History-Preserving Bisimulations on Reversible Calculus of Communicating Systems
Clément Aubert, Ioana Cristescu

To cite this version:
Clément Aubert, Ioana Cristescu. History-Preserving Bisimulations on Reversible Calculus of Communicating Systems. 2018. <hal-01778656>

HAL Id: hal-01778656
https://hal.archives-ouvertes.fr/hal-01778656
Submitted on 26 Apr 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
History-Preserving Bisimulations on Reversible Calculus of Communicating Systems

Clément Aubert\textsuperscript{1} and Ioana Cristescu\textsuperscript{2}

\textsuperscript{1} Augusta University, Augusta, GA 30912, USA
\textsuperscript{2} Inria Rennes, France

Abstract. History- and hereditary history-preserving bisimulation (HPB and HHPB) are equivalences for denotational models of concurrency. Finding their counterpart in process algebras is an open problem, with some partial successes: there exists in calculus of communicating systems (CCS) an equivalence based on causal trees that corresponds to HPB. In Reversible CCS (RCCS), there is a bisimulation that corresponds to HHPB, but it considers only processes without auto-concurrency. We propose equivalences on CCS with auto-concurrency that correspond to HPB and HHPB, and their so-called “weak” variants. The equivalences exploit not only reversibility but also the memory mechanism of RCCS.

Keywords: Formal semantics · Process algebras and calculi · Reversible CCS · Hereditary history-preserving bisimulation.

1 Introduction

Reversing Concurrent Computation Implementing reversibility in a programming language often requires to record the history of the execution. Ideally, this history should be complete, so that every forward step can be unrolled, and minimal, so that only the relevant information is saved. Concurrent programming languages have a third requirement: the history should be distributed, to avoid centralization of information. To fulfill those requirements, Reversible CCS\textsuperscript{[6,7]} uses memories attached to the threads of a process.

Equivalences for Reversible Processes A theory of reversible concurrent computation relies not only on a syntax, but also on “meaningful” behavioral equivalences. In this paper we study behavioral equivalences defined on configuration structures\textsuperscript{[14]}, which are denotational models for concurrency. In configuration structures, an event represents an execution step, and a configuration—a set of events that occurred—represents a state. A forward transition is then represented as moving from a configuration to one of its superset, and backward transitions have a “built-in” representation: it suffices to move from a configuration to one of its subset. Many behavioral equivalences have been defined for configuration structures; some of them, like history- and hereditary history-preserving bisimulations (HPB and HHPB), use that “built-in” notion of reversibility.
Encoding Reversible Processes in Configuration Structures

An ongoing research effort \cite{1,10} is to transfer equivalences defined in denotational models, which are by construction adapted for reversibility, back into the reversible process algebra. Of course, showing that an equivalence on configuration structures corresponds to one on RCCS processes depends on the encoding of RCCS terms into configuration structures. One of them uses the fact that we are typically interested only in reachable reversible processes—processes $R$ which can backtrack to a process $O_R$ with an empty memory—called its origin. Then, a natural choice is to consider $[O_R]$—the encoding of the origin of $R$, using the common mapping for CCS processes \cite{14}—, and to identify in it the configuration corresponding to the current state of the reversible process. In this set-up, the encoding of $R$ is one configuration, $[R]$, in the configuration structure $[O_R]$: every configuration “below” is the “past” of $R$, every configuration “above”, its “future” (Fig. 1).

Contribution

This paper improves on previous results by defining relations on CCS processes that correspond to HPB, HHPB, and their “weak function” variants. The result does not require to consider a restricted class of processes. We introduce an encoding of memories independent of the rest of the process and show that, as expected, the “past” of a process corresponds to the encoding of its memory. The memories attached to a process are no longer only a syntactic layer to implement reversibility, but become essential for defining equivalences. This result gives an insight on the expressiveness of reversibility, as the back-and-forth moves of a process are not enough to capture HHPB.

Related work

The correspondence between HHPB and back-and-forth bisimulations for processes without auto-concurrency \cite{1,10} motivated some of the work presented here. Our approach shares similarity with causal trees—in the sense that we encode only part of the execution in a denotational representation—where some bisimulations corresponds to HPB \cite{8}.

