Rigid Families for the Reversible π-calculus*

Ioana Cristescu[1], Jean Krivine[2], and Daniele Varacca[3]

[1] Harvard Medical School
ioana_cristescu@hms.harvard.edu
[2] IRIF - Équipe PPS - Université Paris Diderot
jean.krivine@pps.univ-paris-diderot.fr
[3] LACL - Université Paris Est - Créteil
daniele.varacca@u-pec.fr

Abstract. Rigid families, a causal model for concurrency based on configuration structures, can interpret CCS and the π-calculus. However, it is also a causal model suited for reversible calculi. In this paper we use rigid families to give a denotational representation to the reversible π-calculus. The reversible π-calculus defines a causal semantics for the π-calculus as well. We discuss the difference in the two causal representations, in rigid families and in the reversible π-calculus.

Reversible calculi allow one to backtrack computation events as long as the causal order between events is respected: one cannot undo the cause before the effect. Moreover, in reversible operational semantics [1,2,3] backtracking is also *maximally concurrent*: any forward execution is a valid backward path.

In this paper we are interested in an extensional or non-interleaving representation of reversible calculi and in particular the reversible π-calculus [4]. In configuration structures [5], one uses sets of events, called *configurations*, to represent reachable states of computations. The extensional behaviour of a process is represented as a domain the elements of which are configurations ordered by set inclusion. It is noteworthy that such non-interleaving models are implicitly reversible as the inclusion between configurations dictates the allowed forward and backward transitions.

Rigid families [6,7] is a non interleaving model that is close to configuration structures. In this model, configurations are additionally equipped with a partial order on events, called *precedence*, that is a *temporal* relation between computation events that is made during the run of a process. When two events are not ordered by precedence one can see them as having occurred either simultaneously or in such a way that no common clock can be used to compare them. More traditional causality and concurrency relations are derivable from precedence.

Though there have been several non interleaving models proposed for process algebra, one may argue that rigid families is the most suited formalism to study causal semantics of the π-calculus [7] where two notions of causality coexist. A

* This work was partly supported by the ANR-11-INSE-0007 REVER and by the ICT COST Action IC1405.

first type of causality, induced by the prefix operator, is called *structural*. The second type of causality is *contextual* and induced by the mechanism of scope extrusion. We illustrate these two notions with some examples.

The process $\overline{b}\langle a\rangle.a(d)$ is an example for structural causality: the communication on channel b occurs always before the one on channel a. The same situation, this time due to contextual causality, occurs in $\nu a(\overline{b}\langle a\rangle \mid a(d))$, where one cannot use channel a to communicate with the environment as a is private. However, a can be sent on channel b. The output of a private name a to the context is traditionally called *scope extrusion*, and we say that the event that sends a on channel b is an *extruder* of a. Once the context receives a it can use it for future communications. However, contextual causality can also be *disjunctive*. In the following process $\nu a(\overline{b}\langle a\rangle \mid \overline{c}\langle a\rangle \mid a(d))$ the name is a is private but there are two possible extruders of a: one on channel b and the other on channel c. Only one of the two is responsible for the sending of a to the context, but there are executions in which we cannot tell which of the two is the cause. In the trace

$$\nu a(\overline{b}\langle a\rangle \mid \overline{c}\langle a\rangle \mid a(d)) \xrightarrow{\overline{b}\langle a\rangle} \xrightarrow{\overline{c}\langle a\rangle} \xrightarrow{a} 0$$

either the output on channel b or the one on channel c is the cause. If the execution backtracks the communication on b, while b is the cause, we are reaching an inconsistent state, where the cause is undone before the effect. Thus, detecting which extruder is the cause of the communication on channel a is important for a correct backtracking mechanism.

Rigid families can express both structural and contextual causality and disambiguate examples as the one above. It was therefore postulated (see Ref. [4]) that in addition of being a suitable causal model for the classical π-calculus, rigid families should naturally represent reversible computations of the π-calculus [4]. This is what this paper investigates.

Outline. This paper is as self contained as space permits although familiarity with the π-calculus [8] is assumed. In section 1 we briefly introduce the reversible π-calculus. In section 2 we recall how rigid families are defined and, following Ref. [4], show how one can interpret the π-calculus. In section 3 we present the first contribution of this paper, which is the interpretation of *reversible* π-processes in terms of rigid families. Both the reversible π-calculus and the rigid families provide a causal semantics for the π-calculus. In section 4 we compare the two. The second contribution of the paper is presented section 5, where we show an operational correspondence between reversible π processes and their encoding in rigid families. We conclude with section 6.

1 The reversible π-calculus

In this section we give a brief summary of the syntax and the semantics of the reversible π-calculus. We let the reader refer to Ref.[4] for more details.

Rπ processes use memories, added on top of simple π processes to record past computations. Every entry in a memory, called an event, can be used to

backtrack. We use the same notations as in Ref.[4]. Denote I the set of event identifiers, with a distinguished symbol $* \in \mathsf{I}$. Let i, j, k range over elements of I and Δ, Γ range over subsets of I. Terms are built according to the following grammar:

$$
\begin{aligned}
P, Q &::= 0 \parallel \pi.P \parallel (P \mid Q) \parallel \nu a(P) && (\pi \text{ processes}) \\
R, S &::= 0 \parallel m \rhd P \parallel (R \mid S) \parallel \nu a_{\Gamma}(R) && (\mathrm{R}\pi \text{ processes}) \\
m &::= \emptyset \parallel \Upsilon .m \parallel e.m && (\text{memory stacks}) \\
e &::= \langle i, k, \alpha \rangle && (\text{memory events}) \\
\alpha &::= \overline{b}\langle a \rangle \parallel b[\star/c] \parallel b[a/c] && (\text{event labels})
\end{aligned}
$$

where $\pi ::= b(c) \mid \overline{b}\langle a \rangle \mid \tau$ denotes π prefixes.

We store two types of information in the memory. The first is a fork symbol Υ, which distributes memory stacks whenever processes are forking. Secondly, events are stored in the memory as triplets of the form $\langle i, k, \alpha \rangle$. For any event $e = \langle i, k, \alpha \rangle$, we say that $i \in \mathsf{I} - \{*\}$ is the *identifier* of e, $k \in \mathsf{I}$ is the identifier of its *contextual cause* and α its *label*. The label of an event is similar to π prefixes, except that we also store the substitution in case of a synchronisation.

For all events $e = \langle i, k, \alpha \rangle$, we define $id(e) = i$, $c(e) = k$ and $\ell(e) = \alpha$.

Some syntactically correct processes are not correct semantically. Henceforth we only consider semantically correct, reachable processes.

