaming is not an issue.

To have a better intuition about the encoding let us consider the following example.

Example 10. Consider a π-calculus process $b(x).\overline{a}x.\overline{x}d \mid \overline{b}c$. After the synchronisation on the channel b and output on the channel a, the reversible process in the framework is:

$$X = S :: b^*(x)[1,\{*\}].\overline{a}^* c^1[2,\{*\}].\overline{c^1}d^* \mid \overline{b}^* c^*[1,\{*\}]$$

where the substitution set $S = \{\{^{c^1}/_x\}\}$. Then the encoding of the process X is:

$$
\begin{aligned}
[\![X]\!] &= [\![S :: b^*(x)[1,\{*\}].\overline{a}^* c^1[2,\{*\}].\overline{c^1}d^* \mid \overline{b}^* c^*[1,\{*\}]]\!] \\
&= [\![b^*(x)[1,\{*\}].\overline{a}^* c^1[2,\{*\}].\overline{c^1}d^* \mid \overline{b}^* c^*[1,\{*\}], \langle\rangle, S]\!] \\
&= [\![b^*(x)[1,\{*\}].\overline{a}^* c^1[2,\{*\}].\overline{c^1}d^*, \langle\uparrow\rangle.\langle\rangle, S]\!] \mid [\![\overline{b}^* c^*[1,\{*\}], \langle\uparrow\rangle.\langle\rangle, S]\!] \\
&= [\![\overline{a}^* c^1[2,\{*\}].\overline{c^1}d^*, \langle 1, *, \sigma(S, b^*[*/x])\rangle.\langle\uparrow\rangle.\langle\rangle, S]\!] \mid [\![\mathbf{0}, \langle 1, *, \sigma(S, \overline{b}^* c^*)\rangle.\langle\uparrow\rangle.\langle\rangle, S]\!] \\
&= [\![\overline{a}^* c^1[2,\{*\}].\overline{c^1}d^*, \langle 1, *, b[c/x]\rangle.\langle\uparrow\rangle.\langle\rangle, S]\!] \mid [\![\mathbf{0}, \langle 1, *, \overline{b}c\rangle.\langle\uparrow\rangle.\langle\rangle, S]\!] \\
&= [\![\overline{c^1}d^*, \langle 2, *, \sigma(S, \overline{a}^* c^1)\rangle.\langle 1, *, b[c/x]\rangle.\langle\uparrow\rangle.\langle\rangle, S]\!] \mid \langle 1, *, \overline{b}c\rangle.\langle\uparrow\rangle.\langle\rangle \triangleright \mathbf{0} \\
&= [\![\overline{c^1}d^*, \langle 2, *, \overline{a}x\rangle.\langle 1, *, b[c/x]\rangle.\langle\uparrow\rangle.\langle\rangle, S]\!] \mid \langle 1, *, \overline{b}c\rangle.\langle\uparrow\rangle.\langle\rangle \triangleright \mathbf{0} \\
&= \langle 2, *, \overline{a}x\rangle.\langle 1, *, b[c/x]\rangle.\langle\uparrow\rangle.\langle\rangle \triangleright \sigma(S, \overline{c^1}d^*) \mid \langle 1, *, \overline{b}c\rangle.\langle\uparrow\rangle.\langle\rangle \triangleright \mathbf{0} \\
&= \langle 2, *, \overline{a}x\rangle.\langle 1, *, b[c/x]\rangle.\langle\uparrow\rangle.\langle\rangle \triangleright \overline{x}d \mid \langle 1, *, \overline{b}c\rangle.\langle\uparrow\rangle.\langle\rangle \triangleright \mathbf{0}
\end{aligned}
$$

We now show a decomposition property for $R\pi$ transitions, which allows us to isolate the impact of structural congruence inside transitions.

Definition 31. The relation $\twoheadrightarrow$ is the smallest relation induced by the rules in Fig. B.9 (Appendix B), except the rules for congruence (rules (MEM+), (SPLIT) and (RES)).

Note that by definition in $R\pi$, we have $\twoheadrightarrow \subset \rightarrow$.

Lemma 6. *If there is an $R\pi$ transition $R \xrightarrow{(i,j,k):\alpha} S$ then there exist $R' \equiv R$ and $S' \equiv S$ such that $R' \xrightarrow{(i,j,k):\alpha}\!\!\!\!\!\!\twoheadrightarrow\ S'$.*

Proof. The proof is by induction on the derivation of the transition $R \xrightarrow{(i,j,k):\alpha} S$, with a case analysis on the last applied rule. The full proof can be found in Appendix B. □

We can now prove the property that every transition of a reachable process $S :: X$ in the framework can be mimicked with its translation $(R = [\![X, \langle\rangle, S]\!])$ in $R\pi$.

Proposition 1 *(Forward correctness). Let $S :: X$ be a reachable process from the framework and $R = [\![X, \langle\rangle, S]\!]$. For each transition in the framework $S :: X \xrightarrow{(i,K,j):\alpha} S' :: X'$ there exists a corresponding $R\pi$ transition $R \xrightarrow{(i,j,k):\alpha} R'$ with $[\![X', \langle\rangle, S']\!] = R'$ and $K = \{k\}$.*

Proof. The proof is by induction on the derivation of $S :: X \xrightarrow{(i,K,j):\alpha} S' :: X'$ and by case analysis on the last applied rule. The full proof can be found in Appendix B. □

Proposition 2 *(Backward correctness). Let $S :: X$ be a reachable process from the framework and $R = [\![X, \langle\rangle, S]\!]$. For each transition in the framework $S :: X \overset{(i,K,j):\alpha}{\rightsquigarrow} S' :: X'$ there exists a corresponding $R\pi$ transition $R \overset{(i,j,k):\alpha}{\rightsquigarrow} R'$ with $[\![X', \langle\rangle, S']\!] = R'$ and $K = \{k\}$.*

Proof. From the Loop Lemma in the framework (Lemma 3) we have that $S :: X \overset{(i,K,j):\alpha}{\rightsquigarrow} S :: X'$ implies $S' :: X' \xrightarrow{(i,K,j):\alpha} S :: X$. By Proposition 1 we have that there exists a corresponding $R\pi$ transition $R' \xrightarrow{(i,j,k):\alpha} R$ with $[\![X, \langle\rangle, S]\!] = R$ and $[\![X', \langle\rangle, S']\!] = R'$ and $K = \{k\}$. By applying the $R\pi$ Loop Lemma [23, Proposition 3.1] we have that $R \overset{(i,j,k):\alpha}{\rightsquigarrow} R'$, as desired. □

The two propositions above prove that if we have a couple of processes $(S :: X, R) = (S :: X, [\![X, \langle\rangle, S]\!])$ where $S :: X$ is a reachable process in the framework, and if $S :: X$ does an action α, then process R does the same action in $R\pi$. The resulting process $R' = [\![X', \langle\rangle, S']\!]$ is the encoding of process $S' :: X'$. Now we show the opposite direction.

Proposition 3 *(Forward completeness). Let X be a reachable process from the framework and $R = [\![X, \langle\rangle, S]\!]$. For each $R\pi$ transition $R \xrightarrow{(i,j,k):\alpha} R'$ there exists a corresponding transition in the framework $S :: X \xrightarrow{(i,K,j):\alpha} S' :: X'$ with $R' \equiv [\![X', \langle\rangle, S']\!]$ and $K = \{k\}$.*

Proof. Thanks to Lemma 6 we can equivalently write the statement as follows: for each reachable process $S :: X$ in the framework and $R\pi$ processes R and R'', such that $R = [\![X, \langle\rangle, S]\!]$ and $R'' \equiv R$, if there exists a $R\pi$ transition $R'' \xrightarrow{(i,j,k):\alpha} R'$, then there exists a corresponding transition in the framework $S :: X \xrightarrow{(i,K,j):\alpha} S' :: X'$, with $R' \equiv [\![X', \langle\rangle, S']\!]$. When considering $R'' \equiv R$ we will not consider α-conversion, since it can be trivially matched by framework α-conversion.

Now the proof is by structural induction on $S :: X$ with a case analysis on the last applied rule in the derivation of $R'' \xrightarrow{(i,j,k):\alpha} R'$. We have two main cases, depending on whether X is a process without past **P** or not. The full proof can be found in Appendix B. □

Proposition 4 *(Backward completeness). Let X be a reachable process from the framework and $R = [\![X, \langle\rangle, S]\!]$. For each $R\pi$ transition $R \overset{(i,j,k):\alpha}{\rightsquigarrow} R'$ there exists a corresponding transition in the framework $S :: X \overset{(i,K,j):\alpha}{\rightsquigarrow} S' :: X'$ with $R' \equiv [\![X', \langle\rangle, S']\!]$ and $K = \{k\}$.*

Proof. From the $R\pi$ Loop Lemma we have that $R' \xrightarrow{(i,j,k):\alpha} R$ in $R\pi$. From Proposition 3 we have that $S' :: X' \xrightarrow{(i,K,j):\alpha} S :: X$ with $[\![X, \langle\rangle, S]\!] \equiv R$ and $[\![X', \langle\rangle, S']\!] = R'$ and $K = k$. By applying $R\pi$ rule (MEM+) to $R' \xrightarrow{(i,j,k):\alpha} R$ and $R \equiv [\![X, \langle\rangle, S]\!]$ we have that $R' \xrightarrow{(i,j,k):\alpha} [\![X, \langle\rangle, S]\!]$. By applying the framework Loop Lemma we have that $S :: X \overset{(i,K,j):\alpha}{\rightsquigarrow} S' :: X'$. □

Now we have all necessary auxiliary lemmas to prove the operational correspondence between the framework when $\Delta = \Gamma$ and the semantics of [23]. We prove it in terms of strong back and forth bisimulation [16].

Theorem 2 *(Operational correspondence). Given a reachable process from the framework $S :: X$, the relation $\mathcal{R} = \{(S :: X, [\![X, \langle\rangle, S]\!])\}$ is a strong back and forth bisimulation.*

Proof. If $S :: X$ does a forward transition $S :: X \xrightarrow{(i,K,j):\alpha} S' :: X'$, then by Proposition 1, we have that $[\![X, \langle\rangle, S]\!] \xrightarrow{(i,j,k):\alpha} [\![X', \langle\rangle, S']\!]$ with $K = \{k\}$ and $(S' :: X', [\![X', \langle\rangle, S']\!]) \in \mathcal{R}$. If the transition is a backward one we apply Proposition 2.

If $R = [\![X, \langle\rangle, S]\!]$ does a forward transition $R \xrightarrow{(i,j,k):\alpha} R'$, then by Proposition 3 we have $S :: X \xrightarrow{(i,K,j):\alpha} S' :: X'$ with $K = k$, $R' \equiv [\![X', \langle\rangle, S']\!]$ and $(S' :: X', [\![X', \langle\rangle, S']\!]) \in \mathcal{R}$. If the transition is a backward one we apply Proposition 4. □

Let us note that the result presented in this section (Theorem 2) is on the (strong) operational correspondence between two reversible calculi and it is not affected by the fact that we depart from the conflict notion of [23].

$$\text{(BS-Out)}\ \overline{b}a.A \xrightarrow[\emptyset]{k:\overline{b}a} k :: A \qquad \text{(BS-In)}\ b(x).A \xrightarrow[\emptyset]{k:b(x)} k :: A$$

$$\text{(BS-Cau)}\ \frac{A \xrightarrow[\mathrm{K}]{k:\alpha} A'}{\mathrm{K}' :: A \xrightarrow[\mathrm{K},\mathrm{K}']{k:\alpha} \mathrm{K}' :: A'} \qquad \text{(BS-Par)}\ \frac{A_1 \xrightarrow[\mathrm{K}]{k:\alpha} A_1' \quad \mathtt{Bnv}(\alpha) \cap \mathtt{Fnv}(A_2) = \emptyset}{A_1 \mid A_2 \xrightarrow[\mathrm{K}]{k:\alpha} A_1' \mid A_2}$$

$$\text{(BS-Com)}\ \frac{A_1 \xrightarrow[\mathrm{K}_1]{k:\overline{b}a} A_1' \quad A_2 \xrightarrow[\mathrm{K}_2]{k:b(x)} A_2' \quad k \notin \mathcal{K}(A_1, A_2)}{A_1 \mid A_2 \xrightarrow{\tau} A_1'[k \rightsquigarrow \mathrm{K}_2] \mid A_2'\{^a/_x\}[k \rightsquigarrow \mathrm{K}_1]}$$

$$\text{(BS-Open)}\ \frac{A \xrightarrow[\mathrm{K}]{k:\overline{b}a} A'}{\nu a\, A \xrightarrow[\mathrm{K}]{k:\overline{b}\langle \nu a\rangle} A'} \qquad \text{(BS-Res)}\ \frac{A \xrightarrow[\mathrm{K}]{k:\alpha} A' \quad a \notin \mathrm{n}(\alpha)}{\nu a\, A \xrightarrow[\mathrm{K}]{k:\alpha} \nu a\, A'}$$

$$\text{(BS-Close)}\ \frac{A_1 \xrightarrow[\mathrm{K}_1]{k:\overline{b}\langle \nu a\rangle} A_1' \quad A_2 \xrightarrow[\mathrm{K}_2]{k:b(x)} A_2' \quad k \notin \mathcal{K}(A_1, A_2)}{A_1 \mid A_2 \xrightarrow{\tau} \nu a(A_1'[k \rightsquigarrow \mathrm{K}_2] \mid A_2'\{^a/_x\}[k \rightsquigarrow \mathrm{K}_1])}$$

Fig. 7. Causal semantics rules.

6.2. 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 [20]. Later on, Degano and Priami in [19] introduced a causal semantics for π based on localities. While using different approaches to keep track of the dependencies in the π-calculus, these two approaches impose the same order of the forward actions (as claimed in [19]). Hence, from the reversible point of view 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 below.

6.2.1. Boreale and Sangiorgi's late semantics

Originally the causal semantics of [20] was defined for a polyadic π-calculus, with early semantics for inputs. Here we adapt the causal semantics to work with monadic π-calculus and late input semantics.

The authors of [20] distinguish between two forms of dependencies: subject and object. While the object dependence can be detected from the trace (called run in [20]) that a process performed, to track the subject one, the authors introduced a causal term, defined on top of the π-calculus. Every visible transition is bound with a unique cause $k \in \mathcal{K}$, where $\mathcal{K}$ is a set of causes. Let $\mathcal{N}$ and $\mathcal{V}$ be infinite, countable sets of names and variables such that $\mathcal{N} \cap \mathcal{V} \cap \mathcal{K} = \emptyset$.

The syntax of the causal process is defined as follows:

$$\text{(Causal process)}\ A, B ::= P \mid \mathrm{K} :: A \mid A \mid B \mid \nu a(A)$$

where P is a π-calculus process defined in Fig. 1. In causal term $\mathrm{K} :: A$, set K records that every action performed by A depends on K. Two causal processes can be composed in parallel by $A \mid B$ and name a can be restricted in the process A. The set of causes appearing in the causal process A is denoted with $\mathcal{K}(A)$ and its definition is given below.

Definition 32 *(Set of the causes).* The set of causes of a given causal process A, written as $\mathcal{K}(A)$, is inductively defined on the structure of the causal term as:

$$\mathcal{K}(A \mid B) = \mathcal{K}(A) \cup \mathcal{K}(B) \qquad \mathcal{K}(\mathrm{K} :: A) = \mathrm{K} \cup \mathcal{K}(A)$$
$$\mathcal{K}(\nu a(A)) = \mathcal{K}(A) \qquad \mathcal{K}(P) = \emptyset$$

In what follows, we give the definition of the labels on the transitions and operational semantics of the causal term.

Definition 33 *(Label on the transition).* The label on the transition of the causal process is defined as $A \xrightarrow[\mathrm{K}]{k:\alpha} A'$, where cause set K contains causes of all the actions that trigger the action α; k is the cause associated to α and $\alpha ::= \overline{b}a \mid b(x) \mid \overline{b}\langle \nu a\rangle \mid \tau$.

The rules for the causal semantics are given in Fig. 7. Causes are introduced into the processes by the rules (BS-Out) and (BS-In). A new cause k is attached to the executing action. Rule (BS-Cau) allows a causal process $\mathrm{K}' :: A$ to move if A can move, while cause set K' is preserved. Rules (BS-Open) and (BS-Res) are defined in the usual way. The communication between two causal processes can be done through rules (BS-Com) and (BS-Close), while necessary substitution is applied.

Notation $A_1'[k \rightsquigarrow \mathtt{K}_2]$ indicates the fact that cause k needs to be replaced with the set $\mathtt{K}_2$. The condition $k \notin \mathcal{K}(A_1, A_2)$ ensures that cause k is fresh. The synchronisation rules merge the cause sets of processes that communicate and do not produce a new cause bound to τ. The reason for this is that τ actions do not impose causes on future actions.

In order to provide a better intuition of how the semantics works, we give the following example.

Example 11. Let us consider the π-calculus process $P = \overline{a}b.\overline{c}d.\mathbf{0} \mid \overline{e}a_1.c(x).Q$. Process P can perform the output on channel a, and we have:

$$\overline{a}b.\overline{c}d.\mathbf{0} \mid \overline{e}a_1.c(x).Q \xrightarrow[\emptyset]{k:\overline{a}b} \{k\} :: \overline{c}d.\mathbf{0} \mid \overline{e}a_1.c(x).Q = A$$

We can notice that cause k is bound to the action $\overline{a}b$, and saved in the resulting process. Since P was a π-calculus process, the cause set $\mathtt{K}$ is empty. To continue execution, process A can perform the output on channel e:

$$k :: \overline{c}d.\mathbf{0} \mid \overline{e}a_1.c(x).Q \xrightarrow[\emptyset]{k_1:\overline{e}a_1} \{k\} :: \overline{c}d.\mathbf{0} \mid \{k_1\} :: c(x).Q$$

The two executed actions are structurally independent and this is the reason why the cause set is empty in both cases. Now we can synchronise two processes in parallel:

$$k :: \overline{c}d.\mathbf{0} \mid k_1 :: c(x).Q \xrightarrow{\tau} \{k, k_1\} :: \mathbf{0} \mid \{k_1, k\} :: Q\{^d/_x\}$$

As we can notice, after the communication, cause sets $\{k\}$ and $\{k_1\}$ are merged, meaning that the actions of Q will structurally depend on both actions $\overline{a}b$ and $\overline{e}a_1$.

Remark 6. In the original version of the causal semantics [20], labels on transitions are defined as $A \xrightarrow[\mathtt{K};\, k]{\alpha} A'$. We use the notation $A \xrightarrow[\mathtt{K}]{k:\alpha} A'$ to simplify the comparison given in Section 6.2.3. For the same reason we divided the original rule for communication (Com) from [20] into two rules: (BS-Com) and (BS-Close).

The subject causality is given by the cause sets attached to the process, while the object one is defined on the trace that the process performed. The first action that extrudes a bound name causes every further action using that name in the subject or in the object position of the label. To illustrate this, we give the following example.

Example 12. Let us consider a process $P = \nu a(\overline{b}a \mid \overline{c}a \mid a(x))$ and the trace:

$$\nu a(\overline{b}a \mid \overline{c}a \mid a(x)) \xrightarrow[\emptyset]{k:\overline{b}\langle \nu a\rangle} A_1 \xrightarrow[\emptyset]{k_1:\overline{c}a} A_2 \xrightarrow[\emptyset]{k_2:a(x)} \{k\} :: \mathbf{0} \mid \{k_1\} :: \mathbf{0} \mid \{k_2\} :: \mathbf{0}$$

where $A_1 = \{k\} :: \mathbf{0} \mid \overline{c}a \mid a(x)$ and $A_2 = \{k\} :: \mathbf{0} \mid \{k_1\} :: \mathbf{0} \mid a(x)$. The action $\overline{b}\langle \nu a\rangle$ extrudes name a and causes other two actions: $\overline{c}a$ where name a is in the object position and $a(x)$ where a is in the subject position.

We adapt the definition of object causality to the late semantics defined on the traces of the causal process A.

Definition 34 *(Object causality).* In a trace $A_1 \xrightarrow[K_1]{k_1:\alpha_1} A_2 \cdots A_n \xrightarrow[K_n]{k_n:\alpha_n} A_{n+1}$ where A_1 is a π-calculus process P, if

- $\alpha_i = \overline{b}\langle \nu a\rangle$ where $a \cap \mathtt{fn}(A_i) = \emptyset$ and for all $j < i$, $a \cap \mathtt{n}(\alpha_j) = \emptyset$ we say that name a is *introduced* in α_i. Action α_h is *object dependent* on α_i, $1 \leqslant i < h \leqslant n$, if there is a name introduced in α_i which is among the free names of α_h.
- $\alpha_i = b(x)$ where $x \cap \mathtt{fn}(A_i) = \emptyset$ and for all $j < i$, $x \cap \mathtt{v}(\alpha_j) = \emptyset$ we say that variable x is *introduced* in α_i. Action α_h is *object dependent* on α_i, $1 \leqslant i < h \leqslant n$, if there is a variable introduced in α_i which is among the free variables of α_h.

6.2.2. Boreale and Sangiorgi's late semantics captured in the framework

In the following we show how the behaviour of Boreale and Sangiorgi's late semantics is captured by the framework.

Definition 35 *(Boreale and Sangiorgi causal semantics).* To capture the causality induced by Boreale and Sangiorgi's late semantics, the data structure Δ is instantiated with the indexed set Γ_w.

From the object causality definition (Definition 34), we have that an output action can be caused through the subject and the object position of a label. For instance, consider a process $\nu a(\nu b(\overline{c}b \mid \overline{d}a \mid \overline{b}a))$ and its 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 both actions executed before (on the first one because it extrudes the name b and on the second one because it extrudes the name a).

In [27], we present our initial idea of the reversible framework where we discussed the necessity of adding causal information to the silent actions in the semantics of [20]. Here we take different approach and keep the silent actions as they are in the original semantics of [20]. Therefore, silent actions do not exhibit or impose contextual causes.

We give the following example to provide a better intuition about the framework expressing Boreale and Sangiorgi causality.

Example 13. Consider the process $X = \nu a_{\emptyset_*}(\overline{b}^* a^* \mid \overline{c}^* a^* \mid a^*(x))$. By extruding the name a over the channel b (rule (Open) is applied), we obtain the process:

$$\nu a_{\{i_1\}_{i_1}}(\overline{b}^* a^*[i_1, \{*\}] \mid \overline{c}^* a^* \mid a^*(x))$$

From the memory $\{i_1\}_{i_1}$ we have that $w = i_1$. By executing the actions $\overline{c}a$ with the rule (Open) and $a(x)$ with the rule (Cause Ref), predicates $\texttt{Update}(\cdot)$ and $\texttt{Cause}(\cdot)$ ensure that the key $i_1 = w$ will be added to the cause sets. We obtain the process:

$$\nu a_{\{i_1,i_2\}_{i_1}}(\overline{b}^* a^*[i_1, \{*\}] \mid \overline{c}^* a^*[i_2, \{i_1, *\}] \mid a^*(x)[i_3, \{i_1, *\}])$$

In the memories $[i_2, \{i_1, *\}]$ and $[i_3, \{i_1, *\}]$ we can notice that the executed actions are caused by the action i_1 and for this reason the action i_1 needs to be reversed last. The actions i_2 and i_3 can be reversed in any order.

6.2.3. Correspondence with Boreale and Sangiorgi's semantics

In this section, we prove the 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 full proof is given in Appendix C.

Now we observe the differences in the structural causality in both settings by looking at process traces. Since the framework is meant for reversible computation (Lemma 3), every action has its corresponding key, including τ-actions, which is not the case in [20], where synchronisation only merges cause sets of the actions that communicate. Additionally, for the same reason, the framework keeps track of every action that was executed, while the semantics in [20] just records the sets of causes that trigger the performing action. Example 14 will exemplify how our framework carries more information than [20] in a simple execution, and Example 15 will motivate and ease the definition of structural causality that follows.

Example 14. Let us consider the π-calculus process $P = \overline{b}a.\overline{c}d \mid \overline{e}a_1.c(x).Q$. By executing two output actions on the channel b and e and synchronisation on the channel c, we obtain:

- In [20], the resulting process is $A = \{i_1, i_2\} :: \mathbf{0} \mid \{i_1, i_2\} :: Q$ where keys i_1 and i_2 correspond to actions $\overline{b}a$ and $\overline{e}a_1$, respectively. We can notice that the τ-action just merged the cause sets $\{i_1\}$ and $\{i_2\}$. For more details about the computation, see Example 11.
- In our framework, the resulting process is

  $$X = \overline{b}a[i_1, \{*\}].\overline{c}d[i_3, \{*\}] \mid \overline{e}a_1[i_2, \{*\}].c(x)[i_3, \{*\}].Q$$

  As we can notice, the τ-action is identified by the key i_3.

Another difference is that in [20], the executing action brings its cause set into the label of the transition, while in the framework, the structural cause set is defined on the resulting process of the transition.

Example 15. Consider the π-calculus process $P = \overline{b}a.\overline{c}d.\overline{e}f$:

- In [20], actions $\overline{b}a$ with i_1 and $\overline{c}d$ with i_2 can be performed and we obtain the causal process $A = \{i_1, i_2\} :: \overline{e}f$. The transition for the action $\overline{e}f$ is

  $$\{i_1, i_2\} :: \overline{c}d \xrightarrow[\{i_1,i_2\}]{i_3:\overline{e}f} \{i_1, i_2, i_3\} :: \mathbf{0}$$

  We can notice that in the label of the transition we can see the whole set of the causes that cause action $\overline{e}f$.
- In the framework, the same actions are performed and obtained reversible process is $X = \overline{b}a[i_1, \{*\}].\overline{c}d[i_2, \{*\}].\overline{e}f$. The transition for the action $\overline{e}f$ is

  $$\overline{b}a[i_1, \{*\}].\overline{c}d[i_2, \{*\}].\overline{e}f \xrightarrow{(i_3,\{*\},*):\overline{e}f} \overline{b}a[i_1, \{*\}].\overline{c}d[i_2, \{*\}].\overline{e}f[i_3, \{*\}] = X'$$

  The structural causality is defined on the prefixes of the resulting process X' (Definition 21); hence the set of the structural causes of the action with the key i_3 can be computed after the execution.

In the following, we define the structural cause set of an executed action in the framework.

Definition 36 *(Structural causes set K_F).* Given a transition $X \xrightarrow{(i,K,j):\alpha} X_1$, the set of keys of the actions that structurally caused the action $(i, K, j) : \alpha$, denoted with K_F, is defined as: for all $i' \in \mathtt{key}(X_1)$, if $i' \sqsubseteq_{X_1} i$, then $i' \in K_F$.

Example 16. To understand it better, let us consider a reversible process

$$X = \overline{b}^* a^*[i_1, \{*\}].\overline{c}^* d^*[i_3, \{*\}].\overline{e}^* f^*[i_4, \{*\}] \mid \overline{b_1}^* a_1^*[i_2, \{*\}].c^*(x)[i_3, \{*\}].\overline{b}^* c^*$$

and transition $X \xrightarrow{(i,\{*\},*):\overline{b}c} X_1$, where

$$X_1 = \overline{b}^* a^*[i_1, \{*\}].\overline{c}^* d^*[i_3, \{*\}].\overline{e}^* f^*[i_4, \{*\}] \mid \overline{b_1}^* a_1^*[i_2, \{*\}].c^*(x)[i_3, \{*\}].\overline{b}^* c^*[i, \{*\}]$$

The cause set K_F of the action $\overline{b}c$, identified with the key i, is $K_F = \{i_1, i_2, i_3\}$. The action identified with i_4 is not the structural cause of any action in X_1 because there do not exist contexts C_4 and C_4' such that $X_1 = C_4[\overline{e}^* f^*[i_4, \{*\}].Y]$, where $Y = C_4'[\pi_h[h, \{*\}]]$ and $h \in \{i, i_1, i_2, i_3\}$ (Definition 22).

Notation 1. To distinguish the labels of the two semantics, we shall write:

- a transition from [20] as $A \xrightarrow[\mathtt{K}]{\zeta} A_1$, where

  $$\zeta = i : \beta \text{ and } \beta = \overline{b}a \mid b(x) \mid \overline{b}\langle \nu a \rangle \mid \tau$$

  and $i \in \mathcal{K}$ ($\mathcal{K}$ is an infinite denumerable set of keys);
- a transition from the framework as $X \xrightarrow{\mu} X_1^{K_F}$, where

  $$\mu = (i, K, j) : \alpha \text{ and } \alpha = \overline{b}a \mid b(x) \mid \overline{b}\langle \nu a_\Delta \rangle \mid \tau$$

  with $i \in \mathcal{K}$, $j \in \mathcal{K}_*$, $K \subset \mathcal{K}_*$ and K_F the set of the keys belonging to the actions that structurally cause the action μ.