Outline

We start by recalling the definitions of configuration structures (Sect. 2.1), of the encoding of CCS in configuration structures (Sect. 2.2), of (hereditary) history-preserving bisimulations (Sect. 2.3), of RCCS (Sect. 2.4) and of related notions. We also recall previous result on HHPB (Theorem 1). We consider the
reader familiar with CCS, in particular with its congruence relations and reduction rules.

Sect. 3 starts by defining a structure slightly richer than configuration structures, that we call “identified configuration structures” (Sect. 3.1), and defines basic operations on them. Sect. 3.2 defines and illustrates with numerous examples how identified configuration structures can encode memories. Finally, Sect. 3.3 uses this encoding to define relations on RCCS and CSS processes that are then stated to correspond to HPB and HHPB on configuration structures.

Sect. 4 concludes, and Sect. A gathers the proofs and establishes the robustness of the tools introduced.

2 Preliminary Definitions

We recall the definitions of configuration structures, auto-concurrency (Sect. 2.1), how to encode CCS processes into configuration structures (Sect. 2.2) and the history-preserving bisimulations (Sect. 2.3).

We write \(\subseteq\) the set inclusion, \(\mathcal{P}\) the power set, \(\setminus\) the set difference, Card the cardinal, \(\circ\) the composition of functions, \(A \to B\) the set of functions between \(A\) and \(B\), \(A \nrightarrow B\) the partial functions and \(f\rvert_C\) the restriction of \(f : A \to B\) to \(C \subseteq A\).

Let \(N = \{a, b, c, \ldots\}\) be a set of names and \(\overline{N} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}\) its co-names. The complement of a (co-)name is given by a bijection \([\cdot] : N \to \overline{N}\), whose inverse is also denoted by \([\cdot]\). We write \([\overline{a}]^\cdot\) for a list of names \(a_1, \ldots, a_n\). We define the sets of labels \(L = N \cup \overline{N} \cup \{\tau\}\), let \(L^-\) = \(\mathcal{P}(L)\), and use \(\alpha\) (resp. \(\lambda, \nu\)) to range over \(L\) (resp. \(L \setminus \{\tau\}\)).

2.1 Configuration Structures

**Definition 1 (Configuration structures).** A configuration structure \(C\) is a tuple \((E, C, \ell)\) where \(E\) is a set of events, \(L \subseteq \mathcal{L}\) is a set of labels, \(\ell : E \to L\) is a labeling function and \(C \subseteq \mathcal{P}(E)\) is a set of subsets satisfying:

\[
\forall x \in C, \forall e \in x, \exists z \in C \text{ finite}, e \in z, z \subseteq x \quad \text{(Finiteness)}
\]

\[
\forall x \in C, \forall e, e' \in x, e \neq e' \Rightarrow \exists z \in C, z \subseteq x, e \in z \iff e' \notin z \quad \text{(Coincidence Freeness)}
\]

\[
\forall X \subseteq C, \exists y \in C \text{ finite}, \forall x \in X, x \subseteq y \Rightarrow \bigcup X \in C \quad \text{(Finite Completness)}
\]

\[
\forall x, y \in C, x \cup y \in C \Rightarrow x \cap y \in C \quad \text{(Stability)}
\]

We denote \(0\) the configuration structure with \(E = \emptyset\), and write \(x \overset{e}{\to} y\) and \(y \overset{e}{\nrightarrow} x\) for \(x, y \in C\) such that \(x = y \cup \{e\}\).

For the rest of this paper, we often omit \(L\), let \(x, y, z\) range over configurations, and assume that we are always given \(C = (E, C, \ell)\) and \(C_i = (E_i, C_i, \ell_i)\), for \(i = 1, 2\).
Definition 2 (Causality, Concurrency, and Maxinality). For \( x \in C \) and \( d, e \in x \), the causality relation on \( x \) is given by \( d <_x e \) if \( d \leq_x e \) and \( d \neq e \), where \( d \leq_x e \) if \( \forall y \in C \) with \( y \subseteq x \), we have \( e \in y \Rightarrow d \in y \). The concurrency relation on \( x \) [9, Definition 5.6] is given by \( d \text{co}_x e \iff \neg (d <_x e \vee e <_x d) \). Finally, \( x \) is maximal if \( \forall y \in C, x = y \) or \( x \not\subseteq y \).