Definition 1 (Relations on events [4, Definition 2.2.]). *Let R be a process, we define the following relations on events of R.*

- Structural causal relation*: $e' \sqsubset_R e$ if there exists $m \in R$ such that $m = m_2.e.m_1.e'.m_0$ for some (possibly empty) m_2, m_1, m_0. Structural causality is propagated by synchronisations, thus we use $\sqsubset_R^{\star}$ to denote the transitive closure of $\sqsubset_R$.*
- Contextual causal relation*: $e' \ll_R e$ if $c(e) = id(e')$.*
- Instantiation relation*: $e' \rightsquigarrow_R e$ if $e' \sqsubset_R e$ and $\ell(e') = b[a/c]$, for some name a, b, c and c is in subject position in $\ell(e)$. Furthermore for all memories m, we write $\mathsf{inst}_m(c) = i$ if there is an event of the form $\langle i, k, b[a/c] \rangle$ in m that instantiates c. Note that there is at most one such event in m. If no such event exists in m we write $\mathsf{inst}_m(c) = *$.*

For any events $e \in R$ and $e' \in R$ such that $id(e) = i$ and $id(e') = j$, we use the overloaded notations $i \sqsubset_R j$ or $i \ll_R j$, if e and e' are in the corresponding relation. Define in a similar manner $i \rightsquigarrow_R j$ iff $e \rightsquigarrow_R e'$ for $id(e) = i$, $id(e') = j$ and $\star \rightsquigarrow_R j$, for any $j \in \mathsf{I}$.

Definition 2 (Structural congruence [4, Section II-A]). *Structural congruence on monitored processes is the smallest equivalence relation generated by*

the following rules:

$$\frac{P \equiv Q}{m \rhd P \equiv m \rhd Q} \qquad (\pi \text{ congruence})$$

$$m \rhd (P|Q) \equiv (\Upsilon.m \rhd P| \ \Upsilon \ .m \rhd Q) \qquad \text{(distribution memory)}$$

$$m \rhd \nu a(P) \equiv_m \nu a_\emptyset(m \rhd P) \text{ with } a \notin m \qquad \text{(scope of restriction)}$$

The third rule in the definition above rewrites π-calculus restrictions into Rπ restrictions. An Rπ restriction is indexed by a set $\Gamma \subset \mathsf{I}$ (initially empty) and behaves as a classical restriction only when $\Gamma = \emptyset$. It will be used to keep track of past variable scope whenever $\Gamma \neq \emptyset$.

Definition 3 (Bound, free and liberated names [4, Section II-A]). *Free and liberated names are defined inductively on the structure of processes ($f(a)$ denotes either* $\mathrm{fn}(a)$ *or* $\mathrm{lib}(a)$ *whenever the distinction is irrelevant):*

$$\begin{array}{lll} f(\nu a_\emptyset R) = f(R) - \{a\} & (\Gamma \neq \emptyset) & f(\nu a_\Gamma R) = f(R) \cup \{a\} \\ f(R \mid S) = f(R) \cup f(S) & & \mathrm{fn}(\bar{b}\langle a \rangle.P) = \mathrm{fn}(P) \cup \{a, b\} \\ \mathrm{fn}(m \rhd P) = \mathrm{nm}(m) \cup \mathrm{fn}(P) & & \mathrm{lib}(m \rhd P) = \emptyset \\ \mathrm{fn}(b(a).P) = \{b\} \cup (\mathrm{fn}(P) - \{a\}) & & \end{array}$$

with $\mathrm{nm}(m)$ *being all the names occurring in the memory m. All liberated names are free. As usual, names which are not free in R are called bound.*

1.1 Name substitution

In the late LTS of the π-calculus substitutions are applied only upon synchronisation. However, in the case of reversible processes, we do not apply a substitution directly on the process as it can lead to cases where the backtracking is ambiguous. Instead we use *explicit* substitutions: the substitutions are recorded in the memory and are applied only when needed. Therefore, after a synchronisation we *update* the memory in order to store the new substitution. A process communicating on a liberated channel, has to make an assumption on the identity of the event that made the channel public (via an *extrusion*), called its *contextual cause*. The initial assumption can be made more precise while more structure of the process is revealed by the LTS, thus we also need to update the contextual cause.

Definition 4 (Memory updates [4, Definition 2.1.]). *The* synchronisation update*, denoted by $R_{[a/c]@i}$, replaces the partial substitution $[\star/c]$ with the complete substitution $[a/c]$ at the event identified by $i \in \mathcal{I} - \{*\}$, it is defined as:*

$$\begin{array}{ll} (R \mid S)_{[a/c]@i} & = R_{[a/c]@i} \mid S_{[a/c]@i} \\ (\nu a'_\Gamma R)_{[a/c]@i} & = \nu a'_\Gamma (R_{[a/c]@i}) \\ (\langle i, \_, b[\star/c]\rangle.m \rhd P)_{[a/c]@i} & = \langle i, \_, b[a/c]\rangle.m \rhd P \\ (m \rhd P)_{[a/c]@i} & = m \rhd P \quad \textit{otherwise} \end{array}$$

The contextual cause update*, denoted by $R_{[k/k']@i}$ proceeds similarly but substitutes the old cause k' for a new one:*

$$
\begin{array}{ll}
(R \mid S)_{[k/k']@i} & = R_{[k/k']@i} \mid S_{[k/k']@i} \\
(\nu a_\Gamma R)_{[k/k']@i} & = \nu a_\Gamma (R_{[k/k']@i}) \\
(\langle i, k', \_ \rangle.m \rhd P)_{[k/k']@i} & = \langle i, k, \_ \rangle.m \rhd P \\
(m \rhd P)_{[k/k']@i} & = m \rhd P \quad \textit{otherwise}
\end{array}
$$

The substitutions on a variable are applied only when a process uses it for a communication. The *public name* is thus a name on which all the substitutions were applied.

Definition 5 (Public label [4, Definition 2.3.]). *For all process of the form $m \rhd \pi.P$ let $m[\pi]$ be the* public label *of π. It is defined by lexicographical induction on the pair (π,m):*

$$
\begin{array}{lllll}
\emptyset[a] & = a & & m[b(c)] & = m[b](c) \\
m[\bar{b}\langle a \rangle] & = \overline{m[b]}\langle m[a] \rangle & & (\langle i, k, b[c/a] \rangle.m)[a] & = c \\
(\langle i, k, b[\star/a] \rangle.m)[a] & = a & & (\Upsilon.m)[a] & = m[a] \\
(e.m)[a] & = m[a] \ \textit{otherwise} & & &
\end{array}
$$

We recall the following notations from Ref.[4]: we write $m \in R$ if there exists a context $C[\bullet]$ such that $R = C[m \rhd P]$. Similarly we write $e \in R$ when there is $m \in R$ such that $m = m_1.e.m_0$ for some (possibly empty) m_1 and m_0. Finally for all $i \in \mathsf{I}$ we write $i \in R$ if there exists $e \in R$ such that $id(e) = i$ or $c(e) = i$.