Focusing on structural causality, the main difference between the two semantics is in the τ-actions; therefore, we need to provide the connection between the structural cause sets $\mathtt{K}$ and K_F of these two semantics. The idea is to represent structural dependences between keys in the reversible process X involved in the executing action (i.e. past prefixes whose keys belong to the set K_F) as a directed graph (digraph) [34] and by removing the nodes (keys) belonging to the τ-actions, obtain the cause set $\mathtt{K}$ of the corresponding causal process A. In order to illustrate the method which connects structural cause sets $\mathtt{K}$ and K_F, we give the following example.

Example 17. Let us consider the π-calculus process

$$P = \overline{b}a.\overline{c}d \mid \overline{b_1}a_1.c(x).\overline{b}c.\overline{b_2}a_2 \mid \overline{f}e.b(y)$$

where the actions $\overline{b}a, \overline{b_1}a_1, \tau, \overline{f}e, \tau$ and $\overline{b_2}a_2$ are identified with the keys i_1, i_2, i_3, i_4, i_5 and i_6, respectively.

- in the semantics of [20], we have:

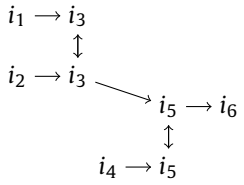
$$\begin{aligned}
P &\xrightarrow[\emptyset]{i_1:\overline{b}a} \{i_1\} :: \overline{c}d \mid \overline{b_1}a_1.c(x).\overline{b}c.\overline{b_2}a_2 \mid \overline{f}e.b(y) \\
&\xrightarrow[\emptyset]{i_2:\overline{b_1}a_1} \{i_1\} :: \overline{c}d \mid \{i_2\} :: c(x).\overline{b}c.\overline{b_2}a_2 \mid \overline{f}e.b(y) \\
&\xrightarrow{\tau} \{i_1, i_2\} :: \mathbf{0} \mid \{i_2, i_1\} :: \overline{b}c.\overline{b_2}a_2 \mid \overline{f}e.b(y) \\
&\xrightarrow[\emptyset]{i_4:\overline{f}e} \{i_1, i_2\} :: \mathbf{0} \mid \{i_2, i_1\} :: \overline{b}c.\overline{b_2}a_2 \mid \{i_4\} :: b(y) \\
&\xrightarrow{\tau} \{i_1, i_2\} :: \mathbf{0} \mid \{i_2, i_1, i_4\} :: \overline{b_2}a_2 \mid \{i_4, i_2, i_1\} :: \mathbf{0} \\
&\xrightarrow[\{i_4,i_2,i_1\}]{i_6:\overline{b_2}a_2} \{i_1, i_2\} :: \mathbf{0} \mid \{i_2, i_1, i_4\} :: \{i_6\} :: \mathbf{0} \mid \{i_4, i_2, i_1\} :: \mathbf{0}
\end{aligned}$$

  As we can notice, τ-actions do not have keys; hence i_3, i_5 are not in A. We separated set $\{i_6\}$ from the rest of the cause set to emphasise the fact that the action with key i_6 depends on the cause set $\mathtt{K} = \{i_1, i_2, i_4\}$.

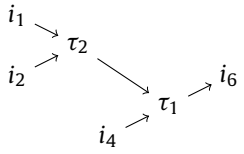
- the resulting process in the framework is:

$$X = \overline{b}^*a^*[i_1,\{*\}].\overline{c}^*d^*[i_3,\{*\}] \mid \overline{b_1}^*a_1^*[i_2,\{*\}].c^*(x)[i_3,\{*\}].\overline{b}^*c^*[i_5,\{*\}].\overline{b_2}^*a_2^*[i_6,\{*\}]$$
$$\mid \overline{f}^*e^*[i_4,\{*\}].b^*(y)[i_5,\{*\}]$$

We can represent dependences between the keys as a digraph $G = (V, E)$, given below:



In the digraph G, nodes represent keys and directed edge $i \to i'$ symbolises that key i causes key i'. From the graph, we can notice that the cause set of the action with key i_6 is $K_F = \{i_1, i_2, i_3, i_4, i_5\}$. If we remove all bidirectional edges, join the nodes that they connect (keys belong to synchronisations) and rename them into τ_l, we obtain the graph $G' = (V', E')$:



Now, if we take the set of vertices V' and remove all τ_i nodes, we obtain the cause set $\mathtt{K}$ of the action i_6 in Boreale and Sangiorgi's semantics ($\mathtt{K} = V' \setminus \{\tau_l\}$).

The whole algorithm of connecting sets K_F and $\mathtt{K}$, illustrated in Example 17 is called *Removing Keys from a Set* written as $\mathtt{Rem}(K_F) = \mathtt{K}$.

Before defining our method, we recall some basic notions about graphs [34]. A *directed graph* or *digraph* $G = (V, E)$ consists of the non-empty set of vertices V (nodes) and the set of directed edges $E = \{(v_1, v_2)|$ where $v_1, v_2 \in V\}$. In the edge (v_1, v_2), v_1 is the *source* vertex of the edge, while v_2 is the *target* vertex.

Now we give the auxiliary definitions necessary to formally define the method $\mathtt{Rem}$. First, we define a function $\gamma(\cdot)$ that translates the label from the framework μ into a label from Boreale and Sangiorgi's semantics ζ. It is done by discarding the instantiator j and cause set K of the label μ, since they are not present in the semantics of [20], and translating the action label α into the corresponding action label of [20]. Additionally, when τ-actions are considered, in semantics [20], they are not bound with a new key; hence the key i is deleted as well. Formally:

Definition 37. The function γ that maps label from the framework μ, with a label from [20] ζ, is inductively defined as follows:

$$\gamma((i,K,j):\alpha) = i:\gamma(\alpha) \quad \text{when } \alpha \neq \tau \qquad \gamma((i,\{*\},*):\tau) = \tau$$
$$\gamma(\overline{b}\langle \nu a_\Delta\rangle) = \overline{b}\langle \nu a\rangle \quad \text{when } \mathtt{empty}(\Delta) = \mathtt{true} \qquad \gamma(b(x)) = b(x)$$
$$\gamma(\overline{b}\langle \nu a_\Delta\rangle) = \overline{b}a \quad \text{when } \mathtt{empty}(\Delta) = \mathtt{false} \qquad \gamma(\overline{b}a) = \overline{b}a$$

Suppose that we have two transitions:

$$t : X \xrightarrow{\mu} \!\!\!\!\rightarrow X_1^{K_F} \quad \text{and} \quad t' : A \xrightarrow[\mathtt{K}]{\zeta} A_1$$

with $\gamma(\mu) = \zeta$ and processes X and A, translated in the π-calculus, give the same process P, i.e. $\varphi(X) = \lambda(A) = P$. The same holds for the processes $X_1^{K_F}$ and A_1, i.e., we have $\varphi(X_1^{K_F}) = \lambda(A_1) = Q$. The function $\varphi(\cdot)$ is the same as the erasing function from Definition 19 with an additional rule $\varphi(X^{K_F}) = \varphi(X)$ that removes the set K_F from the reversible process. The function λ that translates a causal term into a π-calculus process is defined as:

Definition 38. The erasing function λ that maps causal processes from Boreale and Sangiorgi's semantics into the π-calculus is inductively defined as follows:

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

The structural dependences between the past prefixes belonging to the history of the process X involved in the execution of the action $\alpha \in t, t'$ can be represented with a digraph in the following way: keys belonging to the past prefixes are represented as vertices of the digraph (the same keys which are representing synchronisation, are represented by two vertices with the same name); structural dependences between the keys are represented by directed edges where between the same vertices we shall have edges in both directions. Formally, we have:

Definition 39. Given a transition $t : X \xrightarrow{(i,K,j):\alpha} X_1^{K_F}$ the structural dependences between the past prefixes involved in the execution of the action α contained in the history of the reversible process $X_1^{K_F}$ can be represented as a digraph $G = (V, E)$, in the following way:

- $\forall\, \pi[i_1, K] \in X_1^{K_F} \wedge i_1 \sqsubseteq_{X_1^{K_F}} i \implies i \in V$
- $\forall\, i_1, i_2 \in V$ such that $\pi[i_1, K].\pi'[i_2, K'] \in X_1^{K_F} \implies (i_1, i_2) \in E'$
- $E = E' \cup \{(i_1, i_2) \mid \text{when } i_1, i_2 \in V \wedge i_1 = i_2\}$
  $\cup\, \{(i_2, i_1) \mid \text{when } i_1, i_2 \in V \wedge i_2 = i_1\}$

where V is a multiset of vertices and E is a set of directed edges. Having a digraph $G = (V, E)$, the structural cause set of the action α is $K_F = V \setminus \{i\}$.

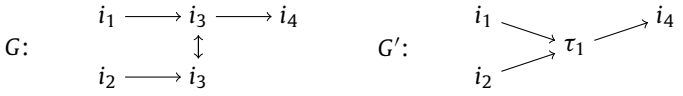
Since bidirectional edges represent dependency flow between vertices with the same name, we can remove them and join two vertices into one, renamed to τ. This operation is known as edge contraction [34]. Here we adapt it to bidirectional edges as follows:

Definition 40 *(Bidirectional edge contraction).* Bidirectional edge contraction is an operation defined on the directed graph $G = (V, E)$, as follows:

- $E' = E \setminus ((i_1, i_2) \cup (i_2, i_1))$ when $i_1 = i_2$
- $V' = (V \setminus \{i_1, i_2\}) \cup \{\tau_l\}$ when $i_1 = i_2$
- $\forall (i, i_h), (i_h, i) \in E$ where $h \in \{1, 2\}$, we have that $(i, \tau_l), (\tau_l, i) \in E'$,

where $G' = (V', E')$ is the obtained graph.

In words, the above definition removes a bidirectional edge and substitutes two nodes that it connects with the τ_l node. Additionally, all the edges that have a source or target in the removed nodes will have a source or target in the τ_l node. For instance, let $G = (V, E)$ be directed graph and G' be the graph obtained from the graph G by applying Definition 40:



From the representations of the graphs G and G' above, we can notice that nodes labelled with i_3 together with the bidirectional edge, are substituted with the node τ_1. At the same time, edge (i_1, i_3) becomes (i_1, τ_1), and similarly for the rest of the edges containing nodes labelled with i_3.

By applying bidirectional edge contraction (Definition 40) on every bidirectional edge of a graph $G = (V, E)$, we obtain a graph $G' = (V', E')$ in which all pairs of the same vertices are joined and renamed as τ_l, for $l = 1, 2, \ldots$ Set V' differs from the multiset V in having τ_l vertices instead of the pairs of vertices labelled with the same name (originally belonging to silent moves in the framework). Hence, we can conclude that $\mathtt{K} = V' \setminus (\{i\} \cup \tau_l)$.

The method 'Removing Keys from a Set', denoted as `Rem` is formally defined as.

Definition 41 *(Method* `Rem`*).* Given two transitions $t : X \xrightarrow{\mu} X_1^{K_F}$ with $\mu = (i, K, j) : \alpha$ and $t' : A \xrightarrow[\mathtt{K}]{\zeta} A_1$, where $\gamma(\mu) = \zeta$ and $\varphi(X) = \lambda(A) = P$ and $\varphi(X_1^{K_F}) = \lambda(A_1) = Q$, a correspondence between sets K_F and $\mathtt{K}$ is defined through method `Rem`, given with the following steps:

- structural dependences in the process $X_1^{K_F}$ involved in the transition t are represented as the digraph $G = (V, E)$ (Definition 39), where $K_F = V \setminus \{i\}$;

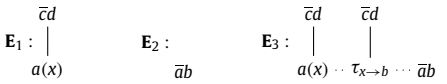


Fig. 8. The event structures representing processes P_1, P_2 and P_3.

- by applying Definition 40 on every bidirectional edge in $G = (V, E)$, the graph $G' = (V', E')$ is obtained, where $V' \setminus (\{i\} \cup \tau_l) = \mathtt{K}$ (τ_l represents all the nodes obtained by bidirectional edge contraction).

Now we can state a lemma expressing the structural correspondence between Boreale and Sangiorgi's late causal semantics and the framework when $\Delta = \Gamma_w$.

Lemma 7 *(Structural correspondence). Starting from initial π-calculus process P, where $P = \varphi(\mathbf{P})$, we have:*

1. *if $P \xrightarrow[\mathtt{K}_1]{\zeta_1} A_1 \dots A_{n-1} \xrightarrow[\mathtt{K}_n]{\zeta_n} A_n$ is a trace in causal semantics [20], then there exists a trace $\mathbf{P} \xrightarrow{\mu_1} X_1^{K_{F1}} \dots X_{n-1}^{K_{Fn-1}} \xrightarrow{\mu_n} X_n^{K_{Fn}}$ and K_{Fi} in the framework, such that for all i, $\lambda(A_i) = \varphi(X_i^{K_{Fi}})$, $\zeta_i = \gamma(\mu_i)$ and $\mathtt{Rem}(K_{Fi}) = \mathtt{K}_i$, for $i = 1, \dots, n$.*
2. *if $\mathbf{P} \xrightarrow{\mu_1} X_1^{K_{F1}} \dots X_{n-1}^{K_{Fn-1}} \xrightarrow{\mu_n} X_n^{K_{Fn}}$ is a trace in the framework, then there exists a trace $P \xrightarrow[\mathtt{K}_1]{\zeta_1} A_1 \dots A_{n-1} \xrightarrow[\mathtt{K}_n]{\zeta_n} A_n$ in the causal semantics, where for all i, $\lambda(A_i) = \varphi(X_i^{K_{Fi}})$, $\zeta_i = \gamma(\mu_i)$ and $\mathtt{Rem}(K_{Fi}) = \mathtt{K}_i$, for $i = 1, \dots, n$.*

Proof. Both clauses (*1.* and *2.*) are proved by induction on the length of the computation followed by induction on the structure of the process A_n ($X_n^{K_{Fn}}$ for the clause *2.*) and the last applied rule on the transition $t : A_n \xrightarrow[\mathtt{K}_{n+1}]{\zeta_{n+1}} A_{n+1}$ ($t' : X_n^{K_{Fn}} \xrightarrow{\mu_{n+1}} X_{n+1}^{K_{Fn+1}}$ for the clause *2.*). The full proof is given in Appendix C. □

The object causality in Boreale and Sangiorgi's semantics is defined on the trace of a process. Hence, to have a direct correspondence, we redefine the object causality induced by the framework, on the forward trace of a reversible process. Previously it was defined on two consecutive transitions (Definition 23). In this way, Definition 23 will represent the case when $n = 2$.

Definition 42 *(Object causality on the trace in the framework).* In the trace $t_1 : X_1 \xrightarrow{(i_1, K_1, j_1):\alpha_1} X_2 \cdots t_n : X_n \xrightarrow{(i_n, K_n, j_1):\alpha_n} X_{n+1}$, transition t_h is an *object cause* of transition t_l ($1 \le h < l \le n$), written $t_h < t_l$, if $i_h \in K_l$.

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

Theorem 3 *(Causal correspondence). The reflexive and transitive closure of the causality introduced in [20] coincides with the causality of the framework when $\Delta = \Gamma_w$.*

Proof. The proof relies on Lemma 7 and the fact that object dependence induced by input actions in Boreale and Sangiorgi's semantics is subject dependence as well. By the design of the framework and the definitions for predicates $\mathtt{Cause}(\cdot)$ and $\mathtt{Update}(\cdot)$, the first extrusion of a name will cause every other action using that name (this is accomplished with the rules (Open) and (Cause Ref)). In Definition 34 object dependence induced by an input action is also the structural one, and the one induced by extrusion coincides with object dependence in the framework. □

6.3. Crafa, Varacca and Yoshida causal semantics

A compositional event structure semantics for the forward π-calculus is introduced in [22]. A process is represented as a pair $(E, \mathtt{X})$, where E is a prime event structure and $\mathtt{X}$ is a set of bound names. For instance, process $P_3 = P_1 \mid P_2$ where $P_1 = a(x).\overline{c}d$ and $P_2 = \overline{a}b$ is given in event structure semantics in Fig. 8. The event structures $\mathbf{E_1}, \mathbf{E_2}$ and $\mathbf{E_3}$ represent processes P_1, P_2 and P_3, respectively. Causal order is represented with straight lines, while conflict is represented with dotted lines. In the event structure $\mathbf{E}_3$ we can notice that there are two possibilities for the computation: either action $a(x)$ will execute and trigger the action $\overline{c}d$, while in parallel the input action $\overline{a}b$ can be performed; or processes P_1 and P_2 will synchronise and then trigger the action $\overline{c}d$. The synchronisation pair $(a(x), \overline{a}b)$ is relabelled into $\tau_{x \to b}$.

Disjunctive object causality is represented in such a way that it is not necessary to remember the exact extruder of a bound name. It is important that at least one happened in the past. In the case of parallel extrusion of the same name, for instance in process $\nu a(\overline{b}a \mid \overline{c}a \mid a(x))$, action $a(x)$ can be caused by any of the extrusions (configurations $\{\overline{b}a, a(x)\}$ and $\{\overline{c}a, a(x)\}$ are both permitted), without recording the actual extruder.

Consequently, events do not have a unique causal history. As discussed in [24] this type of disjunctive causality cannot be expressed when we consider processes with contexts. For example, in the process $P = \nu a(\overline{b}a \mid \overline{c}a \mid a(x))$, the cause of the action $a(x)$ is either $\overline{b}a$ or $\overline{c}a$ but if context is added to this process (i.e. we consider closed terms), the ambiguity of the cause choice is lost. The choice of the cause is determined by the context (for instance, we can add context $b(y).\overline{y}d$ to the process P and in this case we know that the action $a(x)$ is caused by $\overline{b}a$).

We consider two possibilities for keeping track of the causes: the first one is 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 [23]. 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 with the extruded name in the subject position, we shall record the whole set of extruders that happened previously.

Hence, the data structure that can be used to represent the whole set of extruders is: *set indexed with a set* Γ_Ω. The reason why two sets are necessary is because the τ-actions do not impose causes. Every extruder will be recorded in the set Γ and in that way we will keep track about the scope of the restricted name, while extruders which are not part of the synchronisations will be saved in Ω.

In the following we show how modified semantics of [22], where we keep track of the extruders in the way described above, is captured by the framework.

Definition 43 *(Modified semantics of Crafa, Varacca and Yoshida).* To capture the causality induced by modified semantics of [22], the data structure Δ is instantiated with the indexed set Γ_Ω.

Now, let us illustrate it with the common example in this section.

Example 18. Let us consider the process $X = \nu a_{\emptyset_{\{*\}}}(\overline{b}^* a^* \mid \overline{c}^* a^* \mid a^*(x))$. By extruding the name a on the channels b and c (by applying rule (Open)), we obtain the process:

$$\nu a_{\{i_1,i_2\}_{\{*,i_1,i_2\}}}(\overline{b}^* a^*[i_1,\{*\}] \mid \overline{c}^* a^*[i_2,\{*\}] \mid a^*(x))$$

As we can notice, keys i_1 and i_2 are added in the data structure Γ_Ω in both sets. From Definition 16 we have that the cause of the action $a(x)$ will be the whole set $\{*, i_1, i_2\}$. By executing the input action we obtain process:

$$\nu a_{\{i_1,i_2\}_{\{*,i_1,i_2\}}}(\overline{b}^* a^*[i_1,\{*\}] \mid \overline{c}^* a^*[i_2,\{*\}] \mid a^*(x)[i_3,\{*,i_1,i_2\}])$$

From the reversible point of view, action $a(x)$ needs to be reversed as the first one (we can notice it in the memory $[i_3,\{*,i_1,i_2\}]$). The other two actions can be reversed in any order.

7. Conclusions

In a concurrent setting, causally-consistent reversibility relates causality and reversibility. Several works [23,19,20,22,21] have addressed causal semantics for the π-calculus, differing on how object causality is modelled. The main difference is in how these semantics treat the parallel extrusion of the same name. Starting from this observation, we have devised a framework for reversible π-calculi parametric with respect to the data structure used to record extrusions of a name. Our framework is highly influenced by [17], in the way we treat reversibility, and from [23] in the way the extrusion of a name is handled. Nonetheless we improve on both. With respect [17] we extend its reversing technique to work with binders, value passing and scope extrusion. Originally in [17] this is not possible due to the 'limitations' of the chosen SOS format. With respect to [23] we abstract away from how causality is defined, so to account for different notions. Moreover, our semantics (thanks to the technique of [17]) is more compositional (e.g., it does not require global rules) with respect to [23] as it does not require structural rules to deal with the splitting of a memory along a parallel composition.

Depending on the underlying data structure, we can obtain different causal semantics representing three different approaches to the parallel extrusion problem. We have shown that the three instances of our framework, using the aforementioned data structures, enjoy typical properties of reversible process algebra and that their reversibility is causally-consistent. Additionally, this is the first time reversible semantics using the causality of [20,22] are introduced and shown to be causally-consistent. We illustrate how three different semantics [23,20,22] can be captured by the framework and we have proved a causal correspondence with the semantics introduced in [20] and an operational correspondence with [23]. Moreover, we depart from [23] by using a different notion of concurrency as explained in Definition 25. This allows us to be more selective and to deem as conflicting two consecutive transition producing/consuming the same prefix, which was not the case in [23].

An important difference between the causal semantics considered in this paper is that the one proposed in [23] satisfies several correctness criteria for causal models [24], while the causal semantics of [19,20] do not. Even if the causal semantics proposed in [23] is the best one according to the aforementioned criteria, there is an open question about causality and

π-calculus which still has to be solved. In [25] it has been shown that the (forward) causal semantics induced by the reversible higher-order π-calculus [26] coincides with the one of [20] while considering reduction semantics (e.g., considering closed systems). That is, the causality considered by [26] coincides with the structural one. One could conclude that while considering closed systems the causal semantics of [26] and [23] are the same, even if [26] does not keep track of causal information about the instantiators. One interpretation of this fact is that [23] uses more causal information than required. But this is still left to be proven. Having a common framework in which one can express all these semantics can lead to a better comparison among them and increase the understanding of causality in the π-calculus.

One of the directions for future work is to develop a behavioural theory of our framework. While exploring causally-consistent reversibility of the different causal semantics that can be expressed in the framework, we are interested in the causality relation as the union of object and subject causality. Therefore, we shall define a causal bisimulation on the framework which does not distinguish object from subject causality. Then, we shall compare it with existing causal sensitive bisimulations and determine whether it is a congruence. Another question would be whether this bisimulation coincides with standard ones [35].

The other directions for future work are to continue working towards a more parametric framework and to compare it with [36,37] and to prove causal correspondence with the revised semantics of [22]. As discussed in Section 6.3, the semantics of [22] is given in terms of prime event structures, where events do not have a unique causal history and the ambiguity of disjunctive causality does not hold when processes with contexts are used, as shown in [24]. In our framework reversibility is causally-consistent; therefore it is necessary to keep track of the causes. Otherwise, by executing backward actions, one could reach a non-consistent state. Therefore we used a modified version of the semantics of [22], where the information about the causes can be tracked. Moreover, we plan to distill the minimum assumptions for predicates `Cause`$(\cdot)$ and `Update`$(\cdot)$ so that causal consistency holds. In this way, we will be able to prove causally consistency in general for our framework (and not for each instance) and then each instance whose predicates satisfy the assumptions will automatically inherit such a property.

Following the approach of [35], we could further abstract our reversing approach and bring it to the (meta-)level of an SOS format dealing with values and binders. One good candidate would be the nominal format [38]. Moreover, it would be interesting to implement our framework in the psi-calculi framework [39].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to thank Ivan Lanese for the valuable discussions on the notion of conflicting transitions. We thank the anonymous reviewers for their helpful suggestions of improvements.

This work has been done partially during the abroad period visit of the first author to the Department of Computing, Imperial College London, supported by IMT School for Advanced Studies Lucca through the Erasmusplus framework.

This work was partially supported by COST Action IC1405: "Reversible computation – extending horizons of computing".

The first author is supported by French ANR project DCore ANR-18-CE25-0007. The second author was supported by the Marie Skłodowska-Curie Individual Fellowship RCADE number 794405 and partially by the Italian INdAM – GNCS project 2020 *Reversible Concurrent Systems: from Models to Languages*. The last author is partially supported by EPSRC EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1, EP/N028201/1, EP/T014709/1, EP/T006544/1 and EP/V000462/1.

Appendix A

In this appendix we give the detailed proofs and further technical material of Section 5.

Proposition 5. *In* $X \xrightarrow{(i,K,j):\overline{b}\langle \nu a_\Delta\rangle} Y$, $\exists C$ *such that* $X = C[\nu a_\Delta X']$. *Similarly in* $X \xrightsquigarrow{(i,K,j):\overline{b}\langle \nu a_\Delta\rangle} Y$, $\exists C$ *such that* $Y = C[\nu a_\Delta X']$.

Proof. The proof is by induction on the derivation of transition $X \xrightarrow{(i,K,j):\overline{b}\langle \nu a_\Delta\rangle} Y$. The interesting cases are when rules (Open) and (Close) are applied and they follow directly from the construction of the rules. □

Proposition 6. *If* $X = C[\nu a_{\Delta'} X']$, *then a transition that has the name a as the object of an output is necessarily of the form* $X \xrightarrow{(i,K,j):\overline{b}\langle \nu a_\Delta\rangle} Y$.

Proof. The proof is by induction on the derivation of transition $X \xrightarrow{(i,K,j):\overline{b}\langle \nu a_\Delta\rangle} Y$. It follows directly from the construction of the rules. □

Proposition 7. *Let* $\Delta = \Gamma$, $X \xrightarrow{(i,K,j):\alpha} Y$ *and* $a = \mathtt{subj}(\alpha)$. *Then* $X = C[\nu a_\Delta X'] \iff K \neq \{*\}$.

Proof. If $X = C[\nu a_\Gamma X']$ and $X \xrightarrow{(i,K,j):\alpha} Y$ with $a = \mathtt{subj}(\alpha)$, then $\Gamma \neq \emptyset$. Having a transition $C[\nu a_\Gamma X'] \xrightarrow{(i,K,j):\alpha} C[\nu a_\Gamma X'']$ the rule (Cause Ref) needs to be applied. Therefore, the cause set $K \subseteq \Gamma$, which means that there exists some key i' such that $i' \in K$.

If $K \neq \{*\}$ and $X \xrightarrow{(i,K,j):\alpha} Y$ and $a = \mathtt{subj}(\alpha)$ it means that the rules (Cause Ref) was applied, since it is the only rule in which cause set K can be updated. Therefore, some $\nu a_\Gamma \in X$ and we can write X as $X = C[\nu a_\Gamma X']$. □

Lemma 4. *If process* $X = C[\nu a_\Delta(Y) \mid Y']$ *is reachable, then* $\nu a_{\Delta'} \notin Y'$, *for all non-empty* Δ *and* Δ'.