Example 1. Consider the configuration structures of Fig. 2, where the set of events and of configurations can be read from the diagram, and where we make the abuse of notation of writing the events as their labels (with a subscript if multiple events have the same label). Note that two events with complement names can happen at the same time (Fig. 2c), in which case they are labeled with \( \tau \) and called silent transition, as it is usual in CCS (Sect. 2.2).

Definition 3 (Category of configuration structures). We define \( C \) the category of configuration structure, where objects are configuration structures, and a morphism \( f \) from \( C_1 \) to \( C_2 \) is a triple \((f_E, f_L, f_C)\) such that

- \( f_E : E_1 \to E_2 \) preserves labels: \( \ell_2(f_E(e)) = f_L(\ell_1(e)) \), for \( f_L : L_1 \to L_2 \);
- \( f_C : C_1 \to C_2 \) is defined as \( f_C(x) = \{ f_E(e) : e \in x \} \).

If there exists an isomorphism \( f : C_1 \to C_2 \), then we write \( C_1 \cong C_2 \).

We omit the \( f_L \) part of the morphisms when it is the identity morphism.

We now recall how process algebra constructors are defined on configuration structures [13]. The definition below may seem technical, but Definition 6 should make it clear that they capture the right notion.

This definition uses the product \((\times, p_1, p_2)\) of the category of sets and partial functions [13, Appendix A]: letting \(*\) denote undefined for a partial function, \( C^* = C \cup \{\ast\} \) for a set \( C \), we define, for two sets \( A \) and \( B \),

\[
A \times \ast = \{(a, \ast) \mid a \in A\} \cup \{(\ast, b) \mid b \in B\} \cup \{(a, b) \mid a \in A, b \in B\}
\]

with \( p_1 : A \times \ast \to A^* \) and \( p_2 : A \times \ast \to B^* \).
Definition 4 (Operations on configuration structures \[1,12\]).

The product of \(C_1\) and \(C_2\) is \(C_1 \times C_2 = (E_1 \times E_2, C, \ell)\). Define the projections \(\pi_i : C \rightarrow C_i\) and the configurations \(x \in C\) such that:

\[
\forall e \in E, \pi_i(e) = p_i(e), \pi_i(\ell_i(e)) = \ell_i(\pi_i(e))
\]

\[
\pi_i(x) \in C_i
\]

\[
\forall e, e' \in x, \pi_1(e) = \pi_1(e') \neq \ast \text{ or } \pi_2(e) = \pi_2(e') \neq \ast \Rightarrow e = e'
\]

\[
\forall e, e' \in x, e \neq e' \Rightarrow \exists z \subseteq x, \pi_1(z) \in C_1, \pi_2(x) \in C_2, e \in z
\]

\[
\forall e, e' \in x, e \neq e' \Rightarrow \exists z \subseteq x, \pi_1(z) \in C_1, e \in z \iff e' \notin z
\]

The labeling function \(\ell : E_1 \times E_2 \rightarrow L_1 \cup L_2 \cup (L_1 \times L_2)\) is

\[
\ell(e) = \begin{cases} 
\ell_1(e_1) & \text{if } \pi_1(e) = e_1 \neq \ast \text{ and } \pi_2(e) = \ast \\
\ell_2(e_2) & \text{if } \pi_1(e) = \ast \text{ and } \pi_2(e) = e_2 \neq \ast \\
(\ell_1(e_1), \ell_2(e_2)) & \text{otherwise} 
\end{cases}
\]

The relabeling of \(C_1\) along \(r : E_1 \rightarrow L\) is \(r \circ C_1 = (E_1, C_1, r)\).

The restriction of \(C_1\) to \(E \subseteq E_1\) is \(C_1|_E = (E, C|_E, \ell|_E)\), where \(x \in C \iff x \in C_1\) and \(x \subseteq E\). The restriction of \(C_1\) to a name \(a\) is \(C_1|_a := C_1|_{E_a}\) where \(E_a^\ell = \{e \in E_1 \mid \ell(e) \notin \{a, \tau\}\}\).