1.2 The LTS

The label ζ of a transition $t : R \xrightarrow{\zeta} S$ is a quadruple of the form $(i, j, k) : \alpha$ where $i \in \mathcal{I} - \{*\}$ is the *identifier* of t, $j \in \mathcal{I}$ is the instantiator of i and $k \in \mathcal{I}$ is the contextual cause of i. The labels α are built on the following grammar:

$$\alpha ::= b(c) \| \bar{b}\langle a \rangle \| \bar{b}(\nu a_\Gamma)$$

where $\bar{b}(\nu a_\Gamma)$ corresponds to the bound output of the π-calculus, whenever $\Gamma = \emptyset$, and otherwise corresponds to a free output, decorated with a set of event identifiers. We extend the subj and obj functions from the π-calculus in a straightforward manner in order to include labels $\bar{b}(\nu a_\Gamma)$.

The labelled transition system of Rπ can be divided into positive rules, presented in Figure 1, and negative rules, derived from the positive ones by simply inverting the rules and keeping the side conditions invariant. We denote $\xrightarrow{\zeta}$ the *positive* transitions and $\xrightarrow{\zeta^-}$ the *backward* ones. The notation $i =_* j$, for $i, j \in I$, stands for $* \in \{i, j\}$ or $i = j$.

Note that the complete positive LTS contains also the symmetrical rules for the Com+, Close+ and Par+ rules with respect to the $\mid$ operator.

2 Rigid families for the π-calculus

In this section we introduce the rigid families [9] and recall how they can be used as a model for the π-calculus [7]. We first introduce unlabelled rigid families, and then, using the set of labels of the π-calculus, we define the labelled rigid families.

$$\textsc{In}+\frac{i \notin m \quad j = \mathsf{inst}_m(b)}{m \rhd b(c).P \xrightarrow{(i,j,*):m[b(c)]} \langle i,*,b[\star/c]\rangle.m \rhd P}$$

$$\textsc{Out}+\frac{i \notin m \quad j = \mathsf{inst}_m(b)}{m \rhd \overline{b}\langle a\rangle.P \xrightarrow{(i,j,*):m[\overline{b}\langle a\rangle]} \langle i,*,\overline{b}\langle a\rangle\rangle.m \rhd P}$$

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

$$\textsc{Cause ref}+\frac{R \xrightarrow{(i,j,k):\alpha} R' \ a \in subj(\alpha)}{\nu a_\Gamma R \xrightarrow{(i,j,k'):\alpha} \nu a_\Gamma R'_{[k'/k]@i}} \begin{array}{l} either k = k' \\ or \exists k' \in \Gamma,\ k \rightsquigarrow_R k' \end{array}$$

$$\textsc{Com}+\frac{R \xrightarrow{(i,j,k):\overline{b}\langle a\rangle} R' \quad S \xrightarrow{(i,j',k'):b(c)} S'}{R \mid S \xrightarrow{(i,*,*):\tau} R' \mid S'_{[a/c]@i}} \begin{array}{l} k =_* j' \\ k' =_* j \end{array}$$

$$\textsc{Close}+\frac{R \xrightarrow{(i,j,k):\overline{b}\langle \nu a_\Gamma\rangle} R' \quad S \xrightarrow{(i,j',k'):b(c)} S'}{R \mid S \xrightarrow{(i,*,*):\tau} \nu a_\Gamma(R' \mid S'_{[a/c]@i})} \begin{array}{l} k =_* j' \\ k' =_* j \end{array} \textit{with } a \notin \mathrm{fn}(S) \textit{ whenever } \Gamma = \emptyset$$

$$\textsc{Par}+\frac{R \xrightarrow{(i,j,k):\alpha} R'}{R \mid S \xrightarrow{(i,j,k):\alpha} R' \mid S} \mathrm{bn}(\alpha) \cap \mathrm{fn}(S) = \emptyset, i \notin S \quad \textsc{Mem}+\frac{R \equiv_m S \xrightarrow{\zeta} S' \equiv_m R'}{R \xrightarrow{\zeta} R'}$$

Fig. 1. The positive rules of the LTS

2.1 The unlabelled rigid families

A set equipped with a partial order is denoted x, with $|x|$ the underlying set and $e \leq_x e'$ whenever $(e, e') \in x$. The partial orders are called *precedences*, and they represent temporal ordering between the events.

Definition 6 (Rigid families [7, Definitions 1. and 2.]).

- Rigid inclusion *of partial orders $x \preceq y$ is defined iff the following hold:*

$$|x| \subseteq |y| \textit{ and } \begin{cases} \forall e, e' \in x : e \leq_x e' \iff e \leq_y e' \\ \forall e \in y, \forall e' \in x, e \leq_y e' \implies e \in x \end{cases}$$

- *A* rigid family $\mathcal{F} = (E, C)$ *is a set of events E and a non-empty family C of partial orders, called* configurations*, such that $\forall x \in C$, $|x| \in \mathcal{P}(E)$ and C is downward closed w.r.t. rigid inclusion: $\forall y \preceq x, y \in C$.*
- *A* rigid morphism *on rigid families $f : (E, C) \to (E', C')$ is a partial function on events $f : E \rightharpoonup E'$ that is local injective:*

$$\textit{for all } x \in C, e, e' \in x, f(e) = f(e') \implies e = e'$$

and that extends to a (total) function on configurations:

$$f(x) = x' \textit{ iff } \begin{cases} |x'| = \{f(e) \mid e \in x\} \\ e \leq_x e' \iff f(e) \leq_{x'} f(e') \end{cases}$$

Rigid families and their morphisms form a category. Next, let us define some operations on rigid families.

Definition 7 (Operations on rigid families [7, Definition 6.]). *Let* $E^\star = E \cup \{\star\}$.

1. **Product** *Let* $\star$ *denote* undefined *for a partial function. Define* $(E, C) = (E_1, C_1) \times (E_2, C_2)$ *where* $E = E_1 \times_\star E_2$ *is the product in the category of sets and partial functions with the projections* $\sigma_1 : E \to E_1^\star$, $\sigma_2 : E \to E_2^\star$. *Define the projections* $\pi_1 : (E, C) \to (E_1, C_1)$, $\pi_2 : (E, C) \to (E_2, C_2)$ *such that* $\pi_1(e) = \sigma_1(e)$ *and* $\pi_2(e) = \sigma_2(e)$ *and the collection of configurations* $x \in C$ *such that the following hold:*
   (a) x *is a partial order with* $|x| \in \mathcal{P}(E)$;
   (b) $\pi_1(x) \in C_1$ *and* $\pi_2(x) \in C_2$;
   (c) $\forall e, e' \in x$, *if* $\pi_1(e) = \pi_1(e') \neq \star$ *or* $\pi_2(e) = \pi_2(e') \neq \star$ *then* $e = e'$.
   (d) $\forall e, e' \in |x|$, $e <_x e' \iff \pi_1(e) <_{\pi_1(x)} \pi_1(e')$ *and* $\pi_2(e) <_{\pi_2(x)} \pi_2(e')$, *where* $\pi_i(e)$ *defined.*
   (e) $\forall y \preceq x$ *we have that* $\pi_1(y) \in C_1$ *and* $\pi_2(y) \in C_2$.
2. **Restriction** *Define the restriction of an upward closed set of configurations* $X \subseteq C$ *as* $(E, C) \restriction X = (\cup C', C')$ *with* $C' = C \setminus X$. *We equip the operation with a projection* $\pi : (E, C) \restriction X \to (E, C)$ *such that* π *is the identity on events.*
3. **Prefix** *Define* $e.(E, C) = (e \cup E, C' \cup \emptyset)$, *for* $e \notin E$ *where*

$$x' \in C' \iff x' = \big(\{e <_{x'} e' \mid \forall e' \in x\} \cup x\big) \text{ for some } x \in C.$$

   Let $\pi : e.(E, C) \to (E, C)$ *the projection such that* $\pi(e)$ *is undefined and* π *is the identity on the rest of the events.*

Causality and concurrency. Precedence is a partial order that is local to a configuration, but one may also define a global (partial) order as follows.