Proof. The proof is by induction on the trace that leads to the process X: $X_1 \to \cdots \to X_n \to X$, where X_1 is an initial reversible process,[4] and the last applied rule on the transition $X_n \to X$. The base case is trivial, since for every $\nu a_\Delta \in X_1$, $\mathtt{empty}(\Delta) = \mathtt{true}$. In the inductive case, we have that in the transition $X_n \to X$, the property holds for X_n. We continue by case analysis on the last applied rule on the transition $X_n \to X$:

- rule (Par)

$$\frac{Y_0 \xrightarrow{(i,K,j):\alpha} Y_1 \quad i \notin Y' \quad \mathtt{Bnv}(\alpha) \cap \mathtt{Fnv}(Y') = \emptyset}{Y_0 \mid Y' \xrightarrow{(i,K,j):\alpha} Y_1 \mid Y'}$$

  where the property holds for $Y_0 \mid Y'$. We proceed with the following cases:
  - if $\nu a_\Delta \in Y_0$, then by the inductive hypothesis $\nu a_{\Delta'} \notin Y'$. After the execution of the action α, the property is preserved and it holds in $Y_1 \mid Y'$ as well.
  - if $\nu a_\Delta \in Y'$, then by the inductive hypothesis $\nu a_{\Delta'} \notin Y_0$, when Δ, Δ' are not empty (name a is free in Y', since Δ is not empty). To satisfy the property, we need to show that $\nu a_{\Delta'} \notin Y_1$ holds.
    Let us suppose that $\nu a_{\Delta'} \in Y_1$. Then some restriction $\nu a_{\Delta''}$ needs to belong to Y_0 and the only possibility is $\nu a_{\Delta''} \in Y_0$ when $\mathtt{empty}(\Delta'') = \mathtt{true}$. Since name a is bound in Y_0 and free in Y_1 we have that action α extrudes bound name a and we have $\alpha = \overline{b}\langle \nu a_{\Delta''} \rangle$, where $\mathtt{empty}(\Delta'') = \mathtt{true}$. This is in contradiction with the side condition in the rule Par, since the bound action a is between the free names in Y'. Hence, $\nu a_{\Delta'} \notin Y_1$ and the property holds.
- rule (Com)

$$\frac{Y_0 \xrightarrow{(i,K,j):\overline{b}c} Y_1 \quad Y_0' \xrightarrow{(i,K',j'):b(x)} Y_1' \quad K \bowtie j' \ \wedge \ K' \bowtie j}{Y_0 \mid Y_0' \xrightarrow{(i,\{*\},*):\tau} Y_1 \mid Y_1'\{^{c^i}/_x\}}$$

  where the property holds for $Y_0 \mid Y_0'$. We proceed with the following cases:
  - if $\nu a_\Delta \in Y_0$, then by the inductive hypothesis $\nu a_{\Delta'} \notin Y_0'$, when Δ, Δ' are not empty. We need to show that $\nu a_{\Delta'} \notin Y_1'\{^{c^i}/_x\}$.
    If $c = a$, and $\nu a_\Delta \in Y_0$, then by Proposition 6, action $\overline{b}c$ should be $\overline{b}\langle \nu a_{\Delta''} \rangle$, for some Δ''. In this case the rule Close should be applied.
    If $c \neq a$, then $\nu a_{\Delta'} \notin Y_1'$ since a is not in the object position of the output action $\overline{b}c$. Hence, for the process $Y_1 \mid Y_1'\{^{c^i}/_x\}$ the property holds.
  - if $\nu a_\Delta \in Y_0'$, then by inductive hypothesis $\nu a_{\Delta'} \notin Y_0$, when Δ, Δ' are not empty. Since the action $\overline{b}c$ is executed on the Y_0, there is no possibility for $\nu a_{\Delta'}$ to belong to the process Y_1, and we have $\nu a_{\Delta'} \notin Y_1$ and the property holds.

For the rest of the rules, the property trivially holds. □

Lemma 8. *Given a process* $X = C[\nu a_{\Gamma'} Y]$, *where the context* C *does not contain the restriction of the name* a, *and transition* $\nu a_\Gamma X \xrightarrow{(i,K',j):\pi} \nu a_\Gamma X'$ *with the premise* $X \xrightarrow{(i,K,j):\pi} X'$ *where* $\pi = \overline{a}b$ *or* $\pi = \overline{a}\langle \nu b_{\Gamma_1} \rangle$, *there exists at most one* K *such that* $K \rightsquigarrow_X K'$.

Proof. Before starting with the proof we make a small observation. From the definition of the instantiation relation (Definition 13) we have that the only possibility to have one key to be instantiated by two different keys is if one instantiation relation is defined for the name in the subject position of the prefix and the other one for the name in the object position.

[4] From Definition 8 we have that the reversible process X is initial when all its names are decorated with $*$ and for all restrictions νa_Δ, for some name a, Δ is empty.

Therefore, we can have a process $Z = C[c(x)[i_1, K_1].C'[d(y)[i_2, K_2].\pi[i_3, K_3].Y]$ where $\pi = \overline{a}^{i_1} b^{i_2}$ such that $i_1 \rightsquigarrow_Z i_3$ and $i_2 \rightsquigarrow_Z i_3$.

Let us now proceed with the proof of the lemma and assume that there exist K'' and K''' such that $K'' \neq K'''$ and $K'' \rightsquigarrow_X K'$ and $K''' \rightsquigarrow_X K'$. From Definition 13 and the observation above, we know that one of them is related to the name a and the other one to the name b. Let us say that $K'' \rightsquigarrow_X K'$ is related to the name a and $K''' \rightsquigarrow_X K'$ to the name b. Therefore, the action K''' was a communication on some channel over which name b was sent.

On the other side, from $X \xrightarrow{(i,K''',j):\pi} X'$ where $X = C[\nu a_{\Gamma'} Y]$, we know that $K''' \subset \Gamma'$. This implies that the action K''' was an extrusion of the name a (since this is the only way to update the set Γ') which is not the case. □

In the following we prove that having two transitions and their derivation trees, the conclusions of transitions are the same if and only if they have the same premises. This result is needed to show Lemma 10 when two transitions having the same conclusions are considered. By applying Lemma 9, we can conclude that two transitions are equal.

Lemma 9. *Two derivation trees have the same conclusion if and only if they have the same premises.*

Proof. If the premises of the rules are the same, then it is trivial to show that derivations will reach the same conclusion. In the other direction, let us consider two derivation trees with the same conclusions and premises p_1 and p_2:

$$\frac{p_1}{X \xrightarrow{\mu} Y} \qquad \frac{p_2}{X \xrightarrow{\mu} Y}$$

We proceed with the induction on the derivation tree of the transition $X \xrightarrow{\mu} Y$. For each rule of Figs. 2 and 3 we need to show that only one premise is possible. The interesting cases for us are the rules in which the label changes, in particular the communication rules (Com) and (Close), and the rules in which the contextual cause set K is updated, rules (Open) and (Cause Ref). For the rest of rules, the lemma holds by construction.

- (Cause Ref) Given a transition

$$\frac{p_q}{\nu a_\Delta(X) \xrightarrow{(i,K',j):\pi} Y} \quad a \in \mathtt{subj}(\pi) \quad q \in \{1,2\}$$

the only applicable rule is (Cause Ref) with $Y = \nu a_\Delta X'_{[K'/\ K_q]@i}$, and we have

$$\frac{X \xrightarrow{(i,K_q,j):\pi} X' \quad a \in sub(\pi) \quad q \in \{1,2\} \ \mathtt{Cause}(\Delta, K_q, K', X)}{\nu a_\Delta(X) \xrightarrow{(i,K',j):\pi} \nu a_\Delta X'_{[K'/\ K_q]@i}}$$

Now we want to show that $K_1 = K_2$. Depending on the data structure used for Δ, we have different definitions for the predicate $\mathtt{Cause}(\Delta, K_q, K', X)$:
  - if Δ is a set Γ, then predicate $\mathtt{Cause}(\cdot)$ is defined as $K_q = K'$ when $K_q \subset \Gamma$ or $K' \subseteq \Gamma$ $K_q \rightsquigarrow_X K'$ for $q \in \{1,2\}$ and we have the following cases:
    - □ there does not exist $\nu a_{\Gamma'}$ such that $\nu a_{\Gamma'} \in X$. Then $K_1 = K_2 = \{*\}$. Follows from Proposition 7.
    - □ if $\nu a_{\Gamma'} \in X$, then, we can write X as $X = C[\nu a_{\Gamma'} Y]$, where the context C does not contain the restriction of the name a. Then, we have either $K_q = K'$ or $K' \subseteq \Gamma$ $K_q \rightsquigarrow_X K'$. Therefore, we have the following options:
      - $K_1 = K'$ and $K_2 = K'$, therefore $K_1 = K_2$ as desired;
      - $K' \subseteq \Gamma$ $K_1 \rightsquigarrow_X K'$ and $K_2 \rightsquigarrow_X K'$. Since there is only one substitution per variable this case can only happen when $\pi = \overline{a}b$, for some b, and $K_1 \rightsquigarrow_X K'$ is related to the name a while $K_2 \rightsquigarrow_X K'$ is related to the name b. Then by Lemma 8 we have that K_2 cannot be the cause of the premise (i.e. cannot be the cause of action $\pi = \overline{a}b$).
      - $K_1 = K'$ and $K' \subseteq \Gamma$ $K_2 \rightsquigarrow_X K'$. Since $X = C[\nu a_{\Gamma'} Y]$ and the executed action has $a = \mathtt{subj}(\pi)$, the rule (Cause Ref) is applied so that $K_q \subset \Gamma'$. Then since $K_1 = K'$ we have $K_2, K' \subset \Gamma'$ which is impossible since $K_2 \rightsquigarrow_X K'$.
  - if Δ is an indexed set Γ_w, we have $K' = K_q \cup \{w\}$, i.e. $K_1 \cup \{w\} = K_2 \cup \{w\}$. We can distinguish two cases, depending if w is a $*$ or not:
    - □ if $w = *$, we have $K_1 = K_2 = K'$
    - □ if $w = k$, where $k \neq *$, we need to prove that $K_1 \cup \{k\} = K_2 \cup \{k\}$. Since it is an operation on sets, we need to prove that the key k belongs to both sets K_1 and K_2 or neither one of them. If $k \in K_1$ we have that there is a restriction $\nu a_{\Gamma'_k} \in X$. By the condition $\mathtt{Cause}(\cdot)$ of the rule (Cause Ref) we have $k \in K_2$. Similarly if $k \in K_2$ we need to prove that $k \in K_1$. If $k \notin K_1$, then there does not exist the restriction $\nu a_{\Gamma'_k}$ in the process X; hence k cannot be part of the causal set K_2.
  - if Δ is a set indexed with a set Γ_Ω, then we have $K' = K_q \cup \Omega$, i.e. $K_1 \cup \Omega = K_2 \cup \Omega$. The proof is similar to the one above but we need to prove it for every element in a set Ω.

- (Open) Given a transition

$$\frac{p_q}{\nu a_\Delta(X) \xrightarrow{(i,K',j):\overline{b}\langle \nu a_\Delta\rangle} Y} \quad q \in \{1,2\}$$

the only applicable rule is (Open) and we have $Y = \nu a_{\Delta+i} X'_{[K'/K_q]@i}$ with a rule

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

Now we want to show that $K_1 = K_2$ and $\pi_1 = \pi_2$. Depending on the data structure used for Δ, we have different definitions for the condition $\texttt{Update}(\Delta, K_q, K')$:

- if Δ is a set Γ, we have $K_1 = K_2 = K'$ and we just need to prove that $\pi_1 = \pi_2$. Let us suppose that $\pi_1 = \overline{b}a$; then by Proposition 6 there does not exist a context $C[\bullet]$ such that $X = C[\nu a_{\Gamma'} X'']$; hence from Proposition 5, we have $\pi_2 = \overline{b}a$. Similarly for $\pi_1 = \overline{b}\langle \nu a_{\Gamma'}\rangle$.
- if $\Delta = \Gamma_\Omega$, the cause does not change with the rule Open and the proof is similar to the case above.
- if Δ is an indexed set Γ_w, we have $K' = K_q \cup \{w\}$ and we need to prove that $K_1 = K_2$ and $\pi_1 = \pi_2$. Since set K_q can contain causes of the names a and b, we have $K_q = K_{q_a} \cup K_{q_b}$. Now, let us suppose that $\pi_1 = \overline{b}a$. By Proposition 6 we have that there does not exist a context $C[\bullet]$ such that $X = C[\nu a_{\Gamma'_{w'}} X'']$; hence from Proposition 5, we can conclude that $\pi_2 = \overline{b}a$ and $K_{1_a} = K_{2_a} = \{*\}$. Now we have two cases: if the action b was extruded in the process X, then exist restriction $\nu b_{\Gamma''_{w''}} \in X$ such that $K_{q_b} = \{w'', *\}$; if b was free in X, then $K_{q_b} = \{*\}$.

  Let us suppose that $\pi_1 = \overline{b}\langle \nu a_{\Gamma'_{w'}}\rangle$, by Proposition 6 there exists a context $C[\bullet]$ such that $X = C[\nu a_{\Gamma'_{w'}} X'']$, and from Proposition 5 we have $\pi_2 = \overline{b}\langle \nu a_{\Gamma'_{w'}}\rangle$. For the cause sets K_{q_a} we have two possibilities: if $w' = *$, we have $K_{q_a} = \{*\}$; if $w' = k$, we have $K_{q_a} = \{*, k\}$. For the set K_{q_b} we reason in the same way as when $\pi_1 = \overline{b}a$.

- (Close): Given a transition

$$\frac{p_q}{X \mid Y \xrightarrow{(i,\{*\},*):\tau} Z} \quad q \in \{1,2\}$$

if there exists $\nu a_\Delta \in X$, we can apply only the rule (Close) (by Proposition 6) and get $Z = \nu a_\Delta(X'_{\#i} \mid Y')$ with a rule:

$$\frac{X \xrightarrow{(i,K_q,j):\overline{b}\langle \nu a_\Delta\rangle} X' \quad Y \xrightarrow{(i,K'_q,j'):b(x)} Y'}{X \mid Y \xrightarrow{(i,\{*\},*):\tau} \nu a_\Delta(X'_{\#i} \mid Y')} \quad K_q \bowtie j' \ \wedge \ K'_q \bowtie j$$

Now we want to show that $K_1 = K_2$. Depending on the data structure used for Δ, we have different cases:

- if Δ is a set Γ, we have $K_q \bowtie j' \ \wedge \ K'_q \bowtie j$ and two cases:
  - ☐ if b was a free name that was not bound in the past, we have that $\nu b'_\Gamma \notin X$ and then by Proposition 7 $K_q = K'_q = \{*\}$.
  - ☐ if b is a name that was bound in the past, then there exists a restriction $\nu b_{\Gamma'} \in X$ such that $K_i \neq \{*\}$ and $K'_i = \{*\}$ for $i \in \{1,2\}$ or there exists a restriction $\nu b_{\Gamma'} \in Y$ such that $K_i = \{*\}$ and $K'_i \neq \{*\}$ for $i \in \{1,2\}$. In the first case we have $K'_1 = K'_2 = \{*\}$; hence in order to satisfy the condition of the rule, we have $K_1 = K_2 \bowtie j'$. The second case is similar.
- if Δ is an indexed set Γ_w, the action $\overline{b}\langle \nu a_{\Gamma_w}\rangle$ can be caused through subject and object position in the label, and we need to reason on both names: a and b. We shall divide cause sets K_q into two subsets: $K_q = K_{q_a} \cup K_{q_b}$ and for K'_q we have $K'_q = K'_{q_b}$ since it can be only caused through the name b. The rule Close is used; hence there exists a context $C[\bullet]$, such that $X = C[\nu a_{\Gamma_w} X'']$ and we have three possibilities:
  - ☐ if a restriction $\nu b_{\Gamma'_{w'}}$ is not in X and Y, then $K_{q_b} = K'_{q_b} = \{*\}$. For K_{q_a} we have two cases: if $w = *$, we can conclude that $K_{q_a} = \{*\}$ and we have $K_1 = K_2$; if $w = k''$ for some $k'' \in \Gamma$, we can conclude that $K_{q_a} = \{*, k''\}$ and $K_1 = K_2$.
  - ☐ if a restriction $\nu b_{\Gamma'_{w'}} \in X$ then by Lemma 4 there is no restriction $\nu b_{\Gamma''_{w''}}$ in Y, hence $K'_{q_b} = \{*\}$. For the cause set K_q we reason as in the case when rule (Open) was applied.
  - ☐ if a restriction $\nu b_{\Gamma'_{w'}} \in Y$ then by Lemma 4 there is no restriction $\nu b_{\Gamma''_{w''}}$ in X, hence $K_{q_b} = \{*\}$. For K_{q_a} we have two cases: if $w = *$, we can conclude that $K_{q_a} = \{*\}$ and we have $K_1 = K_2$; if $w = k''$ for some $k'' \in \Gamma$, we can conclude that $K_{q_a} = \{*, k''\}$ and $K_1 = K_2$. From $\nu b_{\Gamma'_{w'}} \in Y$ we know that the rule (Cause Ref) is applied on the process Y and we reason as in the first case of the proof.
- if $\Delta = \Gamma_\Omega$, we have two cases:
  - ☐ if b was a free name, then $K_q = K'_q = \{*\}$

- □ if b is a name that was bound in the past, then by Lemma 4 there exists a restriction $\nu b_{\Gamma'_{\Omega'}} \in X$ such that $K_i \neq \{*\}$ and $K'_i = \{*\}$ for $i \in \{1,2\}$ or exists a restriction $\nu b_{\Gamma'_{\Omega'}} \in Y$ such that $K_i = \{*\}$ and $K'_i \neq \{*\}$ for $i \in \{1,2\}$. In the first case we have: if $\Omega' = \{*\}$ then $K_1 = K_2 = \{*\}$; if $\Omega' \neq \{*\}$ then from the definition of the predicate $\texttt{Cause}(\cdot)$ of the rule (Cause Ref) applied on the process X we have $K_q = K''_q \cup \Omega'$, where K''_q are the cause sets belonging to the premise of the rule (Cause Ref). Then we reason as in the case when the rule (Cause Ref) is applied. The case when $\nu b_{\Gamma'_{\Omega'}} \in Y$ is similar.
- (Com): This case is handled similarly to the case when we considered rule (Close). □

With the following definition we introduce the *normal form* of a reversible process which we will use in this section.

Definition 44 *(Normal form).* The normal form of the reversible process X is given with

$$X = \nu\widetilde{a_{n\Delta_n}} C_n[\nu\widetilde{a_{n-1\Delta_{n-1}}} C_{n-1}[\dots \nu\widetilde{a_{0\Delta_0}} C_0[X_1]]]$$

where the contexts C_i when $i = 0, 1, \dots, n$ (Definition 1) do not contain any restriction on the name a and $\nu\widetilde{a_{l\Delta_l}} = \nu\widetilde{a_{l1\Delta_{l1}}}, \dots, \nu\widetilde{a_{ln\Delta_{ln}}}$.

Notation 2. If the contexts are not relevant, we will write $X = \mathbf{C}[\nu\widetilde{a_{0\Delta_0}} C_0[X_1]]$ where $\mathbf{C} = \nu\widetilde{a_{n\Delta_n}} C_n[\nu\widetilde{a_{n-1\Delta_{n-1}}} C_{n-1}[\dots \nu\widetilde{a_{1\Delta_1}} C_1[]]]$.

With the next properties, we show how in the transition $t : X \xrightarrow{(i,K,j):\alpha} X'$, executing action α influences the resulting process X'. Depending on the nature of action α, just one part of it or the whole process X' will be modified.

Property 1. *Given a process $X = \nu\widetilde{a_{n\Delta_n}} C_n[\nu\widetilde{a_{n-1\Delta_{n-1}}} C_{n-1}[\dots \nu\widetilde{a_{0\Delta_0}} C_0[X_1]]]$, and transition $t : X \xrightarrow{(i,K,j):\alpha} X'$, $\alpha \neq \tau$, executed by any component in X, there are two possibilities for modifying the resulting process X', depending on the nature of the action α:*

- *if $\alpha = \overline{b}\langle \nu a_\Delta \rangle$, for some name b, then transition t modifies the component on which it is executed and all the restrictions on the name a before it;*
- *if $\alpha \neq \overline{b}\langle \nu a_\Delta \rangle$, then transition t modifies only the component on which it is executed:*

Proof. The proof follows directly from the semantics of the framework. □

For instance, consider the process $X = \nu a_{\Delta_n} C_n[\nu a_{\Delta_{n-1}} C_{n-1}[\dots \nu a_{\Delta_0} C_0[X_1]]]$ and transition $t : X \xrightarrow{(i,K,j):\alpha} X'$, where $\alpha \neq \tau$. Let us assume that transition t is performed by the process X_1. Then we have:

- if $\alpha = \overline{b}\langle \nu a_\Delta \rangle$, for some name b; transition t modifies the process X_1 and all the restrictions, since all of them are before X_1, and we have:

$$t : X \xrightarrow{(i,K,j):\overline{b}\langle \nu a_{\Delta_n} \rangle} \nu a_{\Delta_n + i} C_n[\nu a_{\Delta_{n-1}+i} C_{n-1}[\dots \nu a_{\Delta_0 + i} C_0[X'_1]]]$$

  Every time when action α passes the restriction on the name a, rule (Open) is applied and key i is added to Δ_l when $l = 0, 1, \dots, n$.
- if $\alpha \neq \overline{b}\langle \nu a_\Delta \rangle$, then transition t modifies only the process X_1:

$$t : X \xrightarrow{(i,K,j):\alpha} \nu a_{\Delta_n} C_n[\nu a_{\Delta_{n-1}} C_{n-1}[\dots \nu a_{\Delta_0} C_0[X'_1]]]$$

Property 2. *Given a process $X = \nu\widetilde{a_{n\Delta_n}} C_n[\nu\widetilde{a_{n-1\Delta_{n-1}}} C_{n-1}[\dots \nu\widetilde{a_{0\Delta_0}} C_0[X_1]]]$, and transition $t : X \xrightarrow{(i,\{*\},*):\tau} X'$, there are two possibilities for modifying the resulting process X', depending whether the transition t involves one or two contexts:*

- *if transition t involves one context, then considering the synchronisation $t : X_1 \xrightarrow{(i,\{*\},*):\tau} X'_1$ we have:*
  - *if $C_0 = \bullet \mid Y$, transition t modifies just that context, hence*

$$t : \mathbf{C}[\nu\widetilde{a_{0\Delta_0}}(X_1 \mid Y)] \xrightarrow{(i,\{*\},*):\tau} \mathbf{C}[\nu\widetilde{a_{0\Delta_0}}(X'_1 \mid Y)]$$

  - *if $C_0 = \bullet$, the synchronisation can add or remove an element of $\widetilde{a_{0\Delta_0}}$ but not of Δ_0, and it does not change the context $\mathbf{C}$, hence:*

$$t : \mathbf{C}[\nu\widetilde{a_{0\Delta_0}}(X_1)] \xrightarrow{(i,\{*\},*):\tau} \mathbf{C}[\nu\widetilde{a'_{0\Delta_0}}(X'_1)]$$

- *if transition t involves two contexts and rules* (Com) *and* (Com$^\bullet$) *are applied, then transition t modifies just the two contexts. Otherwise if rule* (Close) *and* (Close$^\bullet$) *are applied and transition t involves X_1 and context C_{n-1}, we have:*

$$\widetilde{\nu a_{n\Delta_n}} C_n[\widetilde{\nu a_{n-1\Delta_{n-1}}} C_{n-1}[\dots \widetilde{\nu a_{0\Delta_0}} C_0[X_1]]] \xrightarrow{(i,\{*\},*):\tau}$$
$$\widetilde{\nu a_{n\Delta_n}} C_n[\widetilde{\nu a'_{n-1\Delta_{n-1}}} C'_{n-1}[\dots \widetilde{\nu a_{0\Delta'_0}} C_0[X'_1]]]$$

where transition t modifies both contexts, restriction $\widetilde{\nu a_{n-1\Delta_{n-1}}}$ where it can add or remove element from it, and restrictions $\widetilde{\nu a_{l\Delta_l}}$ when $l = 0, 1, \dots, n-1$ where it can change just Δ_l.

Proof. The proof is straightforward from the rules for communication. □

Notation 3. For the sake of simplicity we shall assume that the vector of names $\widetilde{\nu a_{l\Delta_l}}$ from Definition 44 is a singleton and we shall write: $X = \nu a_{\Delta_n} C_n[\nu a_{\Delta_{n-1}} C_{n-1}[\dots \nu a_{\Delta_0} C_0[X_1]]]$.

Lemma 5 *(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 $\mu_i =_\lambda \mu'_i$.*

Proof. The proof is by case analysis on the form of the transitions t_1 and t_2. The case when transitions t_1 and t_2 execute on exactly the same prefix is impossible since the transitions are concurrent. In particular:

- if $X = C[\overline{a}^* b^*.P]$, for some a, b, then t_1 is a forward transition executed on prefix $\overline{a}^* b^*$ and t_2 is a backward transition executed on history prefix $\overline{a}^* b^*[i_1, K_1]$ such that $i_1, K_1 \in \mu_1$. Then we have $t_2 = t_1^\bullet$ and by Definition 26, transitions t_1 and t_2 are not concurrent, which is not the case.
- if $X = C[\overline{a}^* b^*[i_1, K_1].P]$, for some a, b, then t_1 is a backward transition executed on history prefix $\overline{a}^* b^*[i_1, K_1]$ and t_2 is a forward transition executed on prefix $\overline{a}^* b^*$. Then we have that transition t_2 consumes the prefix freed by transition t_1 and by Definition 26, transitions t_1 and t_2 are not concurrent, which is not the case.

Similarly if the transitions are executed on input prefixes.

We proceed with the proof by considering four main cases depending on whether transitions t_1 and t_2 are synchronisations or not and then proceed with the induction on the structure of the process while checking all possible combination of the rules applied on the transitions t_1 and t_2. We just show interesting cases when the restriction of a name is involved and the reversible process is written in the form $X = \nu a_{\Delta_n} C_n[\nu a_{\Delta_{n-1}} C_{n-1}[\dots \nu a_{\Delta_0} C_0[X_1]]]$ (Notation 3).

We proceed with case analysis on whether transitions t_1 and t_2 are synchronisations or not:

1. t_1 and t_2 are not synchronisations. We need to prove that by changing the order of the transitions we shall obtain the same process. We consider the cases where transitions t_1 and t_2 modify not just their own context, but also other contexts or restrictions.
   Let us assume that transition t_1 modifies process X_1 and that the performed action is $\alpha_1 = \overline{b}\langle \nu a_\Delta \rangle$. By Property 1,we have

$$t_1 : X \xrightarrow{\mu_1} \nu a_{\Delta'_n} C_n[\nu a_{\Delta_{n-1}} C_{n-1}[\dots \nu a_{\Delta'_0} C_0[X'_1]]]$$

   where $\mu_1 = (i_1, K_1, j_1) : \overline{b}\langle \nu a_\Delta \rangle$. Let α_2 be the action of the transition t_2. If $a \notin \alpha_2$ then t_2 modifies just its own context (Property 1) and it is not prevented by the restrictions on a. Let us consider the case when $a \in \alpha_2$. We continue the proof by the induction on the structure of the process.
   - The base case of induction is to prove that if

$$\nu a_\Delta (X_1 \mid X_2) \xrightarrow{\mu_1} \nu a_{\Delta'_1} (X'_1 \mid X_2) \xrightarrow{\mu_2} \nu a_{\Delta'} (X'_1 \mid X'_2)$$

     then

$$\nu a_\Delta (X_1 \mid X_2) \xrightarrow{\mu'_2} \nu a_{\Delta'_2} (X_1 \mid X'_2) \xrightarrow{\mu'_1} \nu a_{\Delta'} (X'_1 \mid X'_2)$$

     Since t_1 has performed action $\alpha_1 = \overline{b}\langle \nu a_\Delta \rangle$, rules (Open) and (Open$^\bullet$) can be used. We consider just the interesting cases when on t_1 is applied rule (Open) and continue with the case analysis on the rules that can be applied on t_2 such that $a \in \alpha_2$.

– Rule (Open) applied on t_2. We have

$$\nu a_\Delta(X_1 \mid X_2) \xrightarrow{(i_1,K_1,j_1):\overline{b}\langle \nu a_\Delta\rangle} \nu a_{\Delta+i_1}(X_1' \mid X_2) \xrightarrow{(i_2,K_2,j_2):\overline{c}\langle \nu a_{\Delta+i_1}\rangle} \nu a_{\Delta+i_1+i_2}(X_1' \mid X_2')$$

where $X_2 \xrightarrow{(i_2,K,j_2):\overline{c}\langle \nu a_{\Delta'}\rangle} X_2'$. If $i_1 \in K_2$ then transition t_1 causes transition t_2 and this is not the case (they are concurrent). If $i_1 \in K$, then by definition of the predicate `Update(·)` (on the rule (Open)) for the three data structures that we considered, we will have $i_1 \in K_2$ and again, we have that t_1 causes transition t_2 which is not the case. Therefore, $i_1 \notin K, K_2$ and we can safely commute transitions and obtain:

$$\nu a_\Delta(X_1 \mid X_2) \xrightarrow{(i_2,K_2,j_2):\overline{c}\langle \nu a_\Delta\rangle} \nu a_{\Delta+i_2}(X_1 \mid X_2') \xrightarrow{(i_1,K_1,j_1):\overline{b}\langle \nu a_{\Delta+i_2}\rangle} \nu a_{\Delta+i_1+i_2}(X_1' \mid X_2')$$

as desired. The labels of transitions t_1 and t_1' are not equal; they are label equivalent, i.e. $\mu_1 =_\lambda \mu_1'$.