For \(\overline{a} = a_1, \ldots, a_n\) a list of names, we define similarly \(C_1|_{\overline{a}} = C_1|_{E_{\overline{a}}}\) for \(E_{\overline{a}}^\ell = \{e \in E_1 \mid \ell(e) \notin \{a_1, \overline{a_1}, \ldots, a_n, \overline{a_n}\}\}\).

The parallel composition of \(C_1\) and \(C_2\) is \(C_1 \parallel C_2 = (r \circ (C_1 \times C_2)|_F)\), with

\[-C_1 \times C_2 = C_3 = (E_3, C_3, \ell_3)\] is the product;

\[-r \circ C_3\] with \(r : E_3 \rightarrow L_1 \cup L_2 \cup \{\bot\}\) defined as follows:

\[
r(e) = \begin{cases} 
\ell_3(e) & \text{if } \ell_3(e) \notin \{(a, \overline{a}), (\overline{a}, a), \tau\} \\
\tau & \text{if } \ell_3(e) \in \{(a, \overline{a}), (\overline{a}, a), \tau\} \\
\bot & \text{otherwise} 
\end{cases}
\]

\[-(r \circ C_3)|_F,\] where \(F = \{e \in E_3 \mid r(e) \neq \bot\}\).

The coproduct of \(C_1\) and \(C_2\) is \(C_1 + C_2 = C\), where \(E = \{1\} \times E_1 \cup \{2\} \times E_2\) and \(C = \{1\} \times x \mid x \in C_1\} \cup \{2\} \times x \mid x \in C_2\}\). The labeling function \(\ell\) is defined as \(\ell(e) = \ell_1(\tau_2(e))\) when \(\pi_1(e) = i\).

The prefixing of \(C_1\) by the name \(\lambda\) is \(\lambda C_1 = (e \cup E_1, C, \ell)\), for \(e \notin E_1\) where \(x \in C \iff x = \emptyset \lor \exists x' \in C_1, x = x' \cup e\); \(\ell(e) = \lambda \text{ and all } e' \neq e\), \(\ell(e') = \ell_1(e')\).

Definition 5 (Auto-concurrency \[9, \text{Definition 9.5}\]). If \(\forall x \in C, \forall e_1, e_2 \in x, e_1 co_x e_2 \text{ and } \ell(e_1) = \ell(e_2) \text{ implies } e_1 = e_2\), then \(C\) is without auto-concurrency.

Any configuration structure where configurations have at most one event (like Fig. 2a) are without auto-concurrency. Fig. 2b, on the other hand, is a configuration structure with auto-concurrency: for \(x = \{a_1, a_2\}\) we have that \(a_1 co_x a_2\), \(\ell(a_1) = \ell(a_2)\), and yet \(a_1 \neq a_2\).
2.2 CCS and its Encoding in Configuration Structures

The set of CCS processes $P$ is inductively defined:

\[ P, Q := \lambda P \parallel P \parallel Q \parallel \lambda P + \nu Q \parallel P \setminus a \parallel 0 \]  

(CCS Processes)

In the category of configuration structures $C$ (Definition 3) one can “match” the process constructors of CCS with the categorical operations of Definition 4.

Definition 6 (Encoding a CCS process [15, p. 57]). Given a CCS process $P$, its encoding $[P]$ as a configuration structure is built inductively:

\[
\begin{align*}
[\lambda P] &= \lambda[P] \\
[P \parallel Q] &= [P] \parallel [Q] \\
[\lambda P + \nu Q] &= [\lambda P] + [\nu Q] \\
[P \setminus a] &= [P]_a \\
[0] &= 0
\end{align*}
\]

For now on we assume that all structures use the same set of labels $L$.

Definition 7 (Auto-concurrency in CCS). A process $P$ is without auto-concurrency if $[P]$ is.

2.3 (Hereditary) History-Preserving Bisimulations

HPB [11,10], [2, Theorem 4] and HHPB [3, Definition 1.4], [2, Theorem 1] are equivalences on configuration structures that use label- and order-preserving bijections between the events of the two configuration structures.