Definition 8 (Causality [7, Definition 3.]). *Let* $e, e' \in E$ *for* (E, C) *a rigid family. Define* $e' < e$ *if there exists* $x \in C$ *such that* $e, e' \in x$ *and for every* $y \in C$, *if* $e, e' \in y$ *then* $e' <_y e$.

We can generalise Definition 8 to define *disjoint* causality: i.e an event e_1 is caused by either e_2 or e_3.

Definition 9 (Disjoint causality [7, Definition 4.]). *Let* (E, C) *a rigid family and* $e \in E$, $X \subset E$ *such that* $e \notin X$. *Then* X *is a disjoint causal set for* e, *denoted* $X < e$ *iff the following hold:*

1. **disjointness** $\forall e' \in X$, $\exists x \in C$ *such that* $e' <_x e$ *and* $\forall e'' \in X \setminus e'$, $e'' \not<_x e$.
2. **completeness** $\forall x \in C$, $e \in x \implies \exists e' \in X$ *such that* $e' <_x e$;

In particular $e' < e$ *whenever* $\{e'\} < e$.

Definition 10 (Concurrency [7, Definition 5.]). *Let* (E, C) *a rigid family and* $e, e' \in E$. *Define* $e \diamond e' \iff \exists x \in C$, $e, e' \in x$ *such that* $e' \not\leq_x e$ *and* $e \not\leq_x e'$.

Example 1. Consider as an example the rigid family in the left of Figure 2 corresponding to the product of $(\emptyset \prec \{e_1\})$ and $(\emptyset \prec \{e_2\})$. We have that $e_1 \diamond e_2$ thanks to the configuration $\{e_1, e_2\}$.

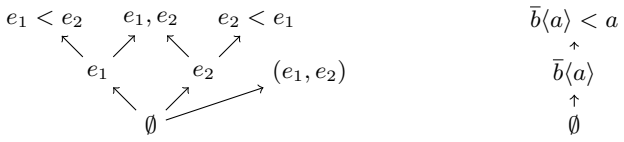

Fig. 2. Examples of rigid families

2.2 The labelled rigid families for the π-calculus

Definition 11 (Labelled rigid families [7, Definition 7.]). *A labelled rigid family $\mathcal{F} = (E, C, \ell, \mathsf{P})$ is a rigid family equipped with a distinguished set of names P (the private names of $\mathcal{F}$) and a labelling function $\ell : E \to L$, where L is the set of labels.*

As in Rπ we apply the substitutions only when needed. Let Σ be the set of all name substitutions. Consider the function $\sigma_x : x \to \Sigma$ that returns a set of substitutions in x. We can then apply the substitutions to the label of e.

Definition 12 (Substitution [7, Definition 13.]). *We define σ_x by induction on the partial order in x:*

$$\begin{aligned}
\sigma_\emptyset =& \emptyset \\
\sigma_x =& \sigma_{x\setminus e} \text{ if } \ell(e) \neq (d(a), \overline{b}\langle a'\rangle) \\
& \sigma_{x\setminus e} \cup \{a'/a\} \text{ if } \ell(e) = (d(a), \overline{b}\langle a'\rangle) \text{ and } \{a''/a'\} \notin \sigma_{x\setminus e} \\
& \sigma_{x\setminus e} \cup \{a''/a\} \text{ if } \ell(e) = (d(a), \overline{b}\langle a'\rangle) \text{ and } \{a''/a'\} \in \sigma_{x\setminus e}
\end{aligned}$$

Define $\ell_x(e) = \ell(e)\sigma_x$, where

$$\big(d(a)\big)\sigma_x = \begin{array}{ll} d'(a) & \text{if } \{d'/d\} \in \sigma_x \\ d(a) & \text{otherwise} \end{array} \qquad \big(\overline{b}\langle a\rangle\big)\sigma_x = \begin{array}{ll} \overline{b'}\langle a'\rangle & \text{if } \{b'/b\}, \{a'/a\} \in \sigma_x \\ \overline{b}\langle a'\rangle & \text{if } \{a'/a\} \in \sigma_x \\ \overline{b'}\langle a\rangle & \text{if } \{b'/b\} \in \sigma_x \end{array}$$

$$\big((\alpha, \beta)\big)\sigma_x = \begin{array}{ll} \big((\alpha)\sigma_x, (\beta)\sigma_x\big) & \\ \overline{b}\langle a\rangle & \text{otherwise} \end{array}$$

The label $\ell_x(e)$ is the *public label* of an event, similar to the public label in Rπ.

The product of rigid families, as in other models for concurrency [10,11], creates all pairs of events that respects the constraints imposed by the morphisms. Labels are then used to detect and remove the events that do not correspond to events in the parallel composition of processes. We do not give here the formal definition of an allowed label.[4] Intuitively, disallowed labels cannot occur during a run of the encoded process.

Definition 13 (Dynamic label [7, Definition 17.]). *Define the dynamic label of an event as $\widehat{\ell}_x(e) = \ell_x(e)$ if $\ell_x(e)$ is allowed and $\perp$ otherwise.*

We extend now the operations of Definition 7 in order to take labels into account.

[4] The reader can refer to the appendix or to [7] for the formal definition.

Definition 14 (Operations on labelled rigid families [7, Definition 10.]).

1. **Restriction of a name** *Let a be a free name. Then $(E, C, \ell, \mathsf{P}) \restriction a = (E, C, \ell, \mathsf{P} \cup \{a\}) \restriction X$, where $x \in X$ iff $\exists e \in x$ such that $\widehat{\ell}_x(e) = \bot$.*
2. **Prefix** *Define $\alpha.(E, C, \ell, \mathsf{P}) = (E', C', \ell', \mathsf{P})$ where, for some $e \notin E$, $e.(E, C) = (E', C')$ and $\ell'(e) = \alpha$ and $\ell'(e') = \ell(e')$ for $e' \neq e$.*
3. **Product** *Let $(E, C) = (E_1, C_1) \times (E_2, C_2)$ be the product and π_1, π_2 the projections $\pi_i : (E, C) \to (E_i, C_i)$. Then*

$$(E_1, C_1, \ell_1, \mathsf{P}_1) \times (E_2, C_2, \ell_2, \mathsf{P}_2) = (E, C, \ell, \mathsf{P}_1 \cup \mathsf{P}_2)$$

where $\ell(e) = \begin{cases} \ell_i(\pi_i(e)) & \text{if } \pi_{3-i}(e) = \star \\ \big(\ell_1(\pi_1(e)), \ell_2(\pi_2(e_2))\big) & \text{otherwise} \end{cases}$

4. **Parallel composition** *Define*

$$(E_1, C_1, \ell_1, \mathsf{P}_1) \mid (E_2, C_2, \ell_2, \mathsf{P}_2) = \big((E_1, C_1, \ell_1, \mathsf{P}_1) \times (E_2, C_2, \ell_2, \mathsf{P}_2)\big) \restriction X$$

where $x \in X$ iff $\exists e \in x$ such that $\widehat{\ell}_x(e) = \bot$.