– Rule (Cause Ref) applied on t_2. We have

$$\nu a_\Delta(X_1 \mid X_2) \xrightarrow{(i_1,K_1,j_1):\overline{b}\langle \nu a_\Delta\rangle} \nu a_{\Delta+i_1}(X_1' \mid X_2) \xrightarrow{(i_2,K_2,j_2):\alpha} \nu a_{\Delta+i_1}(X_1' \mid X_2')$$

with $X_2 \xrightarrow{(i_2,K,j_2):\alpha} X_2'$ and $a \in \mathtt{subj}(\alpha)$. If $i_1 \in K_2$ then transition t_1 causes transition t_2 but this is not the case as they are concurrent. If $i_1 \in K$, then by definition of the predicate `Cause(·)` from the rule (Cause Ref) we have three cases: if $\Delta = \Gamma$, then by $i_1 \in K$ and $i_1 \notin K_2$, we have $K \rightsquigarrow_{X_2} K_2$ which implies that i_1 is a synchronisation and that is not the case; if $\Delta = \Gamma_w$, then by the predicate `Cause(·)`, new cause set K_2 is the union of the old cause set K and w, therefore $i_1 \in K$ implies $i_1 \in K_2$, which is not the case; similarly if $\Delta = \Gamma_\Omega$. Hence $i_1 \notin K, K_2$ and we can commute transitions:

$$\nu a_\Delta(X_1 \mid X_2) \xrightarrow{(i_2,K_2,j_2):\alpha} \nu a_\Delta(X_1 \mid X_2') \xrightarrow{(i_1,K_1,j_1):\overline{b}\langle \nu a_\Delta\rangle} \nu a_{\Delta+i_1}(X_1' \mid X_2')$$

• In the inductive case it is necessary to show that if $X \xrightarrow{\mu_1} Y \xrightarrow{\mu_2} Z$ and $X \xrightarrow{\mu_2'} Y_1 \xrightarrow{\mu_1'} Z$, then the following holds

$$\nu a_\Delta X \xrightarrow{\mu_1} \nu a_{\Delta_1'} Y \xrightarrow{\mu_2} \nu a_{\Delta'} Z \text{ and } \nu a_\Delta X \xrightarrow{\mu_2'} \nu a_{\Delta_2'} Y_1 \xrightarrow{\mu_1'} \nu a_{\Delta'} Z$$

$$Z_i \mid X \xrightarrow{\mu_1} Z_i \mid Y \xrightarrow{\mu_2} Z_i \mid Z \text{ and } Z_i \mid X \xrightarrow{\mu_2'} Z_i \mid Y_1 \xrightarrow{\mu_1'} Z_i \mid Z$$

Both subcases are straightforward.

2. t_2 is a synchronisation and t_1 is not. We observe the case when t_1 is performing an action $\overline{b}\langle \nu a_\Delta\rangle$ (the rest of the cases are straightforward). In this case, the applied rule could be (Open) or (Open•).
If t_2 is a synchronisation which does not involve name a or involve just one component of the process X, then by Property 2 the case is trivial.
We shall consider the case when transition t_2 involves two contexts $C_i[\bullet]$ and $C_j[\bullet]$ of the process X written as:

$$X = \nu a_{\Delta_n} C_n[\ldots \nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} C_j[\nu a_{\Delta_0} C_0[X_1]]]]$$

and name a is used in the object position of the label α_2 of t_2. Then rules (Close) and (Close•) can be applied. Now we proceed with the induction on the structure of the process.

• In the base case, since transitions will modify contexts just up to νa_{Δ_i}, we can reason on process X written as:

$$X = \nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} C_j[X_1]]]$$

Now we combine rules applied on transition t_1 with the one applied on t_2 and we have:

– Rule (Open) applied on t_1 and rule (Close) on t_2. We assume that transition t_1 is executed on the component X_1. Then we have:

$$\nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} C_j[X_1]] \xrightarrow{(i_1,K_1,j_1):\overline{b}\langle \nu a_{\Delta_i}\rangle} \twoheadrightarrow$$
$$\nu a_{\Delta_i+i_1} C_i[\ldots \nu a_{\Delta_j+i_1} C_j[X_1']] \xrightarrow{(i_2,\{*\},*):\tau} \twoheadrightarrow$$
$$\nu a_{\Delta_i+i_1} \nu a_{\Delta+i_1} C_i'[\ldots \nu a_{\Delta_j'+i_1} C_j'[X_1']] = Z$$

where $\nu a_{\Delta+i_1} \in C_i[\ldots \nu a_{\Delta_j+i_1} C_j[X_1']]$ (from Proposition 5). To permute transitions, we need to be sure that transition t_1 does not cause transition t_2. Since t_2 is a synchronisation, we need to check if t_1 causes the transitions that are involved in the communication; but then we have to check if the lemma holds for transitions that are not synchronisations and that is done in Case 1. Hence, we commute transitions and obtain:

$$\nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} C_j[X_1]] \xrightarrow{(i_2,\{*\},*):\tau} \twoheadrightarrow$$
$$\nu a_{\Delta_i} \nu a_{\Delta} C_i'[\ldots \nu a_{\Delta_j'} C_j'[X_1]] \xrightarrow{(i_1,K_1,j_1):\overline{b}\langle \nu a_{\Delta_i}\rangle} \twoheadrightarrow$$
$$\nu a_{\Delta_i+i_1} \nu a_{\Delta+i_1} C_i'[\ldots \nu a_{\Delta_j'+i_1} C_j'[X_1']] = Z$$

where $\nu a_{\Delta} \in C_i[\ldots \nu a_{\Delta_j} C_j[X_1]]$. We have $Z = Z$, as desired.

– Rule (Open•) applied on t_1 and rule (Close) on the t_2. We have:

$$\nu a_{\Delta_i+i_1} C_i[\ldots \nu a_{\Delta_j+i_1} C_j[X_1]] \overset{(i_1,K_1,j_1):\overline{b}\langle \nu a_{\Delta_i}\rangle}{\rightsquigarrow}$$
$$\nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} C_j[X_1']] \xrightarrow{(i_2,\{*\},*):\tau} \twoheadrightarrow \nu a_{\Delta_i} \nu a_{\Delta'} C_i'[\ldots \nu a_{\Delta_j'} C_j'[X_1']] = Z$$

where $\nu a_{\Delta} \in C_i[\ldots \nu a_{\Delta_j} C_j[X_1']]$. It is not possible that backward transition t_1 causes transition t_2 and we can swap transitions and obtain:

$$\nu a_{\Delta_i+i_1} C_i[\ldots \nu a_{\Delta_j+i_1} C_j[X_1]] \xrightarrow{(i_2,\{*\},*):\tau} \twoheadrightarrow$$
$$\nu a_{\Delta_i+i_1} \nu a_{\Delta'+i_1} C_i'[\ldots \nu a_{\Delta_j'+i_1} C_j'[X_1]] \overset{(i_1,K_1,j_1):\overline{b}\langle \nu a_{\Delta_i}\rangle}{\rightsquigarrow}$$
$$\nu a_{\Delta_i} \nu a_{\Delta'} C_i'[\ldots \nu a_{\Delta_j'} C_j'[X_1']] = Z$$

where $\nu a_{\Delta+i_1} \in C_i[\ldots \nu a_{\Delta_j+i_1} C_j[X_1]]$.

– Similarly for the rules (Open) and (Open•) combined with rule (Close•).

- The inductive case is trivial since t_1 only modifies processes in the context, not the context by itself. Considering the synchronisation $t : X_1 \xrightarrow{(i,\{*\},*):\tau} X_1'$, we have $t : C[\nu a_{\Delta}(X_1 \mid X_2)] \xrightarrow{(i,\{*\},*):\tau} C[\nu a_{\Delta}(X_1' \mid X_2)]$ where we can notice that transition t modified just X_1 (Property 2).

3. The case when t_1 is a synchronisation and t_2 is not, is similar to the one above.
4. t_1 and t_2 are synchronisations. We consider the cases when synchronisations involve two different components of the process X. Assume that t_1 is a synchronisation between process X_1 where output is executed and context C_j where input is performed; while t_2 is a synchronisation between input in context C_i and output in C_k. Since transitions will modify contexts just up to νa_{Δ_i}, we can reason on process X written as:

$$X = \nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} C_j[\ldots \nu a_{\Delta_k} C_k[X_1]]]$$

We continue with the case analysis depending whether name a is used in the subject or in the object position of the output actions involved in the communications t_1 and t_2.

- name a is in the object position in both output actions involved in the communications t_1 and t_2; in this case, rules (Close) and (Close•) can be applied. Let us consider the case when rule (Close) is applied on both t_1 and t_2; the rest of the cases are similar. We have

$$\nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} C_j[\ldots \nu a_{\Delta_k} C_k[X_1]]] \xrightarrow{(i_1,\{*\},*):\tau} \twoheadrightarrow$$
$$\nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} \nu a_{\Delta} C_j'[\ldots \nu a_{\Delta_k'} C_k[X_1']]] \xrightarrow{(i_2,\{*\},*):\tau} \twoheadrightarrow$$
$$\nu a_{\Delta_i} \nu a_{\Delta_l''} C_i'[\ldots \nu a_{\Delta_j'} \nu a_{\Delta''} C_j'[\ldots \nu a_{\Delta_k''} C_k'[X_1']]]$$

where from Proposition 5, we have $\nu a_{\Delta} \in C_j[\ldots \nu a_{\Delta_k} C_k[X_1]]$ and $\nu a_{\Delta_l''} \in C_i[\ldots \nu a_{\Delta_j} \nu a_{\Delta'} C_j'[\ldots \nu a_{\Delta_k'} C_k[X_1']]]$. To permute transitions, we need to ensure that transition t_1 is not the cause of transition t_2. Since both transitions are synchronisations we need to check if the output transition involved in t_1 causes the output transition involved in t_2. This is covered with Case 1 when we proved that the lemma holds for single transitions.

Now we can safely swap the transitions and transition t_2' is:

$$\nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} C_j[\ldots \nu a_{\Delta_k} C_k[X_1]]] \xrightarrow{(i_2,\{*\},*):\tau\!\!\!\!\rightarrow}$$
$$\nu a_{\Delta_i} \nu a_{\Delta_l''} C_i'[\ldots \nu a_{\Delta_j'} C_j[\ldots \nu a_{\Delta_k'} C_k'[X_1]]]$$

where $\nu a_{\Delta_l} \in C_i[\ldots \nu a_{\Delta_j} C_j[\ldots \nu a_{\Delta_k} C_k[X_1]]]$. The derivation for the transition t_1' is:

$$\nu a_{\Delta_i} \nu a_{\Delta_l''} C_i'[\ldots \nu a_{\Delta_j'} C_j[\ldots \nu a_{\Delta_k'} C_k'[X_1]]] \xrightarrow{(i_1,\{*\},*):\tau\!\!\!\!\rightarrow}$$
$$\nu a_{\Delta_i} \nu a_{\Delta_l''} C_i'[\ldots \nu a_{\Delta_j'} \nu a_{\Delta''} C_j'[\ldots \nu a_{\Delta_k''} C_k'[X_1']]]$$

where $\nu a_{\Delta''} \in C_j[\ldots \nu a_{\Delta_k'} C_k'[X_1]]$, as desired.

- name a is in the object position in the output transition involved in the communication t_1 and in the subject position in the output transition involved in communication t_2. In this case, rules (Close) and (Close$^\bullet$) can be applied on t_1 and (Com) and (Com$^\bullet$) on t_2. Let us consider the case when (Close) is applied on t_1 and (Com) on t_2. The rest of the cases are similar. By executing the rule (Close) on t_1, we have:

$$\nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} C_j[\ldots \nu a_{\Delta_k} C_k[X_1]]] \xrightarrow{(i_1,\{*\},*):\tau\!\!\!\!\rightarrow}$$
$$\nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} \nu a_{\Delta} C_j'[\ldots \nu a_{\Delta_k+i_1} C_k[X_1']]]$$

where $\nu a_{\Delta} \in C_j[\ldots \nu a_{\Delta_k} C_k[X_1]]]$ (Proposition 5). By executing the rule (Com) we are changing just contexts C_i and C_k, not the restrictions.

$$\nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} \nu a_{\Delta} C_j'[\ldots \nu a_{\Delta_k+i_1} C_k[X_1']]] \xrightarrow{(i_2,\{*\},*):\tau\!\!\!\!\rightarrow}$$
$$\nu a_{\Delta_i} C_i'[\ldots \nu a_{\Delta_j} \nu a_{\Delta} C_j'[\ldots \nu a_{\Delta_k+i_1} C_k'[X_1']]]$$

To permute transitions, we need to ensure that transition t_1 is not the cause of transition t_2. Since both transitions are synchronisations we need to check if the output action involved in t_1 causes the output or the input action involved in transition t_2. This is covered with Case 1 when we proved that the lemma holds for single transitions t_1 and t_2. Hence, we can swap transitions and obtain:

$$\nu a_{\Delta_i} C_i[\ldots \nu a_{\Delta_j} C_j[\ldots \nu a_{\Delta_k} C_k[X_1]]] \xrightarrow{(i_2,\{*\},*):\tau\!\!\!\!\rightarrow}$$
$$\nu a_{\Delta_i} C_i'[\ldots \nu a_{\Delta_j} C_j[\ldots \nu a_{\Delta_k} C_k'[X_1]]] \xrightarrow{(i_1,\{*\},*):\tau\!\!\!\!\rightarrow}$$
$$\nu a_{\Delta_i} C_i'[\ldots \nu a_{\Delta_j} \nu a_{\Delta} C_j'[\ldots \nu a_{\Delta_k+i_1} C_k'[X_1']]]$$

where $\nu a_{\Delta} \in C_j[\ldots \nu a_{\Delta_k} C_k'[X_1]]]$ since transition t_2 does not change restrictions. □

In the following we give some properties defined on the transitions, necessary to show the main result of this section: causal-consistency of the framework.

Definition 45. Two transitions t_1 and t_2 are *prefix equivalent*, written $t_1 =_p t_2$ if they add or remove the same past element $\pi[i, K]$ from the history context of a process.

We would like to remind that a history context is built from the past prefixes. For example, in the process $\overline{a}^* b^*[i, K].c^*(x).\mathbf{P}$, the history context is $\mathtt{H} = \overline{a}^* b^*[i, K].\bullet$, while in the process $\overline{a}^* b^*.c^*(x).\mathbf{P}$, the history context is empty (i.e. $\mathtt{H} = \bullet$).

Example 19. Given the process $\overline{a}^* b^*.P_1 \mid \overline{a}^* b^*.P_2$, we have two transitions

$$t_1 : \overline{a}^* b^*.P_1 \mid \overline{a}^* b^*.P_2 \xrightarrow{(i,\{*\},*):\overline{a}b\!\!\!\!\rightarrow} \overline{a}^* b^*[i, \{*\}].P_1 \mid \overline{a}^* b^*.P_2$$
$$t_2 : \overline{a}^* b^*.P_1 \mid \overline{a}^* b^*.P_2 \xrightarrow{(i,\{*\},*):\overline{a}b\!\!\!\!\rightarrow} \overline{a}^* b^*.P_1 \mid \overline{a}^* b^*[i, \{*\}].P_2$$

Transitions t_1 and t_2 are not the same, but they are prefix equivalent because they add the same past element into the history. The LTS ensures that keys are unique in the process.

Lemma 10. *In a trace, if two transitions t_1 and t_2 are prefix equivalent, then $t_1 = t_2^\bullet$.*

Proof. If they are prefix equivalent, we know that $i_1 = i_2$ and we have that the transitions happened on the same component in the parallel composition. Therefore $t_1 : X \xrightarrow{(i_1,K_1,j_1):\alpha} Y$ and $t_1 : Y \overset{(i_1,K_1,j_1):\alpha}{\rightsquigarrow} X$. Then $t_2^\bullet = X \xrightarrow{(i_1,K_1,j_1):\alpha} Y$. From Lemma 9, we have $t_1 = t_2^\bullet$. □

Lemma 11. *If transitions t_1 and t_2 are prefix equivalent, coinitial and they are on exactly the same prefix in a process, then $t_1 = t_2$.*

Proof. The proof follows from the fact that keys are unique and that transitions are coinitial on the same prefix. □

The following lemma states that reversible computation can be rearranged as the backward-only transitions, followed by the forward-only ones.

Lemma 12 *(Parabolic traces). Let s be a trace. Then there exist a backward-only trace r and a forward-only trace r' such that $s \sim r; r'$.*

Proof. The proof is by induction on the length of s and the distance between the very first transition in s and the pair of transitions contradicting the statement of the lemma. Let suppose that this is a pair of transitions $t_1; t_2$. Then we have:

$$t_1 : X \xrightarrow{(i_1,K_1,j_1):\alpha_1} Y \qquad t_2 : Y \overset{(i_2,K_2,j_2):\alpha_2}{\rightsquigarrow} Z$$

We have two cases depending whether transitions t_1 and t_2 are concurrent or not.

- if t_1 and t_2 are concurrent; then we can apply Lemma 5 and swap them. In this way we decrease the distance between the very first transition in s and the pair of transitions contradicting the statement of the lemma.
- if t_1 and t_2 are not concurrent, then we have two possibilities:
  - t_1 and t_2 are causally related. If structural causality is concerned, then they are performed on the same component of the process. Since they are consecutive and in opposite directions they are prefix equivalent. From Lemma 10 we have $t_2 = t_1^\bullet$ and since $t_1; t_1^\bullet \sim \epsilon$ we can decrease the length of s on which we can apply the induction hypothesis. Regarding to contextual causality, we have that this case is impossible. They cannot be object dependent since transitions t_1 and t_2 are consecutive and t_2 is the backward one.
  - transition t_2 is the reverse of transition t_1, i.e., $t_2 = t_1^\bullet$. Then as in the case above since $t_1; t_1^\bullet \sim \epsilon$ we can decrease the length of s on which we apply the induction hypothesis. □

With the next lemma we show that if s_1 and s_2 are two coinitial and cofinal traces and s_2 is made just of forward transitions, then there exists a forward-only trace s_1' equivalent to s_1. Intuitively, backward transitions of the trace s_1 can be deleted, since s_1 and s_2 are coinitial and cofinal and s_2 is forward-only.

Lemma 13. *Let us denote with s_1 and s_2 two coinitial and cofinal traces, where s_2 is forward only. Then there exists a forward-only trace s_1', shorter or equal to s_1, such that $s_1 \sim s_1'$.*

Proof. The proof is by induction on the length of s_1. If s_1 is forward-only then $s_1' = s_1$. If not, by Lemma 12, we can assume that s_1 is parabolic, and write it as $s_1 = u; t_1; t_2; v$ where $t_1; t_2$ is the only pair of consecutive transitions in the opposite direction; $u; t_1$ is backward-only and $t_2; v$ is forward-only. Since traces s_1 and s_2 are coinitial and cofinal and s_2 is forward-only, we can notice that the history element which transition t_1 takes out of the history, some transition in $t_2; v$ needs to put back, otherwise, the difference will stay visible (i.e. s_1 and s_2 would not be cofinal). Let us denote with t' the first such transition. To preserve the same target in the end of the traces s_1 and s_2, we have that transitions t' and t_1 are prefix equivalent, hence from Lemma 10 we have $t' = t_1^\bullet$. We can rewrite s_1 as $u; t_1; t_2; v_1; t'; v_2$ where trace $t_2; v_1; t'; v_2$ is forward-only.

We proceed by showing that t_1 is concurrent with all transitions up to t'. Let us suppose the opposite, namely that there exists some transition t'' between t_1 and t' such that t_1 and t'' are not concurrent. From Definition 26, we can distinguish three cases:

- if t_1 and t'' are structural causal then we have a contradiction with the hypothesis that t' is the first transition that will put back the history element that t_1 deletes.
- the case when t_1 and t'' are object causal is impossible, since t_1 is a backward transition and t'' is forward.
- t_1 is a backward transition and t'' is a forward transition such that t'' consumes the prefix freed by transition t_1. Then we have a contradiction with the hypothesis that t' is the first transition that will put back the history element that t_1 deletes.

We can conclude that transition t_1 is concurrent with all transitions between t_1 and t'. By Lemma 5 we can swap t_1 with each transition up to t' and by Definition 28 we have $s_1 \sim u; t_2; v_1; t_1; t'; v_2$. By the same definition we have $s_1 \sim$

$u; t_2; v_1; v_2$ (since $t_1^\bullet = t'$, we can erase the transitions because $t_1; t' \sim \epsilon$). In this way, the length of s_1 decreases and we can apply the inductive hypothesis. □

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

Proof. Let us denote two traces with s_1 and s_2. If $s_1 \sim s_2$ then from the definition of $\sim$ (Definition 28) we can conclude that they are coinitial and cofinal.

Let us suppose that s_1 and s_2 are coinitial and cofinal. From Lemma 12 we can suppose that they are parabolic. We shall reason by induction on the lengths of s_1, s_2 and on the depth of the very first disagreement between them. We shall denote it with the pair t_1, t_2. Then we can write the traces s_1 and s_2 as

$$s_1 = u_1; t_1; v_1 \qquad s_2 = u_2; t_2; v_2$$

where $u_1 \sim u_2$. Depending on whether t_1 and t_2 are forward or not, we have the following cases:

- t_1 is forward and t_2 is backward. Since s_1 is parabolic we have that u_1 is backward-only and v_1 is forward-only. From $u_1 \sim u_2$ we have that u_1 and u_2 are coinitial and cofinal; hence the traces $t_1; v_1$ and $t_2; v_2$ are coinitial and cofinal (s_1 and s_2 are cofinal) where $t_1; v_1$ is forward only.
  Now we can apply Lemma 13 on the traces $t_1; v_1$ and $t_2; v_2$ and we have that there exists a trace s_2' (forward-only), shorter or equal to $t_2; v_2$ such that $s_2' \sim t_2; v_2$. If it is equal then t_2 needs to be forward and this is in contradiction with the fact that t_2 is backward. If it is shorter then we proceed by induction with $u_2; s_2'$ shorter.
- t_1 and t_2 are forward. Then $t_1; v_1$ and $t_2; v_2$ are coinitial, cofinal and forward-only. We have two cases depending on whether t_1 and t_2 are concurrent or not.
  - if t_1 and t_2 are concurrent then whatever t_1 puts in the history, v_2 needs to do the same. Let t_1' be the first such transition, then $t_1' \in v_2$ and $t_1' =_p t_1$. Now we can rewrite $t_2; v_2$ as $t_2; v_2'; t_1'; v_2''$ and show that t_1' is concurrent with all transitions in v_2':
    - □ t_1' is the first transition on the same prefix as t_1 (since $t_1; v_1$ and $t_2; v_2$ are coinitial, cofinal and forward-only). Hence, it is not structural causal with any transition in $t_2; v_2'$.
    - □ from $t_1' =_p t_1$, cause sets of both transitions are the same and since t_1 is coinitial with $t_2; v_2'$ and $t_2; v_2'$ are forward-only, transition t_1 cannot have as contextual cause any transition from $t_2; v_2'$.

    We can conclude that transitions t_1 and t_2 are concurrent and from Lemma 5 we have:

    $$t_2; v_2 = t_2; v_2'; t_1'; v_2'' \sim t_1'; t_2; v_2'; v_2''.$$

    Since $t_1' =_p t_1$, they are on exactly the same prefix and they are coinitial, from Lemma 11 we have that $t_1' = t_1$. Without changing the length of s_1 and s_2 we obtain the first disagreement pair later and we can rely on the inductive hypothesis.
  - the case when t_1 and t_2 are causally related is impossible since they are both forward, and coinitial.
- The proof is similar if both transitions t_1 and t_2 are backward. here □

Appendix B

In this appendix we report the full proofs from Section 6.1.1. We start by recalling the semantics of $R\pi$ [23] in Fig. B.9. For more information see [23].

Before proving the operational correspondence of the encoding we shall define the operation @ on the $R\pi$ memories as follows:

Definition 46. We define $m_1@m_2$ by structural induction on m_1:

$$\langle\rangle@m_2 = m_2$$
$$(e \cdot m_1)@m_2 = e \cdot (m_1@m_2)$$

We assume that the cons operator $\cdot$ has precedence over @. We remark that the operation @ does not appear in [23].

Assumption 1. We always choose $[\![X, \langle\rangle, S]\!]$ in such a way that $\mathtt{Bnv}([\![X, \langle\rangle, S]\!]) \cap \mathtt{Fnv}([\![X, \langle\rangle, S]\!]) = \emptyset$.

Before stating the correctness of the encoding, we need some auxiliary results. First, we can view any $R\pi$ process R as a context C^R composed of parallel and restriction operators annotated with the set Γ, containing numbered holes filled by monitored processes. Hence, we will use the following notation.

$$(\text{IN+})\ \frac{i \notin m \quad j = inst_m(b)}{m \rhd b(c).P \xrightarrow{(i,j,*):m[b(x)]} \langle i, *, b[*/x]\rangle.m \rhd P}$$

$$(\text{OUT+})\ \frac{i \notin m \quad j = inst_m(b)}{m \rhd \overline{b}a.P \xrightarrow{(i,j,*):m[\overline{b}a]} \langle i, *, \overline{b}a\rangle.m \rhd P}$$

$$(\text{PAR+})\ \frac{R \xrightarrow{(i,j,k):\alpha} R' \quad i \notin S \quad \mathtt{Bnv}(\alpha) \cap \mathtt{Fnv}(S) = \emptyset}{R \mid S \xrightarrow{(i,j,k):\alpha} R' \mid S}$$

$$(\text{COM+})\ \frac{R \xrightarrow{(i,j,k):\overline{b}a} R' \quad S \xrightarrow{(i,j,k):b(x)} S' \quad k =_* j' \wedge k' =_* j}{R \mid S \xrightarrow{(i,*,*):\tau} R' \mid S'_{[a/x]@i}}$$

$$(\text{CAUSE REF+})\ \frac{R \xrightarrow{(i,j,k):\alpha} R' \quad a \in \mathtt{subj}(\alpha) \qquad \Gamma \neq \emptyset \quad k = k' \vee \exists k' \in \Gamma\ k \rightsquigarrow_R k'}{\nu a_\Gamma(R) \xrightarrow{(i,j,k'):\alpha} \nu a_\Gamma(R'_{[k'/k]@i})}$$

$$(\text{OPEN+})\ \frac{R \xrightarrow{(i,j,k):\alpha} R' \quad \alpha = \overline{b}a \vee \alpha = \overline{b}\langle \nu a_{\Gamma'}\rangle}{\nu a_\Gamma(R) \xrightarrow{(i,j,k):\overline{b}\langle \nu a_\Gamma\rangle} \nu a_{\Gamma+i}(R')}$$

$$(\text{CLOSE+})\ \frac{R \xrightarrow{(i,j,k):\overline{b}\langle \nu a_\Gamma\rangle} R' \quad S \xrightarrow{(i,j',k'):b(x)} S' \quad k =_* j' \wedge k' =_* j}{R \mid S \xrightarrow{(i,*,*):\tau} \nu a_\Gamma(R' \mid S'_{[a/x]@i})}$$

$$(\text{NEW+})\ \frac{R \xrightarrow{(i,j,k):\alpha} R' \quad a \notin \alpha}{\nu a_\Gamma(R) \xrightarrow{(i,j,k):\alpha} \nu a_\Gamma(R')} \qquad (\text{MEM+})\ \frac{R \equiv S \xrightarrow{(i,j,k):\alpha} S' \equiv R'}{R \xrightarrow{(i,j,k):\alpha} R'}$$

$$(\text{SPLIT})\ m \rhd (P \mid Q) \equiv \langle\uparrow\rangle.m \rhd P \mid \langle\uparrow\rangle.m \rhd Q$$

$$(\text{RES})\ m \rhd \nu a(P) \equiv \nu a_\emptyset(m \rhd P) \quad \text{with } a \notin m$$

Fig. B.9. $R\pi$ semantics.

Notation 4. We write an $R\pi$ process R as $C^R[1 \mapsto m_1 \rhd P_1, \ldots, n \mapsto m_n \rhd P_n]$, or, more compactly, as $C^R_{i\in\{1,\ldots,n\}}[i \mapsto m_i \rhd P_i]$. We may drop R when not relevant. If $R = C^R[1 \mapsto m_1 \rhd P_1, \ldots, n \mapsto m_n \rhd P_n]$ then we denote by $R@m$ the process $C^R[1 \mapsto m_1@m \rhd P_1, \ldots, n \mapsto m_n@m \rhd P_n]$.

We can use the notation above to establish useful properties of the translation of the processes from the framework. Most of the terminology and the proof schemas are adapted from [16] and extended to work with the π-calculus. In what follows, we prove some auxiliary properties necessary for the proof of the main theorem.

Lemma 14. *Let $S :: X$ be a process from the framework. There exist $C^{[\![X,\langle\rangle,S]\!]}$, n, $m_1, \ldots, m_n$, $P_1, \ldots, P_n$ such that, for each $R\pi$ memory m, such that for every name $a \in \mathtt{Fnv}(m)$, $\nu a_\Gamma \notin R$ for all Γ and for some i', k', c, b, d and event $\langle i', k', c[b/d]\rangle \notin m$, we have*

$$[\![X, m, S]\!] = C^{[\![X,\langle\rangle,S]\!]}[1 \mapsto m_1@m \rhd P_1, \ldots, n \mapsto m_n@m \rhd P_n]$$

Proof. The proof is by induction on the derivation of $[\![X, m, S]\!]$.