Definition 8 (Label- and order-preserving functions). A function $f : x_1 \rightarrow x_2$, for $x_i \in C_i$, $i \in \{1, 2\}$ is label-preserving if $\ell_1(e) = \ell_2(f(e))$ for all $e \in x_1$. It is order-preserving if $e_1 \leq x_1, e_2 \Rightarrow f(e_1) \leq x_2, f(e_2)$, for all $e_1, e_2 \in x_1$.

Definition 9 (HPB and HHPB). A relation $R \subseteq C_1 \times C_2 \times (E_1 \rightarrow E_2)$ such that $(\emptyset, \emptyset, \emptyset) \in R$, and if $(x_1, x_2, f) \in R$, then $f$ is a label- and order-preserving bijection between $x_1$ and $x_2$ and (1) and (2) (resp. (1-4)) hold is called a history- (resp. hereditary history-) preserving bisimulation between $C_1$ and $C_2$.

\[
\begin{align*}
\forall y_1, x_1 \xrightarrow{c_1} y_1 \Rightarrow & \exists y_2, g, x_2 \xrightarrow{c_2} y_2, g \mid x_1 = f, (y_1, y_2, g) \in R \quad (1) \\
\forall y_2, x_2 \xrightarrow{c_2} y_2 \Rightarrow & \exists y_1, g, x_1 \xrightarrow{c_1} y_1, g \mid x_1 = f, (y_1, y_2, g) \in R \quad (2) \\
\forall y_1, x_1 \xrightarrow{\tau} y_1 \Rightarrow & \exists y_2, g, x_2 \xrightarrow{\tau} y_2, g = f \mid x_1, (y_1, y_2, g) \in R \quad (3) \\
\forall y_2, x_2 \xrightarrow{\tau} y_2 \Rightarrow & \exists y_1, g, x_1 \xrightarrow{\tau} y_1, g = f \mid x_1, (y_1, y_2, g) \in R \quad (4)
\end{align*}
\]

Note that the bijection on events is preserved from one step to the next. This condition can be weakened, and we call the corresponding relations the weak-function HPB and weak-function HHPB [3, Definition1.4], [10, Definition3.11].

\footnote{The names weak-HPB and weak-HHPB are more common [3,10], but can be confused with the weak equivalences of process algebra, which refers to ignoring $\tau$-transitions.}
**Definition 10 (wfHPB).** A weak-function history-preserving bisimulation between $C_1$ and $C_2$ is a relation $R \subseteq C_1 \times C_2 \times (E_1 \rightarrow E_2)$ such that $(\emptyset, \emptyset, \emptyset) \in R$ and if $(x_1, x_2, f) \in R$, then $f$ is a label- and order-preserving bijection between $x_1$ and $x_2$ and

$$
\forall y_1, x_1 \xrightarrow{e_1} y_1 \Rightarrow \exists y_2, g, x_2 \xrightarrow{e_2} y_2, (y_1, y_2, g) \in R
$$

$$
\forall y_2, x_2 \xrightarrow{e_2} y_2 \Rightarrow \exists y_1, g, x_1 \xrightarrow{e_1} y_1, (y_1, y_2, g) \in R
$$

Similarly one defines $wfHHPB$. If there is a HPB between $C_1$ and $C_2$, we just write that $C_1$ and $C_2$ are HPB, and similarly for HHPB, $wfHPB$ and $wfHHPB$.

![Fig. 3: There is no HHPB relation between $[a.a | b]$ and $[a | a | b]$.](image)

**Example 2.** Observe that $[a.a | b]$ and $[a | a | b]$, presented in Fig. 3, are HPB, but not HHPB. Any HHPB relation would have to associate the maximal configurations $\{a_1, a_2, b\}$ of the two structures and to construct a bijection: taking $(a_1 \mapsto a_1, a_2 \mapsto a_2, b \mapsto b)$ wouldn’t work, since $[a | a | b]$ can backtrack on $a_2$ and $[a.a | b]$ cannot. Taking the other bijection, $(a_1 \mapsto a_2, a_2 \mapsto a_1, b \mapsto b)$, fails too, since $[a | a | b]$ can backtrack on $a_1$ and $[a.a | b]$ cannot.