A rigid family $\mathcal{F}$ is *sound* iff $\forall x \in \mathcal{F}$, $\forall e \in x$, $\widehat{\ell}_x(e) \neq \bot$. All operations defined above preserve the class of sound rigid families. The interpretation of a π process as a rigid family is defined by induction on the structure of a term:

$$[\![\alpha.P]\!] = \alpha.[\![P]\!] \quad [\![P|Q]\!] = [\![P]\!]|[\![Q]\!] \quad [\![\nu a(P)]\!] = [\![P]\!] \restriction a \quad [\![0]\!] = \mathbf{0}$$

One can then show that the encoding is correct by showing that there is an *operational correspondence* between π-processes and their encoding in rigid families. This consists in introducing an LTS defined on rigid families and then establishing a bisimulation between P and $[\![P]\!]$.

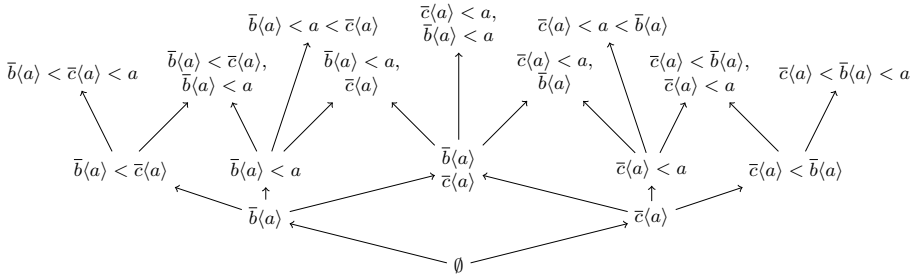

Fig. 3. $\nu a(\overline{b}\langle a\rangle \mid \overline{c}\langle a\rangle \mid a)$ in rigid families

Example 2. Consider first the encoding of the process $\overline{b}\langle a\rangle.a$ depicted in the right of Figure 2, where events are replaced by their labels. We have that $\{\overline{b}\langle a\rangle\}$ is the (only) causal set for a, that is $\overline{b}\langle a\rangle < a$. Moreover, the process $\nu a(\overline{b}\langle a\rangle \mid a)$ has the same interpretation as $\overline{b}\langle a\rangle.a$.

Let us now consider the process $\nu a(\overline{b}\langle a\rangle \mid \overline{c}\langle a\rangle \mid a)$ with its encoding in rigid families presented in Figure 3. The disjoint causal set for the event labelled a consists of the events labelled $\overline{b}\langle a\rangle$ and $\overline{c}\langle a\rangle$, that is $\{\overline{b}\langle a\rangle, \overline{c}\langle a\rangle\} < a$.

3 Encoding the reversible π-calculus

In this section we present our first contribution, which consists in interpreting reversible π-processes in rigid families. We use a similar encoding to the one proposed in Ref.[12] for the encoding of reversible CCS in configuration structures. However, there are notable differences due to name substitution and the extrusion of private names, mechanisms specific to the π-calculus.

The interpretation of a reversible process R consists in a tuple $(\mathcal{F}, x)$, where $x \in \mathcal{F}$ is a configuration, called the *address* of R in $\mathcal{F}$. The tuple $(\mathcal{F}, x)$ can mimick the computations of R: for every forward transition $R \xrightarrow{(i,j,k):\alpha} S$ there exists a configuration $y \in \mathcal{F}$ such that $y = x \cup \{e\}$ and $\ell_y(e) = \alpha$. Similarly, for a backward transition $R \xrightarrow{(i,j,k):\alpha^-} S$, there exists $y \in \mathcal{F}$ such that $x = y \cup \{e\}$ and $\ell_y(e) = \alpha$. The new configuration y is then the address of S inside the same rigid family $\mathcal{F}$.

The configuration $x = \emptyset$ of a rigid family $\mathcal{F}$ corresponds to the initial state of a computation, that is a process with an empty memory. A reversible process R can backtrack all events in its memory until the process with an empty memory is reached. We call this process the *origin* of R and denote it O_R. We apply an *erase* function on O_R, denoted ε, which consists in simply removing the empty memory from the structure of the process. Thus one obtains a π process, which is then encoded in a rigid family using the operations in Definition 14.

In order to interpret a reversible process R one has first to backtrack R to its origin and encode $\varepsilon(O_R)$ in a rigid family $\mathcal{F}$. The encoding of R is then defined by induction on a (forward) trace $\sigma : O_R \longrightarrow^\star R$. The address of the origin is the empty set in $\mathcal{F}$ and every transition in σ is mimicked by a transition inside the same rigid family $\mathcal{F}$. We denote the interpretation of R as $[\![R]\!]_\sigma = (\mathcal{F}, x)$. The configuration x is thus the address of R, which corresponds to the computational state of R, in the sense that they agree on all possible future and past computations.

In the memory of a reversible process we also encode causal relation between events (see Definition 1). We want to ensure that we have the same causal information in the address of the process. For example, consider the origin process $P = \overline{a}\langle d\rangle.\overline{b}\langle c\rangle.Q \mid \overline{b}\langle c\rangle.Q$ and the trace $\emptyset \rhd P \xrightarrow{(i',*,*):\overline{a}\langle d\rangle} R' \xrightarrow{(i,*,*):\overline{b}\langle c\rangle} R$, where we know, by inspecting the memory of R whether $i <_R i'$ or not. We keep track of causality in the rigid family by defining a label and order-preserving bijection between events in R and events in the address of R. The bijection is augmented whenever R does a forward computation and reduced when R backtracks.