- if $X = \mathbf{P}$ then $[\![\mathbf{P}, m, S]\!] = m \rhd \sigma(S, \mathbf{P})$ as desired (with $C^{[\![X,\langle\rangle,S]\!]}[\bullet] = \bullet$, $n = 1$, $m_1 = \langle\rangle$ and $P_1 = \sigma(S, \mathbf{P})$);
- if $X = \overline{b}^{j'} a^j[i, K].X'$ then

  $$[\![\overline{b}^{j'} a^j[i, K].X', m, S]\!] = [\![X', \langle i, K, \sigma(S, \overline{b}a)\rangle.m, S]\!]$$

  and by inductive hypothesis

  $$[\![X', \langle i, K, \sigma(S, \overline{b}a)\rangle.m, S]\!] =$$
  $$C^{[\![X',\langle\rangle,S]\!]}[1 \mapsto m_1@\langle i, K, \sigma(S, \overline{b}a)\rangle.m \rhd P_1, \ldots, n \mapsto m_n@\langle i, K, \sigma(S, \overline{b}a)\rangle.m \rhd P_n]$$

  as desired, by selecting $C^{[\![X,\langle\rangle,S]\!]} = C^{[\![X',\langle\rangle,S]\!]}$;

- if $X = X' \mid X''$ then $[\![X' \mid X'', m, S]\!] = [\![X', \langle\uparrow\rangle.m, S]\!] \mid [\![X'', \langle\uparrow\rangle.m, S]\!]$ and by inductive hypotheses:

  $$[\![X', \langle\uparrow\rangle.m, S]\!] = C'^{[\![X', \langle\rangle, S]\!]}[1 \mapsto m'_1@\langle\uparrow\rangle.m \triangleright P'_1, \ldots, n_1 \mapsto m'_{n_1}@\langle\uparrow\rangle.m \triangleright P'_{n_1}]$$
  $$[\![X'', \langle\uparrow\rangle.m, S]\!] = C''^{[\![X'', \langle\rangle, S]\!]}[1 \mapsto m''_1@\langle\uparrow\rangle.m \triangleright P''_1, \ldots, n_1 \mapsto m''_{n_1}@\langle\uparrow\rangle.m \triangleright P''_{n_1}]$$

  The thesis follows since

  $$\begin{aligned}[\![X, m, S]\!] = & C^{[\![X, \langle\rangle, S]\!]}[1 \mapsto m'_1@\langle\uparrow\rangle.m \triangleright P'_1, \ldots, n_1 \mapsto m'_{n_1}@\langle\uparrow\rangle.m \triangleright P'_{n_1}, \\ & n_1 + 1 \mapsto m''_1@\langle\uparrow\rangle.m \triangleright P''_1, \ldots, n_1 + n_2 \mapsto m''_{n_2}@\langle\uparrow\rangle.m \triangleright P''_{n_2}]\end{aligned}$$

  where $C^{[\![X, \langle\rangle, S]\!]} = C'^{[\![X', \langle\rangle, S]\!]} \mid C'''^{[\![X'', \langle\rangle, S]\!]}$ with $C'''^{[\![X'', \langle\rangle, S]\!]}$ equal to $C''^{[\![X'', \langle\rangle, S]\!]}$ but for having hole numbers increased by n_1.
- if $X = \nu a_\Gamma(X')$ then $[\![\nu a_\Gamma X', m, S]\!] = \nu b_\Gamma [\![X'\{^b/_a\}, m, S]\!]$ with $b \notin \mathtt{Fnv}(m) \wedge (b = a \vee b \notin \mathtt{Fnv}(X'))$. By inductive hypothesis we have:

  $$[\![X'\{^b/_a\}, m, S]\!] = C^{[\![X'\{^b/_a\}, \langle\rangle, S]\!]}[1 \mapsto m_1@m \triangleright P_1, \ldots, n \mapsto m_n@m \triangleright P_n]$$

  Hence:

  $$\nu b_\Gamma [\![X'\{^b/_a\}, m, S]\!] = \nu b_\Gamma C^{[\![X'\{^b/_a\}, \langle\rangle, S]\!]}[1 \mapsto m_1@m \triangleright P_1, \ldots, n \mapsto m_n@m \triangleright P_n]$$

  as desired, by selecting $C^{[\![X, \langle\rangle, S]\!]} = \nu b_\Gamma C^{[\![X'\{^b/_a\}, \langle\rangle, S]\!]}$.

The thesis follows by noticing that m is only inserted into monitored processes, and has no impact on the other parts of the term since for every name $a \in \mathtt{Fnv}(m)$, $\nu a_\Gamma \notin R$ for all Γ and for some i', k', c, b, d, event $\langle i', k', c[b/d]\rangle \notin m$. □

Lemma 6. *If there is an $R\pi$ transition $R \xrightarrow{(i,j,k):\alpha} S$ then there exist $R' \equiv R$ and $S' \equiv S$ such that $R' \xrightarrow{(i,j,k):\alpha} S'$.*

Proof. The proof is by induction on the derivation of the transition $R \xrightarrow{(i,j,k):\alpha} S$, with a case analysis on the last applied rule:

- Rules (IN+) and (OUT+): the thesis holds trivially, by choosing $R' = R$, $S' = S$.
- Rule (PAR+): we have that $R = R_1 \mid R_2$ and $S = S_1 \mid R_2$, with premise $R_1 \xrightarrow{(i,j,k):\alpha} S_1$ and $\mathtt{Bnv}(\alpha) \cap \mathtt{Fnv}(R_2) = \emptyset$. By inductive hypothesis there exist $R'_1 \equiv R_1$ and $S'_1 \equiv S_1$ such that $R'_1 \xrightarrow{(i,j,k):\alpha} S'_1$. By congruence we have $R \equiv R'_1 \mid R_2$ and $S \equiv S'_1 \mid R_2$, and, by applying rule (PAR+) with premise $R'_1 \xrightarrow{(i,j,k):\alpha} S'_1$, we obtain $R'_1 \mid R_2 \xrightarrow{(i,j,k):\alpha} S'_1 \mid R_2$ as desired.
- Rule (OPEN+): we have $R = \nu a_\Gamma(R_1)$ and $S = \nu a_{\Gamma+1}(S_1)$ with premise $R_1 \xrightarrow{(i,j,k):\alpha} S_1$ where $\alpha = \overline{b}\langle\nu a_{\Gamma'}\rangle \vee \alpha = \overline{b}a$. By inductive hypothesis there exist $R'_1 \equiv R_1$ and $S'_1 \equiv S_1$ such that $R'_1 \xrightarrow{(i,j,k):\alpha} S'_1$ with $\alpha = \overline{b}\langle\nu a_{\Gamma'}\rangle \vee \alpha = \overline{b}a$. By congruence we have $R \equiv \nu a_\Gamma(R'_1)$ and $S = \nu a_{\Gamma+1}(S_1)$ and, by applying rule (OPEN+) with premise $R'_1 \xrightarrow{(i,j,k):\alpha} S'_1$ with $\alpha = \overline{b}\langle\nu a_{\Gamma'}\rangle \vee \alpha = \overline{b}a$, we obtain

  $$\nu a_\Gamma(R'_1) \xrightarrow{(i,j,k):\overline{b}\langle\nu a_\Gamma\rangle} \nu a_{\Gamma+1}(S_1)$$

  as desired.
- Rule (CLOSE+): similar to the case above.
- Rule (COM+): similar to the case above.
- Rule (CAUSE REF+): similar to the case above.
- Rule (NEW+): similar to the case above.
- Rule (MEM+): we have as premises $R \equiv R_1$, $R_1 \xrightarrow{(i,j,k):\alpha} S_1$, and $S_1 \equiv S$. By inductive hypothesis there exist $R'_1 \equiv R_1$ and $S'_1 \equiv S_1$ such that $R'_1 \xrightarrow{(i,j,k):\alpha} S'_1$. We can conclude by noticing that $R \equiv R_1 \equiv R'_1$ and $S \equiv S_1 \equiv S'_1$, as desired. □

Lemma 15. *If $R \equiv S$ then, for each memory m such that for every name $a \in \mathtt{Fnv}(m)$, $\nu a_\Gamma \notin R, S$ for all Γ and for some i', k', c, b, d and event $\langle i', k', c[b/d]\rangle \notin m$, we have $R@m \equiv S@m$.*

Proof. By induction on the derivation of $R \equiv S$. The only interesting case is the base one, corresponding to the application of an axiom. We have a case analysis on the applied axiom.

(**SPLIT**): we have $R = m' \triangleright (P \mid Q)$ and $S = \langle\uparrow\rangle.m' \triangleright P \mid \langle\uparrow\rangle.m' \triangleright Q$. By adding the same memory m to the processes R and S we get $R@m = m'@m \triangleright (P \mid Q)$ and $S@m = \langle\uparrow\rangle.m'@m \triangleright P \mid \langle\uparrow\rangle.m'@m \triangleright Q$. By applying the (SPLIT) axiom to the process $R@m$ we get $m'@m \triangleright (P \mid Q) \equiv \langle\uparrow\rangle.m'@m \triangleright P \mid \langle\uparrow\rangle.m'@m \triangleright Q$ as desired.

(**RES**): we have $R = m' \triangleright \nu a(P)$ and $S = \nu a_\emptyset(m' \triangleright P)$ with $a \notin \mathtt{Fnv}(m')$. By adding the same memory m to the processes R and S we get $R@m = m'@m \triangleright \nu a(P)$ and $S@m = \nu a_\emptyset(m'@m \triangleright P)$. By applying the axiom (RES) to the process $R@m$ we get $m'@m \triangleright \nu a(P) \equiv \nu a_\emptyset(m'@m \triangleright P)$ as desired. Note that $a \notin \mathtt{Fnv}(m')$ from the side condition of the inductive hypothesis and $a \notin \mathtt{Fnv}(m)$ from the statement of the lemma.

(α): we have $R \equiv S$ since $R =_\alpha S$. By adding the same memory m to both R and S we still have $R@m =_\alpha S@m$ (note that since for every name $a \in \mathtt{Fnv}(m)$, $\nu a_\Gamma \notin R, S$ for all Γ then m is not changed by α-conversion), which implies $R@m \equiv S@m$, as desired. □

Lemma 16. *If we have a $R\pi$ forward transition $R \xrightarrow{(i,j,k):\alpha} R'$ then, for each memory m such that for every name $a \in \mathtt{Fnv}(m)$, $\nu a_\Gamma \notin R$ for all Γ and for some i', k', c, b, d and event $\langle i', k', c[b/d]\rangle \notin m$, we have $R@m \xrightarrow{(i,j,k):\alpha} R'@m$.*

Proof. The proof is by induction on the derivation of the transition $R \xrightarrow{(i,j,k):\alpha} R'$, with a case analysis on the last applied rule:

- if $R\pi$ rule (OUT+) is applied, then we have

  $$m_1 \triangleright \overline{b}a.P \xrightarrow{(i,j,*):m_1[\overline{b}a]} \langle i, k, \overline{b}a\rangle.m_1 \triangleright P$$

  By applying the rule (OUT+) on $R@m = m_1@m \triangleright \overline{b}a.P$, we have:

  $$m_1@m \triangleright \overline{b}a.P \xrightarrow{(i,j',*):m_1@m[\overline{b}a]} \langle i, k, \overline{b}a\rangle.m_1@m \triangleright P$$

  Since in m there is no event for full synchronisation (there is no substitution), we have $m_1[\overline{b}a] = m_1@m[\overline{b}a]$ and $j = j'$ as desired.
- if $R\pi$ rule (IN+) is applied, it is similar to the case above.
- if $R\pi$ rule (OPEN+) is applied, then we have

  $$\nu a_\Gamma(R_1) \xrightarrow{(i,j,k):\overline{b}\langle\nu a_\Gamma\rangle} \nu a_{\Gamma+i}(R'_1)$$

  with premise $R_1 \xrightarrow{(i,j,k):\alpha} R'_1$ where $\alpha = \overline{b}\langle\nu a_{\Gamma'}\rangle \vee \alpha = \overline{b}a$. Since $\nu a_\Gamma \in R = \nu a_\Gamma(R_1)$, then $a \notin \mathtt{Fnv}(m)$. Then by inductive hypothesis we have that $R_1@m \xrightarrow{(i,j,k):\alpha} R'_1@m$ where $\alpha = \overline{b}\langle\nu a_{\Gamma'}\rangle \vee \alpha = \overline{b}a$. Now we can apply the $R\pi$ rule (OPEN+) on $(\nu a_\Gamma(R_1))@m$ and we have:

  $$(\nu a_\Gamma(R_1))@m = \nu a_\Gamma(R_1@m) \xrightarrow{(i,j,k):\overline{b}\langle\nu a_\Gamma\rangle} \nu a_{\Gamma+i}(R'_1@m)$$

  The thesis follows since $(\nu a_{\Gamma+i}(R'_1))@m = \nu a_{\Gamma+i}(R'_1@m)$.
- if $R\pi$ rule (CAUSE REF+) is applied, it is similar to the case above.
- if $R\pi$ rule (NEW+) is applied, it is similar to the case above.
- if $R\pi$ rule (PAR+) is applied, then we have

  $$R_1 \mid R_2 \xrightarrow{(i,j,k):\alpha} R'_1 \mid R_2$$

  with a premise $R_1 \xrightarrow{(i,j,k):\alpha} R'_1$. By inductive hypothesis we have that $R_1@m \xrightarrow{(i,j,k):\alpha} R'_1@m$. Now we can apply the $R\pi$ rule (PAR+) on $(R_1 \mid R_2)@m = R_1@m \mid R_2@m$ and we have:

  $$R_1@m \mid R_2@m \xrightarrow{(i,j,k):\alpha} R'_1@m \mid R_2@m$$
- if $R\pi$ rule (COM+) is applied, it is similar to the case above.
- if $R\pi$ rule (CLOSE+) is applied, it is similar to the case above.
- if $R\pi$ rule (MEM+) is applied, then we have

  $$R_1 \xrightarrow{(i,j,k):\alpha} R'_1$$

  with a premise $R_1 \equiv S_1 \xrightarrow{(i,j,k):\alpha} S'_1 \equiv R'_1$. By inductive hypothesis and Lemma 15 we have that $R_1@m \equiv S_1@m \xrightarrow{(i,j,k):\alpha} S'_1@m \equiv R_1@m$. Now we can apply the $R\pi$ rule (MEM+) and we have:

  $$R_1@m \xrightarrow{(i,j,k):\alpha} R'_1@m$$

  as desired. □

Lemma 17. *There is a forward transition* $R \xrightarrow{(i,j,k):\alpha}_{\triangleright} R'$ *iff there is* $R@m \xrightarrow{(i,j,k):\alpha}_{\triangleright} R'@m$ *such that for every name* $a \in \mathtt{Fnv}(m)$, $\nu a_\Gamma \notin R$ *for all* Γ *and for some* i', k', c, b, d, *event* $\langle i', k', c[b/d]\rangle \notin m$.

Proof. The proof is similar to that of Lemma 16. □

We can now prove the operational correspondence for forward transitions.

Proposition 1 *(Forward correctness). Let* $S :: X$ *be a reachable process from the framework and* $R = [\![X, \langle\rangle, S]\!]$. *For each transition in the framework* $S :: X \xrightarrow{(i,K,j):\alpha}\!\!\!\!\rightarrow S' :: X'$ *there exists a corresponding* $R\pi$ *transition* $R \xrightarrow{(i,j,k):\alpha}_{\triangleright} R'$ *with* $[\![X', \langle\rangle, S']\!] = R'$ *and* $K = \{k\}$.

Proof. By induction on the derivation of $S :: X \xrightarrow{(i,K,j):\alpha}\!\!\!\!\rightarrow S' :: X'$ and by case analysis on the last applied rule.

(**Out1**): We have $\emptyset : \overline{b}^* a^*.\mathbf{P} \xrightarrow{(i,K,*):\overline{b}a}\!\!\!\!\rightarrow \emptyset :: \overline{b}^* a^*[i, K].\mathbf{P}$ where $K = \{*\}$. By applying the encoding we have:

$$[\![\overline{b}^* a^*.\mathbf{P}, \langle\rangle, \emptyset]\!] = \langle\rangle \triangleright \sigma(\emptyset, \overline{b}^* a^*.\mathbf{P}) = \langle\rangle \triangleright \overline{b}a.P$$

Then, by using $R\pi$ rule (OUT+) we get

$$\langle\rangle \triangleright \overline{b}a.P \xrightarrow{(i,j,*):m[\overline{b}a]} \langle i, *, \overline{b}a\rangle.\langle\rangle \triangleright P$$

with $m[\overline{b}a] = \overline{b}a$ and $j = *$ (since it is the very first action). The thesis follows since $[\![\overline{b}^* a^*[i, \{*\}].\mathbf{P}, \langle\rangle, \emptyset]\!] = \langle i, *, \overline{b}a\rangle.\langle\rangle \triangleright P$.

(**Out2**): We have $S :: \overline{c}^* d^*[i', \{*\}].X_1 \xrightarrow{(i,K,j):\overline{b}a}\!\!\!\!\rightarrow S' :: \overline{c}^* d^*[i', \{*\}].X_1'$ with premise $S :: X_1 \xrightarrow{(i,K,j):\overline{b}a}\!\!\!\!\rightarrow S' :: X_1'$.

Let $R_1 = [\![X_1, \langle\rangle, S]\!]$. By applying the inductive hypothesis we have that $R_1 \xrightarrow{(i,j,k):\overline{b}a} R_1'$, with $R_1' = [\![X_1', \langle\rangle, S']\!]$, where $K = \{k\}$ and $S = S'$ since the executed action is not a synchronisation. From the encoding we have

$$[\![\overline{c}^* d^*[i', \{*\}].X_1, \langle\rangle, S]\!] = [\![X_1, \langle i', *, \sigma(S, \overline{c}^* d^*)\rangle.\langle\rangle, S]\!]$$

where by Assumption 1 from $c, d \in \mathtt{free}([\![\overline{c}^* d^*[i', \{*\}].X_1, \langle\rangle, S]\!])$ we have $\nu c_\Gamma, \nu d_\Gamma \notin X_1$. Since in the Framework, $\overline{c}^* d^*$ is the very first action that process did, we have $\sigma(S, \overline{c}^* d^*) = \overline{c}d$.
Thanks to Lemma 14 we have that $[\![X_1, \langle i', *, \overline{c}d\rangle.\langle\rangle, S]\!] = R_1@\langle i', *, \overline{c}d\rangle.\langle\rangle$. By Lemma 16 we have:

$$R_1@\langle i', *, \overline{c}d\rangle.\langle\rangle \xrightarrow{(i,j,k):\overline{b}a} R_1'@\langle i', *, \overline{c}d\rangle.\langle\rangle$$

The thesis follows since

$$[\![\overline{c}^* d^*[i', \{*\}].X_1', \langle\rangle]\!], S = [\![X_1', \langle i', *, \overline{c}d\rangle.\langle\rangle, S]\!] = R_1'@\langle i', *, \overline{c}d\rangle.\langle\rangle$$

(**In1**): Similar to the case of (Out1).
(**In2**): Similar to the case of (Out2).
(**Open**): We have

$$S :: \nu a_\Gamma(X_1) \xrightarrow{(i,K,j):\overline{b}\langle\nu a_\Gamma\rangle}\!\!\!\!\rightarrow S' :: \nu a_{\Gamma+1}(X_1')$$

with premise $S :: X_1 \xrightarrow{(i,K,j):\alpha}\!\!\!\!\rightarrow S' :: X_1'$, where $\alpha = \overline{b}a \vee \alpha = \overline{b}\langle\nu a_{\Gamma'}\rangle$ and $S' = S$ since the performed action is not a synchronisation. Let us choose, for example, $\alpha = \overline{b}\langle\nu a_{\Gamma'}\rangle$; the other case is similar.

Let $R_1 = [\![X_1, \langle\rangle, S]\!]$. By applying the inductive hypothesis we have that $R_1 \xrightarrow{(i,j,k):\overline{b}\langle\nu a_{\Gamma'}\rangle} R_1'$, where $K = \{k\}$ and $R_1' = [\![X_1', \langle\rangle, S']\!]$.
Since $m = \langle\rangle$, by the encoding for $b = a$, we have $[\![\nu a_\Gamma(X_1), \langle\rangle, S]\!] = \nu a_\Gamma[\![X_1, \langle\rangle, S]\!] = \nu a_\Gamma(R_1)$. By applying the $R\pi$ rule (OPEN+) on $\nu a_\Gamma(R_1)$, we have

$$\nu a_\Gamma(R_1) \xrightarrow{(i,j,k):\overline{b}\langle\nu a_\Gamma\rangle} \nu a_{\Gamma+1}(R_1')$$

The thesis follows since $[\![\nu a_{\Gamma+1}(X_1'), \langle\rangle, S']\!] = \nu a_{\Gamma+1}(R_1')$.
(**Cause Ref**): We have

$$S :: \nu a_\Gamma(X_1) \xrightarrow{(i,K',j):\alpha}\!\!\!\!\rightarrow S' :: \nu a_\Gamma(X'_{1[K'/K]@i})$$

with premise $S :: X_1 \xrightarrow{(i,K,j):\alpha} S' :: X_1'$ where $a \in \mathtt{subj}(\alpha)$, $\mathtt{empty}(\Gamma) \neq \mathtt{true}$ and $K' = K \vee K \rightsquigarrow_{X_1} K'$. As the performed action is not a synchronisation, we have $S' = S$.

Let $R_1 = [\![X_1, \langle\rangle, S]\!]$. By applying the inductive hypothesis we have that $R_1 \xrightarrow{(i,j,k):\alpha} R_1'$, with $a \in \mathtt{subj}(\alpha)$, $k = K$, $\mathtt{empty}(\Gamma) \neq \mathtt{true}$ and $k' = k \vee k \rightsquigarrow_{R_1} k'$ and $R_1' = [\![X_1', \langle\rangle, S']\!]$.
Since $m = \langle\rangle$, in the encoding we can take $b = a$ and we have $[\![\nu a_\Gamma(X_1), \langle\rangle, S]\!] = \nu a_\Gamma [\![X_1, \langle\rangle, S]\!] = \nu a_\Gamma(R_1)$. By applying the $R\pi$ rule (CAUSE REF+) on $\nu a_\Gamma(R_1)$, we have

$$\nu a_\Gamma(R_1) \xrightarrow{(i,j,k'):\alpha} \nu a_\Gamma(R'_{1[k'/k]@i})$$

Since $R_1 = [\![X_1, \langle\rangle, S]\!]$, the restriction $\nu a_{\Gamma''} \in R_1$ iff $\nu a_{\Gamma''} \in X_1$; hence $k' = K'$. The thesis follows since

$$[\![\nu a_\Gamma(X'_{1[K'/K]@i}), \langle\rangle, S']\!] = \nu a_\Gamma [\![X'_{1[K'/K]@i}, \langle\rangle, S']\!] = \nu a_\Gamma(R'_{1[k'/k]@i})$$

(**Par**): We have

$$S :: Y_1 \mid Y_2 \xrightarrow{(i,K,j):\alpha} S' :: Y_1' \mid Y_2$$

with premise $S :: Y_1 \xrightarrow{(i,K,j):\alpha} S' :: Y_1'$ where $i \notin Y_2$ and $\mathtt{Bnv}(\alpha) \cup \mathtt{Fnv}(Y_2) = \emptyset$ and $S' = S$.

Let $R_1 = [\![Y_1, \langle\rangle, S]\!]$. By applying the inductive hypothesis we have that $R_1 \xrightarrow{(i,j,k):\alpha} R_1'$ with $R_1' = [\![Y_1', \langle\rangle, S']\!]$ and $\{k\} = K$.
From the encoding we have $[\![Y_1 \mid Y_2, \langle\rangle, S]\!] = [\![Y_1, \langle\uparrow\rangle.\langle\rangle, S]\!] \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S]\!]$. Thanks to Lemma 14, $[\![Y_1, \langle\uparrow\rangle.\langle\rangle, S]\!] = R_1@\langle\uparrow\rangle.\langle\rangle$. Since $m = \langle\uparrow\rangle$ we can apply Lemma 16 and obtain $R_1@\langle\uparrow\rangle.\langle\rangle \xrightarrow{(i,j,k):\alpha} R_1'@\langle\uparrow\rangle.\langle\rangle$. By applying $R\pi$ rule (PAR+) on $R_1@\langle\uparrow\rangle.\langle\rangle \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S]\!]$ we have:

$$R_1@\langle\uparrow\rangle.\langle\rangle \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S]\!] \xrightarrow{(i,j,k):\alpha} R_1'@\langle\uparrow\rangle.\langle\rangle \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S]\!]$$

The thesis follows since

$$[\![Y_1' \mid Y_2, \langle\rangle, S']\!] = [\![Y_1', \langle\uparrow\rangle.\langle\rangle, S']\!] \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S']\!]$$

and by Lemma 14 we have

$$[\![Y_1', \langle\uparrow\rangle.\langle\rangle, S']\!] \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S']\!] = R_1'@\langle\uparrow\rangle.\langle\rangle \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S']\!]$$

(**Com**): We have

$$S :: Y_1 \mid Y_2 \xrightarrow{(i,\{*\},*):\tau} S \cup \{^{a^i}/_x\} :: Y_1' \mid Y_2'\{^{a^i}/_x\}$$

with premises $S :: Y_1 \xrightarrow{(i,K_1,j_1):\overline{b}a} S :: Y_1'$ and $S :: Y_2 \xrightarrow{(i,K_2,j_2):b(x)} S :: Y_2'$ when $K_1 \bowtie j_2 \wedge K_2 \bowtie j_1$.

Let $R_1 = [\![Y_1, \langle\rangle, S]\!]$ and $R_2 = [\![Y_2, \langle\rangle, S]\!]$. By applying the inductive hypothesis we have that $R_1 \xrightarrow{(i,j_1,k_1):\overline{b}a} R_1'$ and $R_2 \xrightarrow{(i,j_2,k_2):b(x)} R_2'$ with $\{k_1\} = K_1$ and $\{k_2\} = K_2$.
By the encoding we have $[\![Y_1 \mid Y_2, \langle\rangle, S]\!] = [\![Y_1, \langle\uparrow\rangle.\langle\rangle, S]\!] \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S]\!]$.
By Lemma 14 we have

$$[\![Y_1, \langle\uparrow\rangle.\langle\rangle, S]\!] = R_1@\langle\uparrow\rangle.\langle\rangle \quad \text{and} \quad [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S]\!] = R_2@\langle\uparrow\rangle.\langle\rangle$$

We can apply Lemma 16 since $m = \langle\uparrow\rangle$ and we get

$$R_1@\langle\uparrow\rangle.\langle\rangle \xrightarrow{(i,j_1,k_1):\overline{b}a} R_1'@\langle\uparrow\rangle.\langle\rangle \text{ and } R_2@\langle\uparrow\rangle.\langle\rangle \xrightarrow{(i,j_2,k_2):b(x)} R_2'@\langle\uparrow\rangle.\langle\rangle$$

By applying $R\pi$ rule (Com+) on $R_1@\langle\uparrow\rangle.\langle\rangle \mid R_2@\langle\uparrow\rangle.\langle\rangle$ we have:

$$R_1@\langle\uparrow\rangle.\langle\rangle \mid R_2@\langle\uparrow\rangle.\langle\rangle \xrightarrow{(i,\{*\},*):\tau} R_1'@\langle\uparrow\rangle.\langle\rangle \mid R'_{2[a/x]@i}@\langle\uparrow\rangle.\langle\rangle$$

The thesis follows since

$$[\![Y_1' \mid Y_2'\{^{a^i}/_x\}, \langle\rangle, S']\!] = [\![Y_1', \langle\uparrow\rangle.\langle\rangle, S']\!] \mid [\![Y_2'\{^{a}/_x\}, \langle\uparrow\rangle.\langle\rangle, S']\!]$$

where $S' = S \cup \{^{a^i}/_x\}$; and by Lemma 14 we have

$$[\![Y_1', \langle\uparrow\rangle.\langle\rangle, S']\!] \mid [\![Y_2'\{^{a^i}/_x\}, \langle\uparrow\rangle.\langle\rangle, S']\!] = R_1'@\langle\uparrow\rangle.\langle\rangle \mid R'_{2[a/x]@i}@\langle\uparrow\rangle.\langle\rangle$$

(**Close**): We have

$$S :: Y_1 \mid Y_2 \xrightarrow{(i,\{*\},*):\tau} S' :: \nu a_\Gamma(Y_1' \mid Y_2'\{^{a^i}/_x\})$$

with premises $S :: Y_1 \xrightarrow{(i,K_1,j_1):\overline{b}\langle \nu a_\Gamma\rangle} S :: Y_1'$ and $S :: Y_2 \xrightarrow{(i,K_2,j_2):b(x)} S :: Y_2'$ when $K_1 \bowtie j_2 \wedge K_2 \bowtie j_1$ and $S' = S \cup \{^{a^i}/_x\}$.