Definition 15 (Encoding an Rπ process in rigid families).

$$[\![R]\!]_\sigma = (\mathcal{F}, \mathrm{ad}_{\mathcal{F}}(\emptyset, \emptyset, O_R \longrightarrow^\star R)) \text{ where } \mathcal{F} = [\![\varepsilon(O_R)]\!], \sigma : O_R \longrightarrow^\star R \text{ and}$$

$$\mathrm{ad}_{\mathcal{F}}(x_1, f, R_1 \xrightarrow{(i,j,k):\alpha} R_2 \longrightarrow^\star R_3) = \mathrm{ad}_{\mathcal{F}}(x_2, f \cup \{e \leftrightarrow i\}, R_2 \longrightarrow^\star R_3)$$

$$\textit{where} \begin{cases} (a) & \exists x_2 \in \mathcal{F}, x_1 \prec x_2 \textit{ and } |x_2| = |x_1| \cup \{e\} \\ (b) & \ell_{x_2}(e) = \alpha \\ (c) & [\![\varepsilon(R_2)]\!] = \left(\mathcal{F} \setminus x_2\right) \\ (d) & \forall l \in \mathsf{I}(R_2), l <_{R_2} i \textit{ and } l \neq k \iff f(l) <_{x_2} e \\ (e) & f(k) <_{x_2} e \end{cases}$$

$$\mathrm{ad}_{\mathcal{F}}(x, f, R_2 \longrightarrow^\star R_3) = x \textit{ if } R_2 = R_3$$

In the definition above we denote $\mathrm{ad}_{\mathcal{F}}(x, f, R \longrightarrow^\star S)$ a function that computes the address of S, given $\mathcal{F}$ a rigid family, with $x \in \mathcal{F}$ and f a label and order-preserving bijection between events in R and events in x. It also takes as input a trace from R to S. We initially call the function on the empty set of $\mathcal{F}$ and with an empty bijection, that is $\mathrm{ad}_{\mathcal{F}}(\emptyset, \emptyset, O_R \longrightarrow^\star R)$, and proceed by induction on the trace $\sigma : O_R \longrightarrow^\star R$.

In computing the next address there are several constraints that have to be met, described informally above. Consider the transition $R_1 \xrightarrow{(i,j,k):\alpha} R_2$ and let x_1 be the address of R_1 in $\mathcal{F}$. Then x_2 is the address of R_2 if x_1 can be extended to x_2 (condition a) with an event e that has the corresponding label (condition b) and causal relations in x_2 (conditions d and e). Condition c ensures that R_2 and x_2 agree on future computations.

Remark 1 (Auto-concurrency). The encoding looks for a configuration that has the same past and the same future as the process. It might not always be the case that such a configuration is unique. For instance in the process $a \mid a$ one cannot choose between the two identical singleton configurations. Such processes exhibit *auto-concurrency* [13] and for simplicity we do not consider them in this paper.

Lemma 1 (Soundness of the encoding). *Let R be an Rπ process, O_R its origin and $\sigma : O_R \longrightarrow^\star R$ a trace. Denote $\mathcal{F} = [\![\varepsilon(O_R)]\!]$ the encoding of O_R. There exists $x \in \mathcal{F}$ such that $[\![R]\!]_\sigma = (\mathcal{F}, x)$.*

Proof (Sketch). We use the correspondence between the π-calculus and Rπ from Ref. [4]. Also we have to define an LTS on rigid families and use the correspondence between the π-calculus and rigid families from Ref. [7]. We can then proceed by induction on the trace.

Lemma 1 shows that the encoding is correct: a configuration corresponding to the computational state of a reversible process always exists. The encoding is parametric on the trace. Given a trace from the origin of a process R that leads to R there exists a unique[5] configuration that corresponds to R. However the configuration corresponding to R should be the same for any trace $O_R \longrightarrow^\star R$. This is required by the notion of backtracking used in reversible operational semantics: any forward execution is a valid backward path.

[5] From Remark 1.

Lemma 2. *Let R a process. There exists $x \in [\![\varepsilon(O_R)]\!]$ a configuration such that for all $\sigma : O_R \longrightarrow^\star R$ one has $[\![R]\!]_\sigma = x$.*

Proof. It follows from configurations being uniquely identified by the order on the events, by their labels and by their "future".

4 Causality

In section 3 we presented the encoding of an Rπ term in rigid families. Then in section 5 we will see that we can establish a (weak form) of operational correspondence between Rπ processes and their interpretation in rigid families. Usually, the operational correspondence consists in establishing a bisimulation relation between a process and its encoding. However we cannot show this (strong form of) correspondence as a reversible π process and its interpretation in rigid families are not bisimilar. This is due, intuitively, to the fact that in rigid families all temporal orderings are explicit, not just the causal ones. In this section we make this intuition more concrete, by discussing the difference in the causality relations induced in Rπ and in rigid families.

We say that a configuration x in $\mathcal{F}$ is a *temporal* order if for two events $e, e' \in x$ such that $e <_x e'$, $\exists y \in \mathcal{F}$ with $e \diamond_y e'$. On the other hand, it is a *causal* order if whenever $e, e' \in x$ such that $e <_x e'$, $\nexists y \in \mathcal{F}$ with $e \diamond_y e'$.

Example 3. Consider the process $P = \overline{a}\langle d\rangle \mid \overline{b}\langle c\rangle$ with its encoding $[\![P]\!]$ depicted in Figure 4, where events are replaced by labels. The configurations $\{\overline{a}\langle d\rangle < \overline{b}\langle c\rangle\}$ and $\{\overline{b}\langle c\rangle < \overline{a}\langle d\rangle\}$ are temporal, while $\{\overline{a}\langle d\rangle, \overline{b}\langle c\rangle\}$ is causal. There is no back-and-forth bisimulation between P and $[\![P]\!]$. In a temporal configuration (for instance $\{\overline{a}\langle d\rangle < \overline{b}\langle c\rangle\}$) backtracking can only follow the exact order of the forward execution.

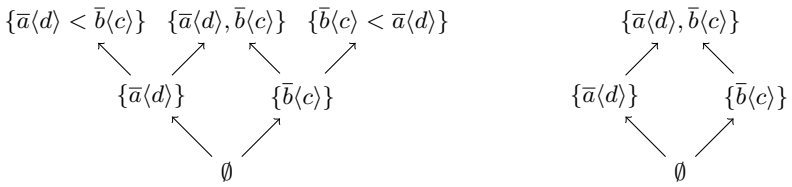

Fig. 4. $\overline{a}\langle d\rangle \mid \overline{b}\langle c\rangle$ in rigid families vs. the LTS of $\overline{a}\langle d\rangle \mid \overline{b}\langle c\rangle$ in RCCS

The causal configurations capture the structural causality induced by a term. Thus one should consider only the orders that are causal and ignore the ones that are temporal. We can define an operator, applied on a rigid familiy that removes the temporal configurations. This step is not compositional, however.

Lemma 3 (Maximal concurrency). *Let $\mathcal{F}$ a rigid family. $\{\mathcal{F}\}$ is obtained by removing all temporal configurations in $\mathcal{F}$:*

$$\{\mathcal{F}\} = \mathcal{F} \upharpoonright X \text{ where}$$
$$x \in X \iff \forall e, e' \in x \text{ if } e <_x e' \text{ then } \forall Y \subseteq E \text{ with } Y < e, e' \notin Y.$$

Proof. We show that the set X defined above is upward closed. The rest follows from the restriction operator in Definition 7. X is upward closed follows from the rigid inclusion between sets.