Let $R_1 = [\![Y_1, \langle\rangle, S]\!]$ and $R_2 = [\![Y_2, \langle\rangle, S]\!]$. By applying the inductive hypothesis we have that $R_1 \xrightarrow{(i,j_1,k_1):\overline{b}\langle \nu a_\Gamma\rangle} R_1'$ and $R_2 \xrightarrow{(i,j_2,k_2):b(x)} R_2'$ with $\{k_1\} = K_1$ and $\{k_2\} = K_2$.
By the encoding we have $[\![Y_1 \mid Y_2, \langle\rangle, S]\!] = [\![Y_1, \langle\uparrow\rangle.\langle\rangle, S]\!] \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S]\!]$.
Thanks to Lemma 14, $[\![Y_1, \langle\uparrow\rangle.\langle\rangle, S]\!] = R_1@\langle\uparrow\rangle.\langle\rangle$ and $[\![Y_2, \langle\uparrow\rangle.\langle\rangle, S]\!] = R_2@\langle\uparrow\rangle.\langle\rangle$. Since $m = \langle\uparrow\rangle$ by Lemma 16 we have

$$R_1@\langle\uparrow\rangle.\langle\rangle \xrightarrow{(i,j_1,k_1):\overline{b}\langle \nu a_\Gamma\rangle} R_1'@\langle\uparrow\rangle.\langle\rangle \text{ and } R_2@\langle\uparrow\rangle.\langle\rangle \xrightarrow{(i,j_2,k_2):b(x)} R_2'@\langle\uparrow\rangle.\langle\rangle$$

By applying $R\pi$ rule (CLOSE+) on $R_1@\langle\uparrow\rangle.\langle\rangle \mid R_2@\langle\uparrow\rangle.\langle\rangle$ we have:

$$R_1@\langle\uparrow\rangle.\langle\rangle \mid R_2@\langle\uparrow\rangle.\langle\rangle \xrightarrow{(i,\{*\},*):\tau} \nu a_\Gamma(R_1'@\langle\uparrow\rangle.\langle\rangle \mid R_{2[a/x]@i}'@\langle\uparrow\rangle.\langle\rangle)$$

The thesis follows since

$$\begin{aligned}[\![\nu a_\Gamma(Y_1' \mid Y_2'\{^{a^i}/_x\}), \langle\rangle, S']\!] &= \nu a_\Gamma [\![Y_1' \mid Y_2'\{^{a^i}/_x\}, \langle\rangle, S']\!] \\ &= \nu a_\Gamma [\![Y_1', \langle\uparrow\rangle.\langle\rangle, S']\!] \mid [\![Y_2'\{^{a^i}/_x\}, \langle\uparrow\rangle.\langle\rangle, S']\!]\end{aligned}$$

and by Lemma 14 we have

$$[\![Y_1', \langle\uparrow\rangle.\langle\rangle, S']\!] \mid [\![Y_2'\{^{a^i}/_x\}, \langle\uparrow\rangle.\langle\rangle, S']\!] = R_1'@\langle\uparrow\rangle.\langle\rangle \mid R_{2[a/x]@i}'@\langle\uparrow\rangle.\langle\rangle$$

(**Res**): We have

$$S :: \nu a_\Gamma(X_1) \xrightarrow{(i,K,j):\alpha} S' :: \nu a_\Gamma(X_1')$$

with premise $S :: X_1 \xrightarrow{(i,K,j):\alpha} S' :: X_1'$ where $a \notin \alpha$.

Let $R_1 = [\![X_1, \langle\rangle, S]\!]$. By applying the inductive hypothesis we have that $R_1 \xrightarrow{(i,j,k):\alpha} R_1'$, with $a \notin \alpha$, $\{k\} = K$ and $R_1' = [\![X_1', \langle\rangle, S']\!]$ where $S' = S$.
Since $m = \langle\rangle$ we have $b = a$ and by the encoding we get $[\![\nu a_\Gamma(X_1), \langle\rangle, S]\!] = \nu a_\Gamma [\![X_1, \langle\rangle, S]\!] = \nu a_\Gamma(R_1)$. By applying the $R\pi$ rule (NEW+) on $\nu a_\Gamma(R_1)$, we have

$$\nu a_\Gamma(R_1) \xrightarrow{(i,j,k):\alpha} \nu a_\Gamma(R_1')$$

The thesis follows since

$$[\![\nu a_\Gamma(X_1'), \langle\rangle, S']\!] = \nu a_\Gamma [\![X_1', \langle\rangle, S']\!] = \nu a_\Gamma(R_1') \quad \square$$

Proposition 3 *(Forward completeness). Let X be a reachable process from the framework and $R = [\![X, \langle\rangle, S]\!]$. For each $R\pi$ transition $R \xrightarrow{(i,j,k):\alpha} R'$ there exists a corresponding transition in the framework $S :: X \xrightarrow{(i,K,j):\alpha} S' :: X'$ with $R' \equiv [\![X', \langle\rangle, S']\!]$ and $K = \{k\}$.*

Proof. Thanks to Lemma 6 we can equivalently write the statement as follows: for each reachable process $S :: X$ in the framework and $R\pi$ processes R and R'', such that $R = [\![X, \langle\rangle, S]\!]$ and $R'' \equiv R$, if there exists a $R\pi$ transition $R'' \xrightarrow{(i,j,k):\alpha} R'$, then there exists a corresponding transition in the framework $S :: X \xrightarrow{(i,K,j):\alpha} S' :: X'$, with $R' \equiv [\![X', \langle\rangle, S']\!]$. When considering $R'' \equiv R$ we will not consider α-conversion, since it can be trivially matched by framework α-conversion.

Now the proof is by structural induction on $S :: X$ with a case analysis on the last applied rule in the derivation of $R'' \xrightarrow{(i,j,k):\alpha} R'$. We have two main cases, depending on whether X is a process without past **P** or not.

In the first case $X = \mathbf{P}$, and we perform a structural induction on **P**.

$\mathbf{P} = \overline{b}^*a^*.\mathbf{P}_1$: we have that $R = [\![\overline{b}^*a^*.\mathbf{P}_1, \langle\rangle, S]\!]$ where $S = \emptyset$ and by applying the encoding, we have

$$[\![\overline{b}^*a^*.\mathbf{P}_1, \langle\rangle, \emptyset]\!] = \langle\rangle \triangleright \sigma(\emptyset, \overline{b}^*a^*.\mathbf{P}_1) = \langle\rangle \triangleright \overline{b}a.P_1$$

Since we cannot apply any structural rule (beyond α-conversion), then $R'' = R$. Then, the only applicable rule in $R\pi$ is (OUT+), and we get

$$\langle\rangle \triangleright \overline{b}a.P_1 \xrightarrow{(i,j,*):\langle\rangle[\overline{b}a]} \langle i,*,\overline{b}a\rangle.\langle\rangle \triangleright P_1 = R'$$

Since $m = \langle\rangle$, we have $j = *$ and $\langle\rangle[\overline{b}a] = \overline{b}a$.
In the framework, process $\emptyset :: \overline{b}^*a^*.\mathbf{P}_1$ can perform the same action $(i,\{*\},*) : \overline{b}a$ by applying the rule (Out1) and we get

$$\emptyset :: \overline{b}^*a^*.\mathbf{P}_1 \xrightarrow{(i,\{*\},*):\overline{b}a} \emptyset :: \overline{b}^*a^*[i,\{*\}].\mathbf{P}_1$$

Since $[\![\overline{b}^*a^*[i,\{*\}].\mathbf{P}_1, \langle\rangle, \emptyset]\!] = [\![\mathbf{P}_1, \langle i,*,\sigma(\emptyset,\overline{b}^*a^*)\rangle.\langle\rangle, \emptyset]\!] = \langle i,*,\overline{b}a\rangle.\langle\rangle \triangleright P_1 = R'$ and $S = S' = \emptyset$ (since the executed action was not a synchronisation) we are done.

$\mathbf{P} = b^*(x).\mathbf{P}_1$: similar to the case above.

$\mathbf{P} = \mathbf{P}_1 \mid \mathbf{P}_2$: we have that $R = [\![\mathbf{P}_1 \mid \mathbf{P}_2, \langle\rangle, S]\!]$. By applying the encoding and the fact that $S = \emptyset$, we have

$$[\![\mathbf{P}_1 \mid \mathbf{P}_2, \langle\rangle, \emptyset]\!] = \langle\rangle \triangleright (\sigma(\emptyset, \mathbf{P}_1 \mid \mathbf{P}_2)) = \langle\rangle \triangleright (P_1 \mid P_2)$$

We have now two possibilities: either $R'' = R$ or $R'' = \langle\uparrow\rangle.\langle\rangle \triangleright P_1 \mid \langle\uparrow\rangle.\langle\rangle \triangleright P_2$. In the first case no rule can be applied and we are done. Let us consider the second one. Now we distinguish three cases depending on whether the last applied rule is (PAR+), (Com+) or (Close+). If (PAR+) is applied, then we have that

$$\langle\uparrow\rangle.\langle\rangle \triangleright P_1 \mid \langle\uparrow\rangle.\langle\rangle \triangleright P_2 \xrightarrow{(i,*,*):\alpha} R_1 \mid \langle\uparrow\rangle.\langle\rangle \triangleright P_2$$

with a premise $\langle\uparrow\rangle.\langle\rangle \triangleright P_1 \xrightarrow{(i,*,*):\alpha} R_1$ and $\mathtt{Bnv}(\alpha) \cup \mathtt{Fnv}(\langle\uparrow\rangle.\langle\rangle \triangleright P_2) = \emptyset$. Since α is the very first action, we have $j,k = *$.

Thanks to Lemma 17 we can derive $\langle\rangle \triangleright P_1 \xrightarrow{(i,*,*):\alpha} R_1'$ with $R_1 = R_1'@\langle\uparrow\rangle.\langle\rangle$.

Since $\langle\rangle \triangleright P_1 = [\![\mathbf{P}_1, \langle\rangle, \emptyset]\!]$ we can apply the inductive hypothesis and get $\emptyset :: \mathbf{P}_1 \xrightarrow{(i,\{*\},*):\alpha} \emptyset :: X_1'$ with $[\![X_1', \langle\rangle, \emptyset]\!] = R_1'$. The set of substitutions remains empty since $\alpha \neq \tau$. We can now apply the rule from the framework (Par) and derive

$$\emptyset :: \mathbf{P}_1 \mid \mathbf{P}_2 \xrightarrow{(i,\{*\},*):\alpha} \emptyset :: X_1' \mid \mathbf{P}_2$$

We can notice that $[\![X_1' \mid \mathbf{P}_2, \langle\rangle, \emptyset]\!] = [\![X_1', \langle\uparrow\rangle.\langle\rangle, \emptyset]\!] \mid [\![\mathbf{P}_2, \langle\uparrow\rangle.\langle\rangle, \emptyset]\!]$. By applying Lemma 14 on $[\![X_1', \langle\rangle, \emptyset]\!] = R_1'$ we can derive $[\![X_1', \langle\uparrow\rangle.\langle\rangle, \emptyset]\!] = R_1'@\langle\uparrow\rangle.\langle\rangle = R_1$. We can now conclude since

$$[\![X_1', \langle\uparrow\rangle \cdot \langle\rangle, \emptyset]\!] \mid [\![\mathbf{P}_2, \langle\uparrow\rangle \cdot \langle\rangle, \emptyset]\!] = R_1 \mid \langle\uparrow\rangle \cdot \langle\rangle \triangleright P_2$$

as desired.
If (Com+) or (Close+) are applied, we use the inductive hypothesis twice and we reason as in the previous cases.

$\mathbf{P} = \nu a_\Gamma(\mathbf{P}_1)$: since it is an initial process, we have $\Gamma = \emptyset$. Then we have that $R = [\![\nu a_\emptyset(\mathbf{P}_1), \langle\rangle, \emptyset]\!]$ and by applying the encoding, we get

$$[\![\nu a_\emptyset(\mathbf{P}_1), \langle\rangle, \emptyset]\!] = \langle\rangle \triangleright \sigma(\emptyset, \nu a_\emptyset(\mathbf{P}_1)) = \langle\rangle \triangleright \nu a_\emptyset(P_1)$$

We have two possibilities: either $R'' = R$ or $R'' = \nu a_\emptyset(\langle\rangle \triangleright P_1)$. In the first case no rule can be applied and we are done. Let us consider the second one. Depending on the action α, rules (NEW+) and (OPEN+) can be applied.

- If $a \notin \alpha$, the rule (NEW+) is applied and we have

  $$\nu a_\emptyset(\langle\rangle \triangleright P_1) \xrightarrow{(i,*,*):\alpha} \nu a_\emptyset(R_1')$$

  with a premise $\langle\rangle \triangleright P_1 \xrightarrow{(i,*,*):\alpha} R_1'$ where $j,k = *$. Since $\langle\rangle \triangleright P_1 = [\![\mathbf{P}_1, \langle\rangle, \emptyset]\!]$ we can apply the inductive hypothesis and get that $\emptyset :: \mathbf{P}_1 \xrightarrow{(i,\{*\},*):\alpha} \emptyset :: X_1'$ with $[\![X_1', \langle\rangle, \emptyset]\!] = R_1'$. The thesis follows by applying the framework rule (Res) on $\emptyset :: \nu a_\emptyset(\mathbf{P}_1)$.
- If $a \in \mathtt{obj}(\alpha)$, then the only possibility to execute the action α is if $\alpha = \overline{b}\langle\nu a_\emptyset\rangle$ for some name b. Then the rule (OPEN+) is applied and we have

  $$\nu a_\emptyset(\langle\rangle \triangleright P_1) \xrightarrow{(i,*,*):\overline{b}\langle\nu a_\emptyset\rangle} \nu a_{\emptyset+i}(R_1')$$

  with a premise $\langle\rangle \triangleright P_1 \xrightarrow{(i,*,*):\overline{b}a} R_1'$. Since $\langle\rangle \triangleright P_1 = [\![\mathbf{P}_1, \langle\rangle, \emptyset]\!]$ we can apply the inductive hypothesis and get that $\emptyset :: \mathbf{P}_1 \xrightarrow{(i,\{*\},*):\overline{b}a} \emptyset :: X_1'$ with $[\![X_1', \langle\rangle, \emptyset]\!] = R_1'$. The thesis follows by applying the framework rule (Open) on $\emptyset :: \nu a_\emptyset(\mathbf{P}_1)$.

In the second case process X is not standard. We proceed by structural induction on X.

$X = \overline{b}^* a^*[i,\{*\}].Y$: we have that $R = [\![\overline{b}^* a^*[i,*].Y, \langle\rangle, S]\!]$, and by applying the encoding and the fact that the past prefix $\overline{b}^* a^*[i,\{*\}]$ represents the very first action the process X performed, we have

$$[\![\overline{b}^* a^*[i,\{*\}].Y, \langle\rangle, S]\!] = [\![Y, \langle i,*,\sigma(S,\overline{b}^* a^*)\rangle.\langle\rangle, S]\!] = [\![Y, \langle i,*,\overline{b}a\rangle.\langle\rangle, S]\!]$$

Take any $R\pi$ transition

$$[\![Y, \langle i,*,\overline{b}a\rangle.\langle\rangle, S]\!] \equiv R'' \xrightarrow{(i',j',k'):\alpha} R'$$

By Lemma 14 there exists $T'' \equiv [\![Y, \langle\rangle, S]\!]$ such that $R'' = T''@\langle i,*,\overline{b}a\rangle.\langle\rangle$.
By Assumption 1, we know that names $a, b \notin \mathtt{Bnv}(Y)$ hence $a, b \notin \mathtt{Bnv}(T'')$ and from X reachable, we can conclude that $\nu a_\Gamma, \nu b_{\Gamma'} \notin Y$ for some $\Gamma, \Gamma' \neq \emptyset$.

Now by Lemma 17 we have that $T'' \xrightarrow{(i',j',k'):\alpha} T'$ with $R' = T'@\langle i,*,\overline{b}a\rangle.\langle\rangle$. By applying the inductive hypothesis on $[\![Y, \langle\rangle, S]\!] \equiv T'' \xrightarrow{(i',j',k'):\alpha} T'$ we have that $S :: Y \xrightarrow{(i',K',j'):\alpha\!\!\rightarrow} S' :: Y'$ with $[\![Y', \langle\rangle, S']\!] \equiv T'$ and $K' = \{k'\}$. Now we have two options, depending whether action α is a synchronisation or not.
If $\alpha \neq \tau$ By applying the rules from framework, depending on the nature of the action α, we can also derive $S :: \overline{b}^* a^*[i,\{*\}].Y \xrightarrow{(i',K',j'):\alpha\!\!\rightarrow} S' :: \overline{b}^* a^*[i,\{*\}].Y'$ with $S = S'$. The thesis follows since

$$[\![\overline{b}^* a^*[i,\{*\}].Y', \langle\rangle, S']\!] = [\![Y', \langle i,*,\overline{b}a\rangle.\langle\rangle, S']\!] \equiv R'$$

thanks to Lemma 14.
If $\alpha = \tau$, the sets S and S' will differ exactly for the substitution pair obtained by action α. The case is similar to the one above.

$X = b^*(x)[i,\{*\}].Y$: similar to the case above. Since X is reachable, we know that past prefix $b^*(x)[i,\{*\}]$ was not part of the synchronisation.

$X = Y_1 \mid Y_2$: we have that $R = [\![Y_1 \mid Y_2, \langle\rangle, S]\!]$, and by applying the encoding we obtain

$$[\![Y_1 \mid Y_2, \langle\rangle, S]\!] = [\![Y_1, \langle\uparrow\rangle.\langle\rangle, S]\!] \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S]\!]$$

Take any term $R'' \equiv [\![Y_1, \langle\uparrow\rangle.\langle\rangle, S]\!] \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S]\!]$. There are two cases: either $R'' = R''_1 \mid R''_2$ with $R''_1 \equiv [\![Y_1, \langle\uparrow\rangle.\langle\rangle, S]\!]$ and $R''_2 \equiv [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S]\!]$, or the two parallel sub-processes have been merged by applying the $R\pi$ structural rule for split from right to left. In this last case no transition can be performed. Let us consider the first case. We have a case analysis depending on whether the last applied rule is (PAR+), (COM) or (CLOSE+).

- If rule (PAR+) is applied we have that $R''_1 \mid R''_2 \xrightarrow{(i,j,k):\alpha} T''_1 \mid R''_2$ with hypothesis $R''_1 \xrightarrow{(i,j,k):\alpha} T''_1$. Moreover, from Lemma 14 there exists R'''_1 such that $R''_1 = R'''_1@\langle\uparrow\rangle.\langle\rangle$ and by Lemma 17 we have $R'''_1 \xrightarrow{(i,j,k):\alpha} T'''_1$, where $T''_1 = T'''_1@\langle\uparrow\rangle.\langle\rangle$. Since $R'''_1 \equiv [\![Y_1, \langle\rangle, S]\!]$ we can apply the inductive hypothesis and obtain that

  $$S :: Y_1 \xrightarrow{(i,K,j):\alpha\!\!\rightarrow} S' :: Y'_1 \text{ with } [\![Y'_1, \langle\rangle, S']\!] \equiv T'''_1 \text{ and } K = k$$

  Since transition $\rightarrow$ is executed without applying a congruence rule, $\alpha \neq \tau$ and $S = S'$. We can now apply the framework rule (PAR) and derive the transition

  $$S :: Y_1 \mid Y_2 \xrightarrow{(i,K,j):\alpha\!\!\rightarrow} S' :: Y'_1 \mid Y_2$$

  and we conclude by noticing that

  $$[\![Y'_1 \mid Y_2, \langle\rangle, S']\!] = [\![Y'_1, \langle\uparrow\rangle.\langle\rangle, S']\!] \mid [\![Y_2, \langle\uparrow\rangle.\langle\rangle, S']\!] \equiv T''_1 \mid R''_2$$

- If rule (CLOSE+) is applied we have that

  $$R''_1 \mid R''_2 \xrightarrow{(i,*,*):\tau\!\!\rightarrow} \nu a_\Gamma(T''_1 \mid T''_{2\,[a/x]@i})$$

  with hypothesis $R''_1 \xrightarrow{(i,j,k):\overline{b}\langle\nu a_\Gamma\rangle\!\!\rightarrow} T''_1$ and $R''_2 \xrightarrow{(i,j',k'):b(x)\!\!\rightarrow} T''_2$ with $k =_* j'$ and $k' =_* j$.
  Moreover, from Lemma 14 there exist R'''_1 and R'''_2 such that $R''_1 = R'''_1@\langle\uparrow\rangle.\langle\rangle$ and $R''_2 = R'''_2@\langle\uparrow\rangle.\langle\rangle$.

  By Lemma 17 we have $R'''_1 \xrightarrow{(i,j,k):\overline{b}\langle\nu a_\Gamma\rangle\!\!\rightarrow} T'''_1$, where $T''_1 = T'''_1@\langle\uparrow\rangle.\langle\rangle$ and $R'''_2 \xrightarrow{(i,j',k'):b(x)\!\!\rightarrow} T'''_2$, where $T''_2 = T'''_2@\langle\uparrow\rangle.\langle\rangle$. Since $R'''_1 \equiv [\![Y_1, \langle\rangle, S]\!]$ and $R'''_2 \equiv [\![Y_2, \langle\rangle, S]\!]$, we can apply the inductive hypothesis and obtain that

$$S :: Y_1 \xrightarrow{(i,K,j):\overline{b}\langle \nu a_\Gamma\rangle} S :: Y_1' \text{ with } [\![Y_1', \langle\rangle, S]\!] \equiv T_1''' \text{ and } K = \{k\} \qquad \text{and}$$

$$S :: Y_2 \xrightarrow{(i,K',j'):b(x)} S :: Y_2' \text{ with } [\![Y_2', \langle\rangle, S]\!] \equiv T_2''' \text{ and } K' = \{k'\}$$

We can now apply framework rule (Close) and derive the transition

$$S :: Y_1 \mid Y_2 \xrightarrow{(i,\{*\},*):\tau} S \cup \{^{a^i}/_x\} :: \nu a_\Gamma(Y_1' \mid Y_2'\{^{a^i}/_x\})$$

and we conclude by noticing that

$$[\![\nu a_\Gamma(Y_1' \mid Y_2'\{^{a^i}/_x\}), \langle\rangle, S \cup \{^{a^i}/_x\}]\!] =$$
$$\nu a_\Gamma([\![Y_1', \langle\uparrow\rangle.\langle\rangle, S \cup \{^{a^i}/_x\}]\!] \mid [\![Y_2'\{^{a}/_x\}, \langle\uparrow\rangle.\langle\rangle, S \cup \{^{a^i}/_x\}]\!])$$
$$\equiv \nu a_\Gamma(T_1'' \mid T''_{2\,[a/x]@i})$$

- If rule (COM+) is applied, it is similar to the case above.

$X = \nu a_\Gamma(Y)$: we have that $R = [\![\nu a_\Gamma(Y), \langle\rangle, S]\!]$ and by applying the encoding with $m = \langle\rangle$, we have $[\![\nu a_\Gamma(Y), \langle\rangle, S]\!] = \nu a_\Gamma[\![Y, \langle\rangle, S]\!]$. Take any term $R'' \equiv \nu a_\Gamma[\![Y, \langle\rangle, S]\!]$. There are two cases: either $R'' = \nu a_\Gamma(R_1'')$ with $R_1'' \equiv [\![Y, \langle\rangle, S]\!]$ for any Γ, or for $\Gamma = \emptyset$ the restriction has been put back inside the term using the $R\pi$ structural rule for restriction. In this last case no transition can be performed. Let us consider the first case. Depending on the action α, rules (NEW+), (OPEN+) and (CAUSE REF+) can be used.

- If rule (OPEN+) is applied, we have $\nu a_\Gamma(R_1'') \xrightarrow{(i,j,k):\overline{b}\langle \nu a_\Gamma\rangle} \nu a_{\Gamma+i}(R''')$ with hypothesis $R_1'' \xrightarrow{(i,j,k):\beta} R'''$ where $\beta = \overline{b}\langle \nu a_{\Gamma'}\rangle \vee \beta = \overline{b}a$. Let us choose $\beta = \overline{b}\langle \nu a_{\Gamma'}\rangle$; the other case is similar. By applying the inductive hypothesis we obtain that

  $$S :: Y \xrightarrow{(i,K,j):\overline{b}\langle \nu a_{\Gamma'}\rangle} S' :: Y'$$

  with $[\![Y', \langle\rangle, S']\!] \equiv R'''$ and $K = \{k\}$. Since the executed action is not a synchronisation, we have $S = S'$. By applying framework rule (Open) we also have

  $$S :: \nu a_\Gamma(Y) \xrightarrow{(i,K,j):\overline{b}\langle \nu a_\Gamma\rangle} S :: \nu a_{\Gamma+i}(Y')$$

  The thesis follows since $[\![\nu a_{\Gamma+i}(Y'), \langle\rangle, S]\!] = \nu a_{\Gamma+i}[\![Y', \langle\rangle, S]\!] \equiv \nu a_{\Gamma+i}(R''')$.
- If rule (CAUSE REF+) is applied, we have

  $$\nu a_\Gamma(R_1'') \xrightarrow{(i,j,k'):\alpha} \nu a_\Gamma(R'''_{[k'/k]@i})$$

  with hypothesis $R_1'' \xrightarrow{(i,j,k):\alpha} R'''$ where $a \in \mathtt{subj}(\alpha)$, $\Gamma \neq \emptyset$ and $k = k' \vee k \rightsquigarrow_{R_1''} k'$.
  By applying the inductive hypothesis we obtain that

  $$S :: Y \xrightarrow{(i,K,j):\alpha} S' :: Y'$$

  with $[\![Y', \langle\rangle, S']\!] \equiv R'''$ and $K = \{k\}$. Since $\alpha \neq \tau$, we have $S = S'$. By applying framework rule (Cause Ref) we also have

  $$S :: \nu a_\Gamma(Y) \xrightarrow{(i,K',j):\alpha} S :: \nu a_\Gamma(Y'_{[K'/K]@i})$$

  Since $R_1'' = [\![Y, \langle\rangle, S]\!]$, the restriction $\nu a_{\Gamma''} \in R_1''$ iff $\nu a_{\Gamma''} \in Y$, hence $K' = \{k'\}$. The thesis follows since

  $$[\![\nu a_\Gamma(Y'_{[K'/K]@i}), \langle\rangle, S]\!] = \nu a_\Gamma[\![Y'_{[K'/K]@i}, \langle\rangle, S]\!] \equiv \nu a_\Gamma(R'''_{[k'/k]@i})$$

- If rule (NEW+) is applied, it is similar to the cases above. □

Appendix C

Here we give the proof of Lemma 7 from Section 6.2.3.

Lemma 7 *(Structural correspondence). Starting from an initial π-calculus process P, where $P = \varphi(\mathbf{P})$, we have:*

1. *if $P \xrightarrow[\mathbb{K}_1]{\zeta_1} A_1 \dots A_{n-1} \xrightarrow[\mathbb{K}_n]{\zeta_n} A_n$ is a trace in causal semantics [20], then there exists a trace $\mathbf{P} \xrightarrow{\mu_1} X_1^{K_{F1}} \dots X_{n-1}^{K_{Fn-1}} \xrightarrow{\mu_n} X_n^{K_{Fn}}$ and K_{Fi} in the framework, such that for all i, $\lambda(A_i) = \varphi(X_i^{K_{Fi}})$, $\zeta_i = \gamma(\mu_i)$ and $\mathtt{Rem}(K_{Fi}) = \mathbb{K}_i$, for $i = 1, \dots, n$.*

2. *if* $\mathbf{P} \xrightarrow{\mu_1} X_1^{K_{F1}} \dots X_{n-1}^{K_{Fn-1}} \xrightarrow{\mu_n} X_n^{K_{Fn}}$ *is a trace in the framework, then there exists a trace* $P \xrightarrow[\mathtt{K}_1]{\zeta_1} A_1 \dots A_{n-1} \xrightarrow[\mathtt{K}_n]{\zeta_n} A_n$ *in the causal semantics, where for all i,* $\lambda(A_i) = \varphi(X_i^{K_{Fi}})$, $\zeta_i = \gamma(\mu_i)$ *and* $\mathtt{Rem}(K_{Fi}) = \mathtt{K}_i$, *for* $i = 1, \dots, n$.