Remark 2 (Maximal and minimal concurrent versions of a rigid family). $\{[\![P]\!]\}$ preserves all causal configurations and removes all temporal ones. Let us define an opposite operation that removes causal configurations. Denote $\{\{[\![P]\!]\}\}$ the rigid family $[\![P]\!] \upharpoonright X$, where

$$x \in X \iff \exists e, e' \in x, e \diamond_x e'.$$

The rigid family $\{\{[\![P]\!]\}\}$ contains all temporal configurations in $[\![P]\!]$. It follows from a property on rigid families [7] which states that whenever two events are concurrent in a configuration, there exist two temporal configurations that orders the two events.

Considering only forward transitions, one can establish bisimulations between $\{\{[\![P]\!]\}\}$ and either $\{[\![P]\!]\}$ or $[\![P]\!]$. Intuitively, the only difference between the three encodings consists in the backtracking mechanism, and thus the three are bisimilar for the forward computations.

However, rigid families cannot capture the contextual causality as it is defined in Rπ.

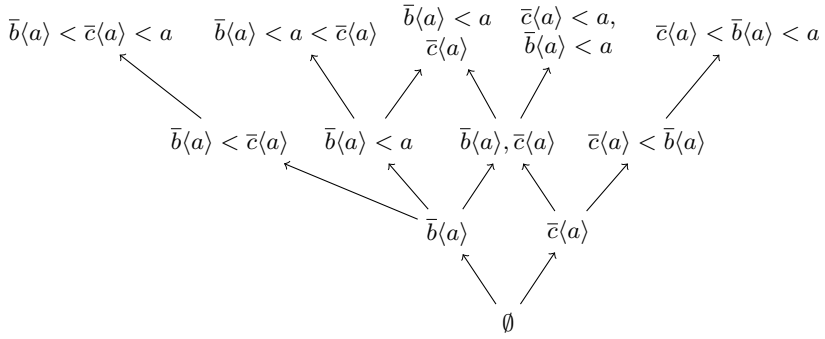

Fig. 5. $[\![\nu a(\overline{b}\langle a\rangle.a \mid \overline{c}\langle a\rangle)]\!] \cong [\![(\overline{b}\langle a\rangle.a \mid \overline{c}\langle a\rangle)]\!]$

Example 4. Consider $P = \nu a(\overline{b}\langle a\rangle.a \mid \overline{c}\langle a\rangle)$ a process with a private name a. The event labelled $\overline{b}\langle a\rangle$ is both a contextual and a structural cause for the input on a. The encoding of P is shown in Figure 5. Denote with e, e', e'' three events such that $\ell(e) = \overline{b}\langle a\rangle$, $\ell(e') = \overline{c}\langle a\rangle$ and $\ell(e'') = a$. The only causal set for event e'' is the singleton $X = \{e\} < e''$. The set $Y = \{e, e'\}$ is not disjoint and it is not a causal set for e''. Lemma 3 applied on this structure yields the rigid family at the right in Figure 6.

In Rπ however, there exists a trace where e'' chooses e' as contextual cause. In such a trace it is not possible then to reverse neither e or e'' before e. The

LTS of P is depicted in the left of Figure 6 and indeed, it contains an additional configuration corresponding to the case where e' is a contextual cause.

Note that the encoding of $\nu a(\overline{b}\langle a\rangle.a \mid \overline{c}\langle a\rangle)$ is isomorphic to the encoding of $\overline{b}\langle a\rangle.a \mid \overline{c}\langle a\rangle$. It suggest that in order to capture the causality in Rπ, one needs an ad-hoc condition on rigid families that takes into account labels and private names.

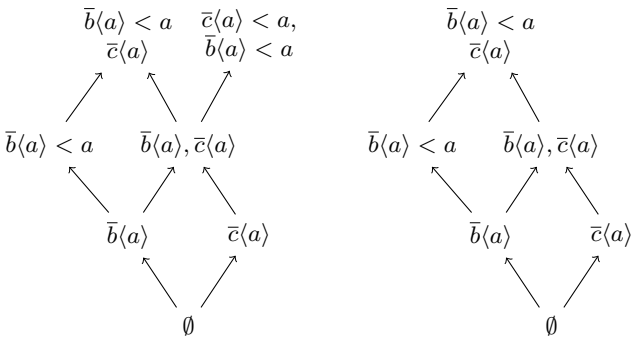

Fig. 6. The LTS in Rπ of $\nu a(\overline{b}\langle a\rangle.a \mid \overline{c}\langle a\rangle)$ and $\{[\![\nu a(\overline{b}\langle a\rangle.a \mid \overline{c}\langle a\rangle)]\!]\}$

Rπ and rigid families both induce causal semantics for the π-calculus. Both integrate reversibility and account for a contextual cause in the case of scope extrusion. Which one is better?

The causal relation in rigid families is coarser than the one in Rπ but it is sufficient for the correctness criteria for reversibility mentioned in the introduction, which consists in *causal consistency* and *maximal concurrency*. These correctness criteria do not require to explicitly choose a contextual cause when a structural one is available. In rigid families, as all temporal orders are explicit, there are configurations where an event is preceded by several contextual causes. It is however not a global relation and it is up to the context to materialise it (i.e. to transform precedence into structural causality).

The contextual causality induced in Rπ satisfies the correctness criteria of reversibility as well. Moreover it also captures the information flow. The contextual cause of an event always precedes the event due to a structural link *in the context*. It is the scope extrusion mechanism that guarantees that such a structural causality exists in the context. In the process $P = \nu a(\overline{b}\langle a\rangle.a \mid \overline{c}\langle a\rangle)$ of Example 4 if the contextual cause of a is $\overline{c}\langle a\rangle$, then a possible reduction context is $C[\cdot] = c(d).\overline{d}$. Thus the synchronisation on channel c structurally precedes the one on channel a. This type of causality requires one to inspect the labels and track the information flow of a name, which is possible in the syntactic setting of Rπ but it is a contrived relation in rigid families.

5 Operational correspondence between Rπ and rigid families

We have seen in the example above that we do not have a bisimulation between R and $[\![R]\!]$. Instead we show Theorem 1 where we only consider transitions on rigid families where both the source and the target of the transitions are interpretations of processes. Let us first define a reversible LTS on rigid families.

Definition 16 (Reversible LTS in rigid families). *Define* $(\mathcal{F}, x_1) \xrightarrow{\ell(e)} (\mathcal{F}, x_2)$ *for* $x_1 \prec x_2$ *and* $|x_2| = |x_1| \cup \{e\}$. *Similarly,* $(\mathcal{F}, x_2) \xrightarrow{\ell(e)^-} (\mathcal{F}, x_1)$. *For* x_1, x_2 *above, one can write* $x_1 \xrightarrow{e} x_2$ *and* $x_2 \xrightarrow{e^-} x_1$.