Proof. Both clauses (*1.* and *2.*) are proved by induction on the length of the computation. Let us consider clause *1.*

(I) The base case is given by 0 transitions. Trivially, the theorem holds.
(II) In the inductive case, for $P = \varphi(\mathbf{P})$, we let

$$s_{BS} : P \xrightarrow[\mathtt{K}_1]{\zeta_1} A_1 \dots A_{n-1} \xrightarrow[\mathtt{K}_n]{\zeta_n} A_n \quad \text{and} \quad s_F : \mathbf{P} \xrightarrow{\mu_1} X_1^{K_{F1}} \dots X_{n-1}^{K_{Fn-1}} \xrightarrow{\mu_n} X_n^{K_{Fn}}$$

be the traces on causal processes and processes from the framework, respectively. Let us suppose that the inductive hypothesis holds for these two traces. We need to prove that for

$$s_{BS} \xrightarrow[\mathtt{K}_{n+1}]{\zeta_{n+1}} A_{n+1} \qquad \text{and} \qquad s_F \xrightarrow{\mu_{n+1}} X_{n+1}^{K_{Fn+1}}$$

the theorem holds. Hence, we need to show $\mathtt{Rem}(K_{Fn+1}) = \mathtt{K}_{n+1}$ and $\lambda(A_{n+1}) = \varphi(X_{n+1}^{K_{Fn+1}})$.

We proceed by structural induction on the process A_n and the last applied rule on the transition $A_n \xrightarrow[\mathtt{K}_{n+1}]{\zeta_{n+1}} A_{n+1}$. We have two main cases depending whether A_n is a π-calculus process, or it is a causal process. If $A_n = P$, where P is a π-calculus process, then it is the first transition; hence the cause sets are empty ($\mathtt{K}_1 = K_{F1} = \emptyset$), and we have:

- $P = \pi.P'$ where $\pi = \overline{b}a$ or $\pi = b(x)$. The rules that can be applied in Boreale and Sangiorgi's semantics are (BS-Out) and (BS-In). We show the case when rule (BS-Out) is applied; the other case is similar.
  We have $\overline{b}a.P' \xrightarrow{i_1 : \overline{b}a} \{i_1\} :: P' = A_1$ where $\lambda(A_1) = P'$.
  In the framework, by applying the rule (Out1), we can execute the corresponding action $\mu_1 = (i_1, \{*\}, *) : \overline{b}a$, where $\gamma(\mu_1) = i_1 : \overline{b}a$. We have:

  $$\overline{b}^* a^*.\mathbf{P}' \xrightarrow{(i_1, \{*\}, *) : \overline{b}a} \overline{b}^* a^*[i_1, \{*\}].\mathbf{P}' = X_1$$

  with $\varphi(\mathbf{P}') = P'$ and $\varphi(X_1) = P'$ as desired.
- $P = Q \mid Q'$. The rules that can be applied in Boreale and Sangiorgi's semantics are (BS-Par), (BS-Com) and (BS-Close). We show the case when the rule (BS-Close) is used; the rest of the cases are similar.
  Since rule (BS-Close) is applied on the process $Q \mid Q'$ one of the parallel components needs to extrude a bound name. Let it be a process $Q = \nu a(Q_1)$. Then we have

  $$\nu a(Q_1) \mid Q' \xrightarrow{\tau} \nu a(Q_2 \mid Q''\{^a/_x\}) = A_1$$

  with the premises $\nu a(Q_1) \xrightarrow{i_1 : \overline{b}\langle \nu a \rangle} Q_2$ and $Q' \xrightarrow{i_1 : b(x)} Q''$, for some name b. Since τ actions do not impose causes and there is no cause set to merge, we have $\lambda(A_1) = \nu a(Q_2 \mid Q''\{^a/_x\})$.
  In the framework we have $\varphi(\mathbf{Q}_1) = Q_1$ and $\varphi(\mathbf{Q}') = Q'$. Hence there exist actions $\mu_1' = (i_1, \{*\}, *) : \overline{b}\langle \nu a_{\emptyset_*} \rangle$ and $\mu_1'' = (i_1, \{*\}, *) : b(x)$ such that $\gamma(\mu_1') = i_1 : \overline{b}\langle \nu a \rangle$ and $\gamma(\mu_1'') = i_1 : b(x)$. Then we have:

  $$\nu a_{\emptyset_*}(\mathbf{Q}_1) \xrightarrow{\mu_1'} \nu a_{\{i_1\}_{i_1}}(Y_1) \quad \text{and} \quad \mathbf{Q}' \xrightarrow{\mu_1''} Y''$$

  where $\varphi(Y_1) = Q_2$ and $\varphi(Y'') = Q''$.
  By applying the rule (Close) on the process $\nu a_{\emptyset_*}(\mathbf{Q}_1) \mid \mathbf{Q}'$ with the given premises, we get

  $$\nu a_{\emptyset_*}(\mathbf{Q}_1) \mid \mathbf{Q}' \xrightarrow{(i_1, \{*\}, *) : \tau} \nu a_{\emptyset_*}(\nu a_{\{i_1\}_*}(Y_1) \mid Y''\{^{a^{i_1}}/_x\}) = X_1$$

  Then we have

  $$\varphi(\nu a_{\emptyset_*}(\nu a_{\{i_1\}_*}(Y_1) \mid Y''\{^{a^{i_1}}/_x\})) = \nu a(Q_2 \mid Q''\{^a/_x\})$$

  as desired.
- $P = \nu a(P')$. The rules that can be applied in Boreale and Sangiorgi's semantics are (BS-Res) and (BS-Open), depending on whether the name a belongs to the executing action or not. We show the case when rule (BS-Open) is applied; the other case is similar to the one above.

If the rule (BS-Open) is applied on the process $\nu a(P')$, the executed action extrudes name a, and we have:

$$\nu a(P') \xrightarrow{i_1:\overline{b}\langle \nu a\rangle} \{i_1\} :: P'' = A_1$$

with the premise $P' \xrightarrow{i_1:\overline{b}a} \{i_1\} :: P''$. By discarding cause set $\{i_1\}$ we have $\lambda(A_1) = P''$.
In the framework we have $\varphi(\mathbf{P}') = P'$. Hence there exists an action $\mu_1 = (i_1, \{*\}, *) : \overline{b}a$ such that $\gamma(\mu_1) = i_1 : \overline{b}a$. By executing the action μ_1, we obtain: $\mathbf{P}' \xrightarrow{(i_1,\{*\},*):\overline{b}a} Y'$ where $\varphi(Y') = P''$.
Now, by applying the rule (Open) on the process $\nu a_{\emptyset_*}(\mathbf{P}')$ (where $\varphi(\nu a_{\emptyset_*}(\mathbf{P}')) = \lambda(\nu a(P')) = \nu a(P')$), with the given premises, we obtain

$$\nu a_{\emptyset_*}(\mathbf{P}') \xrightarrow{(i_1,\{*\},*):\overline{b}\langle \nu a_{\emptyset_*}\rangle} \nu a_{\{i_1\}_{i_1}}(Y') = X_1$$

where $\gamma((i_1, \{*\}, *) : \overline{b}\langle \nu a_{\emptyset_*}\rangle) = i_1 : \overline{b}\langle \nu a\rangle$. By discarding the elements of the history from the process X_1, we have $\varphi(X_1) = P''$ as desired.

If A_n is a casual process, then we have the following cases:

- $A_n = \mathtt{K}' :: A$. In the BS semantics, the only applicable rule is (BS-Cau) and we have:

$$A_n = \mathtt{K}' :: A \xrightarrow[\mathtt{K}',\mathtt{K}]{i_{n+1}:\alpha} \mathtt{K}' :: A' = A_{n+1}$$

with the premise $A \xrightarrow[\mathtt{K}]{i_{n+1}:\alpha} A'$ and $\mathtt{K}_{n+1} = \mathtt{K}' \cup \mathtt{K}$.
From the inductive hypothesis, we have $\lambda(A_n) = \varphi(X_n)$. Hence, in the framework, there exist process X and history context $\mathtt{H}$ such that $X_n = \mathtt{H}[X]$, where $\varphi(X) = \lambda(A)$ and context $\mathtt{H}$ corresponds to cause set $\mathtt{K}'$.
From $\varphi(X) = \lambda(A)$, there exists an action $\mu_{n+1} = (i_{n+1}, K_{n+1}, j_{n+1}) : \alpha'$ where $\gamma(\mu_{n+1}) = i_{n+1} : \alpha$, such that

$$X \xrightarrow{(i_{n+1},K_{n+1},j_{n+1}):\alpha'} Y^{K_F}$$

Now, we can apply the rules (IN2) or (OUT2) in the framework, depending on the nature of the action α', on the process $X_n = \mathtt{H}[X]$ with the premise given above, and we obtain:

$$\mathtt{H}[X] \xrightarrow{(i_{n+1},K_{n+1},j_{n+1}):\alpha'} \mathtt{H}[Y]^{K'_F \cup K_F}$$

where the set of the causes K'_F represents the causes of the action α' belonging to the history context $\mathtt{H}$. Hence, we have $K_{Fn+1} = K'_F \cup K_F$.
To show that actions μ_{n+1} and ζ_{n+1} happened on the same part of the π-calculus process $\varphi(\mathtt{H}[X]) = \lambda(\mathtt{K}' :: A)$, which would imply that $\varphi(\mathtt{H}[Y]^{K'_F \cup K_F}) = \lambda(\mathtt{K}' :: A')$, we need to prove that two actions had corresponding cause sets, i.e. we need to prove $\mathtt{Rem}(K_{Fn+1}) = \mathtt{K}_{n+1}$. To do it, we should look at the action ζ_n because it is the last action that can influence the cause set $\mathtt{K}_{n+1}$ (cause set $\mathtt{K}_{n+1}$ does not depend on the action ζ_{n+1}). There are two main cases:
- action ζ_n is the direct structural cause of the action ζ_{n+1} (it happened on the same component in the parallel composition, immediately before the action ζ_{n+1}). Then the action ζ_n is a visible action and $\mathtt{K}_{n+1} = \mathtt{K}_n \cup \{i_n\}$ where $\mathtt{K}_n = \mathtt{K}' \cup \mathtt{K} \setminus \{i_n\}$. By inductive hypothesis, we have that there exist $K_{Fn}, \mu_n \in s_F$ such that $\gamma(\mu_n) = \zeta_n$ and $\mathtt{Rem}(K_{Fn}) = \mathtt{K}_n$. The action μ_n is identified with a key i_n and it is a direct structural cause of the action i_{n+1}; therefore we have $K_{Fn+1} = K_{Fn} \cup \{i_n\}$. The method $\mathtt{Rem}$ (Definition 41) does not remove keys of the visible actions and we have $\mathtt{Rem}(K_{Fn} \cup \{i_n\}) = \mathtt{K}' \cup \mathtt{K} \setminus \{i_n\} \cup \{i_n\} = \mathtt{K}' \cup \mathtt{K} = \mathtt{K}_{n+1}$ as desired.
- action ζ_n is not the direct cause of the action ζ_{n+1}; then $\zeta_n = i_n : \tau$ or ζ_n happened on a different component in the parallel composition from the action ζ_{n+1}.
  - If $\zeta_n = i_n : \tau$, then there exist $\mathtt{K}_j$, $\mathtt{K}_h \in s_{BS}$ such that $\mathtt{K}_j$ is a cause set of the input action and $\mathtt{K}_h$ is a cause set of the output action which participated in the τ move. Since τ actions merge cause sets, we have that $\mathtt{K}_{n+1} = \mathtt{K}' \cup \mathtt{K} = \mathtt{K}_j \cup \mathtt{K}_h$.
    In the framework, a τ move is composed of the same input and output actions as in the trace on the causal processes. Hence, there exist K_{Fj}, $K_{Fh} \in s_F$, and by inductive hypothesis $\mathtt{Rem}(K_{Fj}) = \mathtt{K}_j$ and $\mathtt{Rem}(K_{Fh}) = \mathtt{K}_h$. Since in the framework the τ-action is identified with the key i_n we have that $K_{Fn+1} = K_{Fj} \cup K_{Fh} \cup \{i_n\}$. By Definition 41, method $\mathtt{Rem}$ removes keys belonging to the τ actions; hence, we have $\mathtt{Rem}(K_{Fj} \cup K_{Fh} \cup \{i_n\}) = \mathtt{K}_j \cup \mathtt{K}_h = \mathtt{K}_{n+1}$ as desired.
  - If ζ_n happened on a different component in the parallel composition, there exist $\mathtt{K}_h, \zeta_h \in s_{BS}$ where ζ_h is the last action on the same component in the parallel composition as ζ_{n+1}. Then we have that $\mathtt{K}_{h+1} = \mathtt{K}_h \cup \{i_h\}$ and $\mathtt{K}_{n+1} = \mathtt{K}' \cup \mathtt{K} = \mathtt{K}_{h+1}$, since ζ_h was the last action before ζ_{n+1}.
    By inductive hypothesis, in the framework, there exist $K_{Fh}, \mu_h \in s_F$, where $\gamma(\mu_h) = \zeta_h$ and $K_{Fh+1} = K_{Fh} \cup \{i_h\}$. We can notice that $\mathtt{Rem}(K_{Fh+1}) = \mathtt{K}_{h+1}$ such that $K_{Fh+1} = K_{Fn+1}$.

- $A_n = A_{n1} \mid A_{n2}$. In the BS semantics, applicable rules are (BS-Par), (BS-Close) and (BS-Com).
  ∘ Let us first consider the case when rule (BS-Par) is applied. We have:

$$A_n = A_{n1} \mid A_{n2} \xrightarrow[\mathtt{K}_{n+1}]{i_{n+1}:\alpha} A'_{n1} \mid A_{n2}$$

with premise $A_{n1} \xrightarrow[\mathtt{K}_{n+1}]{i_{n+1}:\alpha} A'_{n1}$ where $\mathtt{Bnv}(\alpha) \cap \mathtt{Fnv}(A_{n2}) = \emptyset$.
From $\lambda(A_{n1} \mid A_{n2}) = \lambda(A_{n1}) \mid \lambda(A_{n2}) = \varphi(X_{n1}) \mid \varphi(X_{n2})$ we have that action $(i_{n+1}, K_{n+1}, j_{n+1}) : \alpha'$ exists where $\gamma((i_{n+1}, K_{n+1}, j_{n+1}) : \alpha') = i_{n+1} : \alpha$, such that

$$X_{n1} \xrightarrow{(i_{n+1}, K_{n+1}, j_{n+1}):\alpha'} Y_{n1}^{K_{Fn+1}}$$

On the process $X_n = X_{n1} \mid X_{n2}$ we can apply the rule (Par) from the framework, and we have:

$$X_{n1} \mid X_{n2} \xrightarrow{(i_{n+1}, K_{n+1}, j_{n+1}):\alpha'} Y_{n1}^{K_{Fn+1}} \mid X_{n2}$$

To prove $\mathtt{Rem}(K_{Fn+1}) = \mathtt{K}_{n+1}$ we should look at the action ζ_n because it is the last action that can influence the cause set $\mathtt{K}_{n+1}$ (cause set $\mathtt{K}_{n+1}$ does not depend on the action ζ_{n+1}). We then reason much as in the previous case.
Once we have proved $\mathtt{Rem}(K_{Fn+1}) = \mathtt{K}_{n+1}$, we can conclude that

$$\lambda(A'_{n1} \mid A_{n2}) = \lambda(A'_{n1}) \mid \lambda(A_{n2}) = \varphi(Y_{n1}^{K_{Fn+1}}) \mid \varphi(X_{n2})$$

as desired.
∘ Suppose that the applied rule is (BS-Close). Then we have:

$$A_{n1} \mid A_{n2} \xrightarrow{\tau} \nu a(A'_{n1}[i_{n+1} \rightsquigarrow \mathtt{K}''_{n+1}] \mid A'_{n2}\{a/x\}[i_{n+1} \rightsquigarrow \mathtt{K}'_{n+1}]) = A_{n+1}$$

with premises $A_{n1} \xrightarrow[\mathtt{K}'_{n+1}]{i_{n+1}:\overline{b}\langle \nu a\rangle} A'_{n1}$ and $A_{n2} \xrightarrow[\mathtt{K}''_{n+1}]{i_{n+1}:b(x)} A'_{n2}$. Since function λ deletes cause sets we have:

$$\lambda(A_{n+1}) = \nu a\lambda((A'_{n1} \mid A'_{n2}\{a/x\})) = \nu a\lambda(A'_{n1}) \mid \lambda(A'_{n2}\{a/x\})$$

From the inductive hypothesis, we have $\lambda(A_n)=\varphi(X_n)$; hence we have $\lambda(A_{n1} \mid A_{n2})=\lambda(A_{n1}) \mid \lambda(A_{n2})=\varphi(X_{n1}) \mid \varphi(X_{n2})$. From $\lambda(A_{n1}) = \varphi(X_{n1})$ we know that action $(i_{n+1}, K'_{n+1}, j'_{n+1}) : \overline{b}\langle \nu a_{\emptyset_*}\rangle$ exists such that $\gamma((i_{n+1}, K'_{n+1}, j'_{n+1}) : \overline{b}\langle \nu a_{\emptyset_*}\rangle) = i_{n+1} : \overline{b}\langle \nu a\rangle$ and

$$X_{n1} \xrightarrow{(i_{n+1}, K'_{n+1}, j'_{n+1}):\overline{b}\langle \nu a_{\emptyset_*}\rangle} Y_{n1}^{K'_{Fn+1}}$$

Similarly, from $\lambda(A_{n2}) = \varphi(X_{n2})$ we know that action $(i_{n+1}, K''_{n+1}, j''_{n+1}) : b(x)$ exists such that $\gamma((i_{n+1}, K''_{n+1}, j''_{n+1}) : b(x)) = i_{n+1} : b(x)$ and

$$X_{n2} \xrightarrow{(i_{n+1}, K''_{n+1}, j''_{n+1}):b(x)} Y_{n2}^{K''_{Fn+1}}$$

The condition on the rule (Close) is satisfied since by definition of the set K when data structure Γ_w is used, we have $* \in K'_{n+1}$ and $* \in K''_{n+1}$ which makes $K'_{n+1} =_* j''_{n+1}$ and $K''_{n+1} =_* j'_{n+1}$ true.
Now we can apply the rule (Close) on the process $X_n = X_{n1} \mid X_{n2}$ with the given premises, and obtain:

$$X_{n1} \mid X_{n2} \xrightarrow{(i_{n+1}, \{*\}, *):\tau} \nu a_{\emptyset_*}((Y_{n1}^{K'_{Fn+1}})_{\#i_{n+1}} \mid Y_{n2}^{K''_{Fn+1}}\{^{a^{i_{n+1}}}/_x\})$$

where

$$\varphi(\nu a_{\emptyset_*}((Y_{n1}^{K'_{Fn+1}})_{\#i_{n+1}} \mid Y_{n2}^{K''_{Fn+1}}\{^{a^{i_{n+1}}}/_x\})) = \nu a\varphi((Y_{n1}^{K'_{Fn+1}})_{\#i_{n+1}} \mid Y_{n2}^{K''_{Fn+1}}\{^{a^{i_{n+1}}}/_x\}))$$

Since the operator $\#i_{n+1}$ is applied on the memories of the type νa_{Γ_w} when $\mathtt{empty}(\Gamma_w) \neq \mathtt{true}$ it is deleted with the function φ. Therefore, we have:

$$\nu a\varphi((Y_{n1}^{K'_{Fn+1}})_{\#i_{n+1}} \mid Y_{n2}^{K''_{Fn+1}}\{^{a^{i_{n+1}}}/_x\})) = \nu a\varphi(Y_{n1}^{K'_{Fn+1}})) \mid \varphi(Y_{n2}^{K''_{Fn+1}}\{^{a^{i_{n+1}}}/_x\})$$

Now we need to prove $\mathtt{Rem}(K'_{Fn+1}) = \mathtt{K}'_{n+1}$ and $\mathtt{Rem}(K''_{Fn+1}) = \mathtt{K}''_{n+1}$.
To prove it we should look at the action ζ_n because it is the last action that could influence the cause sets $\mathtt{K}'_{n+1}$ and $\mathtt{K}''_{n+1}$ (cause sets $\mathtt{K}'_{n+1}$ and $\mathtt{K}''_{n+1}$ do not depend on the action ζ_{n+1}). There are two main cases:

– action ζ_n is the direct structural cause of the action ζ_{n+1}. Then we have that action ζ_n was a visible action identified with the key i_n and since action ζ_{n+1} is a communication, we have three different possibilities: $\mathtt{K}'_{n+1} = \mathtt{K}_n \cup \{i_n\}$ or $\mathtt{K}''_{n+1} = \mathtt{K}_n \cup \{i_n\}$ or $\mathtt{K}'_{n+1} = \mathtt{K}''_{n+1} = \mathtt{K}_n \cup \{i_n\}$.
Let us consider the first case, when $\mathtt{K}'_{n+1} = \mathtt{K}_n \cup \{i_n\}$; the other cases are similar. We can notice that there exists some $\mathtt{K}_l \in s_{BS}$ such that $\mathtt{K}_l = \mathtt{K}''_{n+1}$ (since $\mathtt{K}''_{n+1}$ does not contain action i_n, it is computed in the trace s_{BS} before execution of the action i_n). By inductive hypothesis we have that exist $K_{Fn}, K_{Fl}, \mu_n \in s_F$ such that:
(i) $\mathtt{Rem}(K_{Fl}) = \mathtt{K}_l$ and $K_{Fl} = K''_{Fn+1}$, hence $\mathtt{Rem}(K''_{Fn+1}) = \mathtt{K}''_{n+1}$;
(ii) $\gamma(\mu_n) = \zeta_n$ and $\mathtt{Rem}(K_{Fn}) = \mathtt{K}_n$. The action μ_n is identified with a key i_n and it is visible and direct structural cause of the action μ_{n+1}, where $\gamma(\mu_{n+1}) = \zeta_{n+1}$; therefore, we have $K'_{Fn+1} = K_{Fn} \cup \{i_n\}$. The method $\mathtt{Rem}$ (Definition 41) does not remove keys of the visible actions and we have $\mathtt{Rem}(K_{Fn} \cup \{i_n\}) = \mathtt{K}_n \cup \{i_n\} = \mathtt{K}'_{n+1}$ as desired.

– action ζ_n is not the direct cause of the action ζ_{n+1}; then $\zeta_n = i_n : \tau$ or ζ_n happened on a different component in the parallel composition from the action ζ_{n+1}.
  – If $\zeta_n = i_n : \tau$, then there exist $\mathtt{K}_j,\ \mathtt{K}_h \in s_{BS}$ such that $\mathtt{K}_j$ is a cause set of the input action and $\mathtt{K}_h$ is a cause set of the output action which participated in the τ move. Since τ actions merge cause sets, we have that $\mathtt{K}'_{n+1} = \mathtt{K}_j \cup \mathtt{K}_h$. For the cause set $\mathtt{K}''_{n+1}$ we have two options: either $\mathtt{K}''_{n+1} = \mathtt{K}'_{n+1}$ or there exists some $\mathtt{K}_l \in s_{BS}$ such that $\mathtt{K}_l = \mathtt{K}''_{n+1}$ (since $\mathtt{K}''_{n+1}$ does not contain action i_n, it is computed in the trace s_{BS} before execution of the action i_n) in which case the proof is similar to the one above.
    In the framework, a τ move is composed of the same input and output actions as in the trace on the causal processes. Hence, there exist $K_{Fj},\ K_{Fh} \in s_F$, and by inductive hypothesis $\mathtt{Rem}(K_{Fj}) = \mathtt{K}_j$ and $\mathtt{Rem}(K_{Fh}) = \mathtt{K}_h$. Since in the framework the τ-action is identified with the key i_n we have that $K'_{Fn+1} = K_{Fj} \cup K_{Fh} \cup \{i_n\}$. By Definition 41, method $\mathtt{Rem}$ removes keys belonging to the τ actions; hence, we have $\mathtt{Rem}(K_{Fj} \cup K_{Fh} \cup \{i_n\}) = \mathtt{K}_j \cup \mathtt{K}_h = \mathtt{K}'_{n+1}$ as desired.
  – If ζ_n happened on a different component in the parallel composition, there exist $\mathtt{K}_h, \zeta_h \in s_{BS}$ where ζ_h is the last action on the same component in the parallel composition as ζ_{n+1} and $\mathtt{K}_{h+1} = \mathtt{K}_h \cup \{i_h\}$. Then we have that $\mathtt{K}'_{n+1} = \mathtt{K}_{h+1}$, since ζ_h was the last action before ζ_{n+1}.
  By inductive hypothesis, in the framework, there exist $K_{Fh}, \mu_h \in s_F$, where $\gamma(\mu_h) = \zeta_h$ and $K_{Fh+1} = K_{Fh} \cup \{i_h\}$. Then we have $\mathtt{Rem}(K_{Fh+1}) = \mathtt{K}_{h+1}$ such that $K_{Fh+1} = K'_{Fn+1}$ as desired. Similarly for $\mathtt{K}''_{n+1}$.

By having $\mathtt{Rem}(K'_{Fn+1}) = \mathtt{K}'_{n+1}$ and $\mathtt{Rem}(K''_{Fn+1}) = \mathtt{K}''_{n+1}$, we can conclude that $\varphi(Y_{n1}^{K'_{Fn+1}}) = \lambda(A'_{n1})$ and $\varphi(Y_{n2}^{K''_{Fn+1}}) = \lambda(A'_{n2})$ which concludes our case.

∘ The case when the rule (BS-Com) is applied in the BS semantics is similar to the case with the rule (BS-Close)

- $A_n = \nu a(A)$. In the BS semantics, applicable rules are (BS-Res) and (BS-Open).
Let us first consider the case when rule (BS-Open) is applied. We have:

$$A_n = \nu a(A) \xrightarrow[\mathtt{K}_{n+1}]{i_{n+1}:\overline{b}\langle \nu a\rangle} A' = A_{n+1}$$

with premise $A \xrightarrow[\mathtt{K}_{n+1}]{i_{n+1}:\overline{b}a} A'$.
From inductive hypothesis, there exists a process X, such that $\lambda(A_n) = \varphi(\nu a_{\emptyset_*}(X))$ and $\lambda(A) = \varphi(X)$. Then, there exists an action $(i_{n+1}, K_{n+1}, j_{n+1}) : \alpha'$, where $\gamma((i_{n+1}, K_{n+1}, j_{n+1}) : \alpha') = i_{n+1} : \overline{b}a$ such that

$$X \xrightarrow{(i_{n+1}, K_{n+1}, j_{n+1}):\alpha'} Y^{K_{Fn+1}}$$

where $\alpha' = \overline{b}\langle \nu a_{\Gamma_w}\rangle$ or $\alpha' = \overline{b}a$.
On the process X_n we can apply the (Open) rule from the framework and we have:

$$\nu a_{\emptyset_*}(X) \xrightarrow{(i_{n+1}, K_{n+1}, j_{n+1}):\overline{b}\langle \nu a_{\emptyset_*}\rangle} \nu a_{\{i_{n+1}\}_{i_{n+1}}}(Y^{K_{Fn+1}})$$

Now we need to prove $\mathtt{Rem}(K_{Fn+1}) = \mathtt{K}_{n+1}$, to obtain $\lambda(A') = \varphi(Y^{K_{Fn+1}})$, which implies $\lambda(A_{n+1}) = \lambda(A')$ and $\varphi(X_{n+1}) = \varphi(\nu a_{\{i_{n+1}\}_{i_{n+1}}}(Y^{K_{Fn+1}})) = \varphi(Y^{K_{Fn+1}}) = \lambda(A')$.
To prove $\mathtt{Rem}(K_{Fn+1}) = \mathtt{K}_{n+1}$ we should look at the action ζ_n because it is the last action that can influence the cause set $\mathtt{K}_{n+1}$ (cause set $\mathtt{K}_{n+1}$ does not depend on the action ζ_{n+1}). We then reason much as in the case when $A_n = \mathtt{K}' :: A$.
- The case when the rule (BS-Res) is applied is similar to the one above.

Let us now consider clause *2*.