Theorem 1 (Operational correspondence between an Rπ process R and its image in rigid families). *Let* R *a process and* $[\![R]\!] = (\mathcal{F}, x)$ *its interpretation.*

1. $\forall \alpha$, S *and* $i, j, k \in \mathsf{I}$ *such that* $R \xrightarrow{(i,j,k):\alpha} S$ *then* $[\![R]\!] \xrightarrow{\alpha} [\![S]\!]$;
2. $\forall \alpha$, S *and* $i, j, k \in \mathsf{I}$ *such that* $R \xrightarrow{(i,j,k):\alpha^-} S$ *then* $[\![R]\!] \xrightarrow{\alpha^-} [\![S]\!]$;
3. $\forall e \in E$, $(\mathcal{F}, x) \xrightarrow{\ell(e)} (\mathcal{F}, y)$ *such that* $\exists S$ *with* $(\mathcal{F}, y) = [\![S]\!]$ *then for some* $i, j, k \in \mathsf{I}$, $R \xrightarrow{(i,j,k):\alpha} S$;
4. $\forall e \in E$, $(\mathcal{F}, x) \xrightarrow{\ell(e)^-} (\mathcal{F}, y)$ *such that* $\exists S$ *with* $(\mathcal{F}, y) = [\![S]\!]$ *then for some* $i, j, k \in \mathsf{I}$, $R \xrightarrow{(i,j,k):\alpha^-} S$ *and* $[\![S]\!] = (\mathcal{F}, y)$.

Proof (Sketch).

1. As $R \xrightarrow{(i,j,k):\alpha} S$, $O_R = O_S$ and there exists a trace $O_R \longrightarrow^\star R \xrightarrow{(i,j,k):\alpha} S$. We have that $[\![S]\!] = (\mathcal{F}, x_s)$, where $x_s = \mathrm{ad}_{\mathcal{F}}(\emptyset, f_\emptyset, O_R \longrightarrow^\star S) = \mathrm{ad}_{\mathcal{F}}(\emptyset, f_\emptyset, O_R \longrightarrow^\star R \xrightarrow{i:\alpha} S)$ and $x_R \prec x_S$, $x_R \setminus x_S = \{e\}$, $\ell(e) = \alpha$ by Lemma 1. As $[\![R]\!] = (\mathcal{F}, x_R)$ it follows that $(\mathcal{F}, x_R) \xrightarrow{\alpha} (\mathcal{F}, x_S)$.
3. Consider the transition $(\mathcal{F}, x) \xrightarrow{e} (\mathcal{F}, y)$ with $y \in \mathcal{F}$, $|y| = |x| \cup \{e\}$ and $x \prec y$. We have that $[\![\varepsilon(R)]\!] = \mathcal{F} \setminus x$ and $\{e\}$ is a configuration in $[\![\varepsilon(R)]\!]$. Then $[\![\varepsilon(R)]\!] \xrightarrow{e} [\![\varepsilon(R)]\!] \setminus \{e\} \implies \varepsilon(R) \xrightarrow{\ell(e)} P$ and $[\![P]\!] \cong [\![\varepsilon(R)]\!] \setminus \{e\}$ which follows from the correspondence between the π-calculus and the rigid families. It implies that $\exists i, j, k. R \xrightarrow{(i,j,k):\ell(e)} S$ and $\varepsilon(S) = P$ using the correspondence between the π-calculus and Rπ. From Definition 15, the encoding of R in rigid families uses a bijection $f_R : \mathsf{I}(R) \leftrightarrow x$. We extend the bijection to $f = f_R \cup \{e \leftrightarrow i\}$. We have that there exists S and $i, j, k \in \mathsf{I}(S)$ such that $R \xrightarrow{(i,j,k):\ell(e)} S$. First we show that for all $i' \in \mathsf{I}(R)$ if $f(i') \leq e$ then either $i' <_s i$ or $i' = k$. Secondly we show that we can derive $R \xrightarrow{(i,j,k):\ell(e)} S$ for k such that if $\exists e' \in y$, $e' <_y e$ and $e' \not< e$ then $f(k) = e'$. Lastly we show that for such an S we have that $[\![S]\!] = (\mathcal{F}, y)$.

6 Conclusion

In this paper we propose a denotational model for the reversible π-calculus, consisting of (i) interpretating the reversible π-calculus in rigid families and (ii) establishing an operational correspondance. Rigid families can be viewed as a causal model for process calculi. Similarly, the reversible π-calculus defines a causal semantics for the π-calculus. We compare the two causal models and show that there are subtle differences between the two, due to intuitively, the syntactic nature of causality in the reversible π-calculus.

Configuration structures and rigid families are causal models of process algebra and are thus natural fit for *causal consistent* reversibility. *Out-of-order* reversibility [14] on the other hand, cannot be interpreted in such denotational models.

Rigid families can interpret the causality in CCS and the π-calculus but can also interpret more subtle forms of causality. We intend to investigate it further.

References

1. V. Danos and J. Krivine, "Reversible communicating systems," in *Proc. CONCUR 2004*, vol. 3170 of *LNCS*, pp. 292–307, 2004.
2. I. Phillips and I. Ulidowski, "Reversibility and models for concurrency," *Electron. Notes Theor. Comput. Sci.*, vol. 192, pp. 93–108, Oct. 2007.
3. I. Lanese, C. A. Mezzina, and J.-B. Stefani, "Reversing higher-order π," in *Proc. CONCUR 2010*, pp. 478–493, 2010.
4. I. Cristescu, J. Krivine, and D. Varacca, "A compositional semantics for the reversible π-calculus," in *Proc. LICS 2013*, pp. 388–397, 2013.
5. R. J. van Glabbeek and G. D. Plotkin, "Configuration structures," in *Proc. LICS 1995*, pp. 199–209, 1995.
6. S. Castellan, J. Hayman, M. Lasson, and G. Winskel, "Strategies as concurrent processes," in *Proc. MFPS XXX*, vol. 308 of *ENTCS*, pp. 87 – 107, 2014.
7. I. Cristescu, J. Krivine, and D. Varacca, "Rigid families for CCS and the π-calculus," in *Proc. ICTAC 2015*, vol. 9399 of *LNCS*, pp. 223–240, 2015.
8. R. Milner, *Communicating and Mobile Systems: The π-calculus*. New York, NY, USA: Cambridge University Press, 1999.
9. J. Hayman and G. Winskel, "Event structure semantics for security protocols." Submitted for publication, 2013.
10. S. Crafa, D. Varacca, and N. Yoshida, "Event structure semantics of parallel extrusion in the π-calculus," in *Proc. FOSSACS 2012*, vol. 7213 of *LNCS*, pp. 225–239, 2012.
11. G. Winskel, "Event structure semantics for CCS and related languages," in *Proc. ICALP 1982*, vol. 140 of *LNCS*, pp. 561–576, 1982.
12. C. Aubert and I. Cristescu, "Reversible barbed congruence on configuration structures," in *Proc. ICE 2015*, vol. 189 of *EPTCS*, pp. 68–85, 2015.
13. R. J. van Glabbeek and U. Goltz, "Refinement of actions and equivalence notions for concurrent systems," *Acta Inf.*, vol. 37, no. 4/5, pp. 229–327, 2001.
14. I. Phillips, I. Ulidowski, and S. Yuen, "A reversible process calculus and the modelling of the ERK signalling pathway," in *Proc. RC 2012*, vol. 7581 of *LNCS*, pp. 218–232, 2012.