(I) The base case is given by 0 transitions. Trivially, the theorem holds.
(II) In the inductive case, for $P = \varphi(\mathbf{P})$, we let

$$s_F : \mathbf{P} \xrightarrow{\mu_1} X_1^{K_{F1}} \dots X_{n-1}^{K_{Fn-1}} \xrightarrow{\mu_n} X_n^{K_{Fn}} \quad \text{and} \quad s_{BS} : P \xrightarrow[\mathtt{K}_1]{\zeta_1} A_1 \dots A_{n-1} \xrightarrow[\mathtt{K}_n]{\zeta_n} A_n$$

be the traces on the processes from the framework and on causal processes, respectively. Let us suppose that the inductive hypothesis holds for these two traces. We need to prove that the Lemma holds for

$$s_F \xrightarrow{\mu_{n+1}} X_{n+1}^{K_{Fn+1}} \qquad \text{and} \qquad s_{BS} \xrightarrow{\zeta_{n+1}}_{\mathbb{K}_{n+1}} A_{n+1}$$

Hence, we need to show $\mathtt{Rem}(K_{Fn+1}) = \mathbb{K}_{n+1}$ and $\lambda(A_{n+1}) = \varphi(X_{n+1}^{K_{Fn+1}})$.

We proceed by structural induction on the process $X_n^{K_{Fn}}$ and the last applied rule on the transition $X_n^{K_{Fn}} \xrightarrow{\mu_{n+1}} X_{n+1}^{K_{Fn+1}}$. We have two main cases depending whether $X_n^{K_{Fn}}$ is a π-calculus process, or it is a reversible process with past. If $X_n^{K_{Fn}} = \mathbf{P}$, where $\varphi(\mathbf{P}) = P$ and P is a π-calculus process, then it is the first transition; hence cause sets are empty ($K_{F1} = \mathbb{K}_1 = \emptyset$), and we have:

- $\mathbf{P} = \overline{b}^* a^*.\mathbf{P}'$ or $\mathbf{P} = b^*(x).\mathbf{P}'$ where $\varphi(\mathbf{P}) = P$ and $\varphi(\mathbf{P}') = P'$. The rules that can be applied in the framework are (Out1) and (In1), respectively. We show the case when the rule (Out1) is applied; the other case is similar. We have

  $$\overline{b}^* a^*.\mathbf{P}' \xrightarrow{(i_1,\{*\},*):\overline{b}a} \overline{b}^* a^*[i_1, \{*\}].\mathbf{P}' = X_1$$

  where $\varphi(X_1) = P'$ and $\gamma((i_1, \{*\}, *) : \overline{b}a) = i_1 : \overline{b}a$.
  In Boreale and Sangiorgi's semantics we can apply the rule (BS-Out) on the process $\overline{b}a.P'$, and we have

  $$\overline{b}a.P' \xrightarrow{i_1:\overline{b}a} \{i_1\} :: P' = A_1$$

  where $\lambda(A_1) = P'$ as desired.
- $\mathbf{P} = \mathbf{Q}_1 \mid \mathbf{Q}_2$ where $\varphi(\mathbf{Q}_1) = Q_1$ and $\varphi(\mathbf{Q}_2) = Q_2$. The rules that can be applied in the framework are (Par), (Com) and (Close). We show the case when the rule (Close) is used; the rest of the cases are similar.
  Since rule (Close) is applied on the process $\mathbf{Q}_1 \mid \mathbf{Q}_2$ one of the parallel components needs to extrude a bound name. Let it be a process $\mathbf{Q}_1 = \nu a_{\emptyset_*}(\mathbf{Q'}_1)$. Then we have:

  $$\nu a_{\emptyset_*}(\mathbf{Q'}_1) \mid \mathbf{Q_2} \xrightarrow{(i_1,\{*\},*):\tau} \nu a_{\emptyset_*}(\nu a_{\{i_1\}_*}(Y_1) \mid Y_2\{^{a^{i_1}}/_x\}) = X_1$$

  with premises

  $$\nu a_{\emptyset_*}(\mathbf{Q'}_1) \xrightarrow{\mu'_1} \nu a_{\{i_1\}_{i_1}}(Y_1) \quad \text{and} \quad \mathbf{Q_2} \xrightarrow{\mu''_1} Y_2$$

  where $\varphi(Y_1) = Q''_1$ and $\varphi(Y_2) = Q'_2$. Hence we have

  $$\begin{aligned}\varphi(\nu a_{\emptyset_*}(\nu a_{\{i_1\}_*}(Y_1) \mid Y_2\{^{a^{i_1}}/_x\})) &= \nu a(\varphi(Y_1) \mid \varphi(Y_2\{^{a^{i_1}}/_x\})) \\ &= \nu a(Q''_1 \mid Q'_2\{^a/_x\})\end{aligned}$$

  From the inductive hypothesis, we have $\varphi(\mathbf{Q'}_1) = Q'_1$ and there exist ζ'_1 and ζ'_1 such that $\gamma(\mu'_1) = \zeta'_1$ and $\gamma(\mu''_1) = \zeta''_1$. Now, in Boreale and Sangiorgi's semantics, we have:

  $$\nu a(Q'_1) \xrightarrow{\zeta'_1} Q''_1 \quad \text{and} \quad Q_2 \xrightarrow{\zeta''_1} Q'_2$$

  We can apply the rule (BS-Close) on the process $\nu a(Q'_1) \mid Q_2$, and we have:

  $$\nu a(Q'_1) \mid Q_2 \xrightarrow{\tau} \nu a(Q''_1 \mid Q'_2\{^a/_x\}) = A_1$$

  with $\varphi(A_1) = \nu a(Q''_1 \mid Q'_2\{^a/_x\})$ as desired.
- $\nu a_{\emptyset_*}(\mathbf{P}')$, where $\varphi(\nu a_{\emptyset_*}(\mathbf{P}')) = \nu a(P')$. The rules that can be applied in the framework are (Res) and (Open). We show the case when the rule (Open) is used; the other case is similar.
  If (Open) is applied on the process $\nu a_{\emptyset_*}(\mathbf{P}')$, the executed action extrudes name a, and we have:

  $$\nu a_{\emptyset_*}(\mathbf{P}') \xrightarrow{(i_1,\{*\},*):\overline{b}\langle\nu a_{\emptyset_*}\rangle} \nu a_{\{i_1\}_{i_1}}(Y_1) = X_1$$

  with the premise $\mathbf{P}' \xrightarrow{(i_1,\{*\},*):\overline{b}a} Y_1$ where $\gamma((i_1, \{*\}, *) : \overline{b}a) = i_1 : \overline{b}a$ and $\varphi(Y_1) = P_1$.

  From the inductive hypothesis, we have $\varphi(\mathbf{P}') = P'$; hence in Boreale and Sangiorgi's semantics we have $P' \xrightarrow{i_1:\overline{b}a} \{i_1\} :: P_1$. Now, we can apply the rule (BS-Open) on the process $\nu a(P')$, and obtain:

  $$\nu a(P') \xrightarrow{i_1:\overline{b}\langle\nu a\rangle} \{i_1\} :: P_1 = A_1$$

  where $\lambda(A_1) = \lambda(\{i_1\} :: P_1) = P_1$ as desired.

If $X_n^{K_{Fn}}$ is a reversible process with past, then we have the following cases:

- $X_n^{K_{Fn}} = \mathtt{H}[X]$; In the framework, we can apply rules (Out2) and (In2), depending whether the executing action is the input or the output. Let us consider the case when the executing action is the output action and the applied rule is (Out2); the other case is similar. Then we have:

  $$\mathtt{H}[X] \xrightarrow{(i_{n+1},K_{n+1},j_{n+1}):\alpha} \mathtt{H}[Y]^{K'_F \cup K_F}$$

  with the premise $X \xrightarrow{(i_{n+1},K_{n+1},j_{n+1}):\alpha} Y^{K_F}$. The set of causes K'_F represents the causes of the executed action belonging to the history context $\mathtt{H}$, while K_F are causes belonging to the process X. Hence, we have $K_{Fn+1} = K'_F \cup K_F$.
  From the inductive hypothesis, we know that $\varphi(\mathtt{H}[X]) = \lambda(A_n)$, where $A_n = \mathtt{K} :: A$ and $\varphi(X) = \lambda(A)$. Therefore, there exists an action $\zeta_{n+1} = i_{n+1} : \alpha$ such that $\gamma((i_{n+1}, K_{n+1}, j_{n+1}) : \alpha) = \zeta_{n+1}$ and $A \xrightarrow[\mathtt{K}]{i_{n+1}:\alpha} A'$.
  Now we can apply the rule (BS-Cau) in Boreale and Sangiorgi's semantics, on process $\mathtt{K} :: A$ with the given premise, and obtain:

  $$A_n = \mathtt{K}' :: A \xrightarrow[\mathtt{K}',\mathtt{K}]{i_{n+1}:\alpha} \mathtt{K}' :: A' = A_{n+1}$$

  where $\mathtt{K}_{n+1} = \mathtt{K}' \cup \mathtt{K}$.
  To prove that actions ζ_{n+1} and μ_{n+1} are executed on the same part of the π-calculus process $\varphi(\mathtt{H}[X]) = \lambda(\mathtt{K}' :: A)$, which implies $\varphi(\mathtt{H}[Y]^{K'_F \cup K_F}) = \lambda(\mathtt{K}' :: A')$, we need to prove that the actions had the same structural cause sets, i.e. $\mathtt{Rem}(K_{Fn+1}) = \mathtt{K}_{n+1}$. To do so, we should look at the action μ_n because it is the last action that can influence the cause set K_{Fn+1}. There are three main cases:
  - action μ_n is the direct structural cause of the action μ_{n+1} and it is visible action (it happened on the same component in the parallel composition, immediately before the action μ_{n+1}). Then we have $K_{Fn+1} = K'_F \cup K_F = K_{Fn} \cup \{i_n\}$. By inductive hypothesis, we have that there exist $\mathtt{K}_n, \zeta_n \in s_{BS}$ such that $\gamma(\mu_n) = \zeta_n$ and $\mathtt{Rem}(K_{Fn}) = \mathtt{K}_n$. Since μ_n is visible, ζ_n needs to be too and it is identified with the key i_n. The action ζ_n is the direct structural cause of the action ζ_{n+1}; hence, we have $\mathtt{K}_{n+1} = \mathtt{K}_n \cup \{i_n\}$ as desired.
  - action μ_n is the direct structural cause of the action μ_{n+1} and it is a silent action. In the framework we have $K_{Fn+1} = K'_F \cup K_F = K_{Fn} \cup \{i_n\}$, since the silent action is identified with the key i_n. Cause set K_{Fn} of the τ action contains cause sets of the communicating actions (the input and the output ones). Hence, there exist K_{Fj}, $K_{Fh} \in s_F$ such that $K_{Fn} = K_{Fj} \cup K_{Fh} = K'_F \cup K_F \setminus \{i_n\}$. By inductive hypothesis, we know that there exist $\mathtt{K}_j$, $\mathtt{K}_h \in s_{BS}$ such that $\mathtt{Rem}(K_{Fj}) = \mathtt{K}_j$ and $\mathtt{Rem}(K_{Fh}) = \mathtt{K}_h$. Since silent actions on causal processes just merge two cause sets, we have $\mathtt{K}_{n+1} = \mathtt{K}_j \cup \mathtt{K}_h$. Method $\mathtt{Rem}$ (Definition 41) removes keys belonging to τ actions; hence we have $\mathtt{Rem}(K_{Fn} \cup \{i_n\}) = \mathtt{Rem}(K_{Fj} \cup K_{Fh}) = \mathtt{K}_j \cup \mathtt{K}_h = \mathtt{K}_{n+1}$ as desired.
  - action μ_n is not the direct cause of the action μ_{n+1}; then μ_n happened on a different component in the parallel composition from the action μ_{n+1}. In this case, there exist $K_{Fh}, \mu_h \in s_F$ where μ_h is the last action on the same component in the parallel composition as μ_{n+1}. Hence, action μ_h is the direct cause of the action μ_{n+1} and we have $K_{Fn+1} = K_{Fh} \cup \{i_h\}$. Now we have the same reasoning as in the first case.
- $X_n^{K_{Fn}} = X_{n1} \mid X_{n2}$; In the framework, applicable rules are (Par), (Close) and (Com). Let us consider the case when the rule (Close) is used; the remaining cases are similar. Now we have that one of the processes in parallel extrudes a bound name, say X_{n1}. Then $X_{n1} = \nu a_{\emptyset_*}(X'_{n1})$. By applying the rule (Close) on the process $\nu a_{\emptyset_*}(X'_{n1}) \mid X_{n2}$, we obtain

  $$\nu a_{\emptyset_*}(X'_{n1}) \mid X_{n2} \xrightarrow{(i_{n+1},\{*\},*):\tau} \nu a_{\emptyset_*}((Y_{n1}^{K'_{Fn+1}})_{\#i_{n+1}} \mid Y_{n2}^{K''_{Fn+1}}\{^{a^{i_{n+1}}}/_x\})$$

  with premises $\nu a_{\emptyset_*}(X'_{n1}) \xrightarrow{(i_{n+1},K'_{n+1},j'_{n+1}):\overline{b}(\nu a_{\emptyset_*})} \nu a_{\{i_{n+1}\}_{i_{n+1}}}(Y_{n1}^{K'_{Fn+1}})$ and $X_{n2} \xrightarrow{(i_{n+1},K''_{n+1},j''_{n+1}):b(x)} Y_{n2}^{K''_{Fn+1}}$.
  The condition on the rule (Close) is satisfied since by definition of the set K when data structure Γ_w is used, we have $* \in K'_{n+1}$ and $* \in K''_{n+1}$, which makes $K'_{n+1} =_* j''_{n+1}$ and $K''_{n+1} =_* j'_{n+1}$ true. By the definition of the function $\varphi()$, we have:

  $$\varphi(\nu a_{\emptyset_*}((Y_{n1}^{K'_{Fn+1}})_{\#i_{n+1}} \mid Y_{n2}^{K''_{Fn+1}}\{^{a^{i_{n+1}}}/_x\})) =$$
  $$\nu a \varphi((Y_{n1}^{K'_{Fn+1}})_{\#i_{n+1}} \mid Y_{n2}^{K''_{Fn+1}}\{^{a^{i_{n+1}}}/_x\}))$$

  Since the operator $\#i_{n+1}$ is applied on the memories of the type νa_{Γ_w} when $\mathtt{empty}(\Gamma_w) \neq \mathtt{true}$ it is deleted with the function φ. Therefore, we have:

  $$\nu a \varphi((Y_{n1}^{K'_{Fn+1}})_{\#i_{n+1}} \mid Y_{n2}^{K''_{Fn+1}}\{^{a^{i_{n+1}}}/_x\})) =$$
  $$\nu a \varphi(Y_{n1}^{K'_{Fn+1}})) \mid \varphi(Y_{n2}^{K''_{Fn+1}}\{^{a^{i_{n+1}}}/_x\})$$

From the inductive hypothesis we know that $\varphi(\nu a_{\emptyset_*}(X'_{n1})) = \lambda(\nu a A_{n1})$ and $\varphi(X_{n2}) = \lambda(A_{n2})$. Hence, there exist $\zeta'_n = i_{n+1} : \overline{b}\langle \nu a \rangle$ and $\zeta''_n = i_{n+1} : b(x)$ such that $\nu a A_{n1} \xrightarrow[\mathtt{K}'_{n+1}]{i_{n+1}:\overline{b}\langle \nu a\rangle} A'_{n1}$ and $A_{n2} \xrightarrow[\mathtt{K}''_{n+1}]{i_{n+1}:b(x)} A'_{n2}$. With the given premises, we can apply the rule (BS-Close) on the process $\nu a A_{n1} \mid A_{n2}$ and obtain:

$$\nu a(A_{n1}) \mid A_{n2} \xrightarrow{\tau} \nu a(A'_{n1}[i_{n+1} \rightsquigarrow \mathtt{K}''_{n+1}] \mid A'_{n2}\{a/x\}[i_{n+1} \rightsquigarrow \mathtt{K}'_{n+1}]) = A_{n+1}$$

Since function λ deletes cause sets we have:

$$\lambda(A_{n+1}) = \nu a \lambda((A'_{n1} \mid A'_{n2}\{a/x\})) = \nu a \lambda(A'_{n1}) \mid \lambda(A'_{n2}\{a/x\})$$

Now we need to prove $\mathtt{Rem}(K'_{Fn+1}) = \mathtt{K}'_{n+1}$ and $\mathtt{Rem}(K''_{Fn+1}) = \mathtt{K}''_{n+1}$. To do that, we should look at the action μ_n because it is the last action that can influence the cause sets K'_{Fn+1} and K''_{Fn+1}. There are three main cases:

- action μ_n is the direct structural cause of the action μ_{n+1} and it is a visible action (it happened in the same component in the parallel composition, immediately before the action μ_{n+1}). Then we have three different cases: $K'_{Fn+1} = K_{Fn} \cup \{i_n\}$ and $K''_{Fn+1} \in s_F$; $K''_{Fn+1} = K_{Fn} \cup \{i_n\}$ and $K'_{Fn+1} \in s_F$ or $K'_{Fn+1} = K''_{Fn+1} = K_{Fn} \cup \{i_n\}$.
  Let us consider the first case, when $K'_{Fn+1} = K_{Fn} \cup \{i_n\}$ and $K''_{Fn+1} \in s_F$; the other cases are similar. From the inductive hypothesis we have $\mathtt{Rem}(K''_{Fn+1}) = \mathtt{K}''_{n+1}$ and there exist $\mathtt{K}_n, \zeta_n \in s_{BS}$ such that: $\gamma(\mu_n) = \zeta_n$ and $\mathtt{Rem}(K_{Fn}) = \mathtt{K}_n$. The action ζ_n is identified with a key i_n and it is a visible action and direct structural cause of the action ζ_{n+1}, where $\gamma(\mu_{n+1}) = \zeta_{n+1}$; therefore, we have $\mathtt{K}'_{n+1} = \mathtt{K}_n \cup \{i_n\}$. The method Rem (Definition 41) does not remove keys of the visible actions and we have $\mathtt{Rem}(K_{Fn} \cup \{i_n\}) = \mathtt{K}_n \cup \{i_n\} = \mathtt{K}'_{n+1}$ as desired.
- action μ_n is the direct structural cause of the action μ_{n+1} and it is a silent action. We have similar reasoning as in the previous case, with the difference that method Rem removes keys of the silent actions; hence key i_n will be deleted from the cause set K'_{Fn+1}.
- action μ_n is not the direct cause of the action μ_{n+1}; then μ_n happened in a different component in the parallel composition from the actions μ'_{n+1} and μ''_{n+1}. In this case, there exist $K_{Fh}, K_{Fl}, \mu_h, \mu_l \in s_F$ where μ_h and μ_l are the last actions on the same components in the parallel composition as μ'_{n+1} and μ''_{n+1}. Hence, action μ_h is the direct cause of the action μ'_{n+1} and action μ_l is the direct cause of the action μ''_{n+1}. Then we have $K'_{Fn+1} = K_{Fh} \cup \{i_h\}$ and $K''_{Fn+1} = K_{Fl} \cup \{i_l\}$. Now we have the same reasoning as in the first case.

By having $\mathtt{Rem}(K'_{Fn+1}) = \mathtt{K}'_{n+1}$ and $\mathtt{Rem}(K''_{Fn+1}) = \mathtt{K}''_{n+1}$, we can conclude that $\varphi(Y_{n1}^{K'_{Fn+1}}) = \lambda(A'_{n1})$ and $\varphi(Y_{n2}^{K''_{Fn+1}}) = \lambda(A'_{n2})$ which concludes our case.

- $X_n^{K_{Fn}} = \nu a_{\Gamma_w}(X)$; In the framework, applicable rules are (Res), (Open) and (Cause Ref). Let us consider the case when the rule (Cause Ref) is applied. In this case, the executing action has the name a in the subject position and $\mathtt{empty}(\Gamma_w) \neq \mathtt{true}$. We have:

$$\nu a_{\Gamma_w}(X) \xrightarrow{(i_{n+1}, K'_{n+1}, j_{n+1}):\overline{a}b} \nu a_{\Gamma_w}(Y^{K_{Fn+1}})$$

with premise $X \xrightarrow{(i_{n+1}, K_{n+1}, j_{n+1}):\overline{a}b} Y^{K_{Fn+1}}$ where $K'_{n+1} = K_{n+1} \cup \{w\}$.
Since $\mathtt{empty}(\Gamma_w) \neq true$ name a is free in the process X and νa_{Γ_w} is discarded by the function $\varphi()$; hence, we have $\varphi(\nu a_{\Gamma_w}(Y^{K_{Fn+1}})) = \varphi(Y^{K_{Fn+1}})$. From the inductive hypothesis we have $\varphi(\nu a_{\Gamma_w}(X)) = \varphi(X) = \lambda(A_n)$ where a is free name in A_n and there exists $\zeta_{n+1} = i_{n+1} : \overline{a}b$ such that:

$$A_n \xrightarrow[\mathtt{K}_{n+1}]{i_{n+1}:\overline{a}b} A_{n+1}$$

To prove $\mathtt{Rem}(K_{Fn+1}) = \mathtt{K}_{n+1}$ which would imply $\varphi(Y^{K_{Fn+1}}) = \lambda(A_{n+1})$ we need to reason on the action μ_n and proceed similarly to the cases above. □

References

[1] C. Bennett, Logical reversibility of computation, IBM J. Res. Develop. 17 (6) (1973), https://doi.org/10.1147/rd.176.0525.
[2] G. Bacci, V. Danos, O. Kammar, On the statistical thermodynamics of reversible communicating processes, in: CALCO 2011, in: LNCS, vol. 6859, Springer, 2011, pp. 1–18.
[3] V. Danos, J. Krivine, Formal molecular biology done in CCS-R, Electron. Notes Theor. Comput. Sci. 180 (3) (2007) 31–49, https://doi.org/10.1016/j.entcs.2004.01.040.
[4] I. Phillips, I. Ulidowski, S. Yuen, Modelling of bonding with processes and events, in: Reversible Computation - RC 2013, in: LNCS, vol. 7948, Springer, 2013, pp. 141–154.
[5] M. Zelkowitz, Reversible execution, Commun. ACM 16 (9) (1973) 566, https://doi.org/10.1145/362342.362360.
[6] E. Giachino, I. Lanese, C. Mezzina, Causal-consistent reversible debugging, in: FASE, in: LNCS, vol. 8411, Springer, 2014, pp. 370–384.
[7] I. Lanese, N. Nishida, A. Palacios, G. Vidal, CauDEr: a causal-consistent reversible debugger for Erlang, in: J.P. Gallagher, M. Sulzmann (Eds.), Functional and Logic Programming - 14th International Symposium, FLOPS, in: LNCS, vol. 10818, Springer, 2018, pp. 247–263.
[8] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information, 10th anniversary edition, Cambridge University Press, 2016.
[9] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Dependable Secure Comput. 1 (1) (2004) 11–33, https://doi.org/10.1109/TDSC.2004.2.

[10] V. Danos, J. Krivine, Transactions in RCCS, in: CONCUR, San Francisco, CA, USA, August 23-26, 2005, pp. 398–412.
[11] I. Lanese, C.A. Mezzina, A. Schmitt, J. Stefani, Controlling reversibility in higher-order pi, in: CONCUR - Concurrency Theory - 22nd International Conference, in: LNCS, vol. 6901, Springer, 2011, pp. 297–311.
[12] D. Medic, C.A. Mezzina, I. Phillips, N. Yoshida, Towards a formal account for software transactional memory, in: I. Lanese, M. Rawski (Eds.), Reversible Computation - 12th International Conference, RC, in: Lecture Notes in Computer Science, vol. 12227, Springer, 2020, pp. 255–263.
[13] M. Vassor, J. Stefani, Checkpoint/rollback vs causally-consistent reversibility, in: Reversible Computation - 10th International Conference, RC, in: LNCS, vol. 11106, Springer, 2018, pp. 286–303.
[14] V. Danos, J. Krivine, Reversible communicating systems, in: CONCUR 2004, in: LNCS, vol. 3170, Springer, 2004, pp. 292–307.
[15] R. Milner, A Calculus of Communicating Systems, LNCS, vol. 92, Springer, 1980.
[16] D. Medic, C.A. Mezzina, Static vs dynamic reversibility in CCS, in: Reversible Computation RC 2016, in: LNCS, vol. 9720, Springer, 2016, pp. 36–51.
[17] I. Phillips, I. Ulidowski, Reversing algebraic process calculi, J. Log. Algebraic Program. 73 (1–2) (2007) 70–96, https://doi.org/10.1016/j.jlap.2006.11.002.
[18] I. Lanese, D. Medić, C.A. Mezzina, Static versus dynamic reversibility in CCS, Acta Inform. (2019), https://doi.org/10.1007/s00236-019-00346-6.
[19] P. Degano, C. Priami, Non-interleaving semantics for mobile processes, Theor. Comput. Sci. 216 (1–2) (1999) 237–270, https://doi.org/10.1016/S0304-3975(99)80003-6.
[20] M. Boreale, D. Sangiorgi, A fully abstract semantics for causality in the π-calculus, Acta Inform. 35 (5) (1998) 353–400, https://doi.org/10.1007/s002360050124.
[21] N. Busi, R. Gorrieri, A Petri net semantics for pi-calculus, in: CONCUR, Proceedings, Philadelphia, PA, USA, August 21-24, 1995, 1995, pp. 145–159.
[22] S. Crafa, D. Varacca, N. Yoshida, Event structure semantics of parallel extrusion in the pi-calculus, in: FOSSACS 2012, in: LNCS, vol. 7213, Springer, 2012, pp. 225–239.
[23] I. Cristescu, J. Krivine, D. Varacca, A compositional semantics for the reversible π-calculus, in: LICS 2013, 2013, pp. 388–397.
[24] I. Cristescu, J. Krivine, D. Varacca, Rigid families for CCS and the π-calculus, in: ICTAC, in: LNCS, vol. 9399, Springer, 2015, pp. 223–240.
[25] I. Lanese, C. Mezzina, J.-B. Stefani, Reversibility in the higher-order π-calculus, Theor. Comput. Sci. 625 (2016) 25–84, https://doi.org/10.1016/j.tcs.2016.02.019.
[26] I. Lanese, C.A. Mezzina, J. Stefani, Reversing higher-order pi, in: CONCUR 2010 - Concurrency Theory, 21th International Conference, in: LNCS, vol. 6269, Springer, 2010, pp. 478–493.
[27] D. Medic, C. Mezzina, 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, 2017, pp. 121–125.
[28] L. Aceto, GSOS and finite labelled transition systems, Theor. Comput. Sci. 131 (1) (1994) 181–195, https://doi.org/10.1016/0304-3975(94)90094-9.
[29] D. Medic, C.A. Mezzina, I. Phillips, N. Yoshida, A parametric framework for reversible pi-calculi, in: Proceedings Combined 25th International Workshop on Expressiveness in Concurrency and 15th Workshop on Structural Operational Semantics and 15th Workshop on Structural Operational Semantics, EXPRESS/SOS, in: EPTCS, vol. 276, 2018, pp. 87–103.
[30] D. Sangiorgi, D. Walker, The Pi-Calculus - a Theory of Mobile Processes, Cambridge Univ. Press, 2001.
[31] C.A. Mezzina, On reversibility and broadcast, in: J. Kari, I. Ulidowski (Eds.), Reversible Computation - 10th International Conference, RC, in: LNCS, vol. 11106, Springer, 2018, pp. 67–83.
[32] J. Lévy, An algebraic interpretation of the $\lambda\beta$k-calculus; and an application of a labelled λ-calculus, Theor. Comput. Sci. 2 (1) (1976) 97–114, https://doi.org/10.1016/0304-3975(76)90009-8.
[33] G. Boudol, I. Castellani, 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, in: LNCS, vol. 354, Springer, 1988, pp. 411–427.
[34] F. Harary, Graph Theory, Addison-Wesley, 1991.
[35] I. Phillips, I. Ulidowski, A hierarchy of reverse bisimulations on stable configuration structures, Math. Struct. Comput. Sci. 22 (2) (2012) 333–372, https://doi.org/10.1017/S0960129511000429.
[36] R. Perera, J. Cheney, Proof-relevant π-calculus: a constructive account of concurrency and causality, Math. Struct. Comput. Sci. (2017) 1–37, https://doi.org/10.1017/S096012951700010X.
[37] T. Hildebrandt, C. Johansen, H. Normann, A stable non-interleaving early operational semantics for the pi-calculus, in: LATA, in: LNCS, vol. 10168, 2017, pp. 51–63.
[38] M. Cimini, M.R. Mousavi, M.A. Reniers, M.J. Gabbay, Nominal SOS, Electron. Notes Theor. Comput. Sci. 286 (2012) 103–116, https://doi.org/10.1016/j.entcs.2012.08.008.
[39] J. Bengtson, M. Johansson, J. Parrow, B. Victor, Psi-calculi: a framework for mobile processes with nominal data and logic, Log. Methods Comput. Sci. 7 (1) (2011), https://doi.org/10.2168/LMCS-7(1:11)2011.