**Example 3 ([9]).** The processes $a \mid (b+c)+a \mid b+(a+c) \mid b$ and $a \mid (b+c)+(a+c) \mid b$ is another example of processes whose encodings are HPB but not HHPB.

**Example 4.** Finally, observe that $a.(b + b)$ and $a.b + a.b$ are not structurally congruent in CCS, but the encoding of the two processes are HHPB.

### 2.4 Reversible CCS and Coherent Memories

Let $I$ be a set of identifiers, and $i, j, k$ range over elements of $I$. The set of RCCS processes $R$ is built on top of the set of CCS processes $P$ (Sect. 2.1):

- $e := (i, \lambda, P)$ (Memory Events)
- $m := \emptyset \parallel \gamma.m \parallel e.m$ (Memory Stacks)
- $T, U := m \triangleright P$ (Reversible Thread)
- $R, S := T \parallel T \parallel U \parallel R \setminus a$ (RCCS Processes)
We denote $I(m)$ (resp. $I(R)$, $I(e)$) the set of identifiers occurring in $m$ (resp. $R$, $e$), and always take $I = \mathbb{N}$. A structural congruence $\equiv$ can be defined on RCCS terms \cite{1}, Definition 5, the only rule we will use here is distribution of memory: $m \triangleright (\overline{P} | Q) \equiv (\overline{\gamma}.m \triangleright P) | (\overline{\gamma}.m \triangleright Q)$. We also note that $R \equiv (\prod_{i=1}^{n} m_{i} \triangleright P_{i}) \backslash \overline{a}$. for any reversible process $R$ and for some names $\overline{a}$, memories $m_{i}$ and CCS processes $P_{i}$, writing $\prod_{i}$ the $i$-ary parallel composition.

![Fig. 4: Rules of the labeled transition system](image)

The labeled transition system for RCCS is given by the rules of Fig. 4. We use $\triangleright$ as a wildcard for $\triangleright_{\lambda}$ (forward) or $\triangleright_{\lambda}^{-}$ (backward transition), and if there are indices $i_{1}, \ldots, i_{n}$ and labels $\alpha_{1}, \ldots, \alpha_{n}$ such that $R_{1} \triangleright_{\lambda}^{-} \cdots \triangleright_{\lambda}^{-} R_{n}$, then we write $R_{1} \triangleright_{\lambda}^{-} \cdots \triangleright_{\lambda}^{-} R_{n}$. If there is a CCS term $P$ such that $\emptyset \triangleright P \triangleright \tau \triangleright R$, we say that $R$ is reachable, that $P$ is the origin of $R$ \cite[Lemma 1]{1} and write $P = O_{R}$. Similarly to what we did in Definition 7, we will write that $R$ is without auto-concurrency if $[O_{R}]$ is. An example of the execution of a reversible process is given at the beginning of Example 7.

Note that we cannot work up to $\alpha$-renaming of the identifiers: concurrent, distributed computation is about splitting threads between independent units of computation. If one of unit were to re-tag a memory event $\langle 89, \lambda, P \rangle$ as $\langle 1, \lambda, P \rangle$, and another were to try to backtrack on the memory event $\langle 89, \lambda, P \rangle$, then the trace of the synchronization would be lost, and backtracking made impossible. Since we don’t want to keep a “global index” which would go against the benefits of distributed computation, the only option is to forbid $\alpha$-renaming of identifiers.

Memory coherence \cite[Definition 1]{6} was defined for RCCS processes with less structured memory events (i.e., without identifiers), but can be adapted.

**Definition 11 (Coherence relation).** Coherence, written $\bowtie$, is the smallest symmetric relation on memory stacks such that rules of Fig. 5 hold.

Note that $\bowtie$ is not reflexive, and hence not an equivalence, nor anti-reflexive. For the rest of this paper, we will just write “memory” for “memory stack".
Definition 12 (Coherent processes, [6, Definition 2]). A RCCS process is coherent if all of its memories are pairwise coherent, or if its only memory is coherent with ∅.

We require the memory to be coherent with ∅ to make it impossible to have \( \langle 1, \lambda, P \rangle, (1, \nu, Q), \emptyset \) as a memory in a coherent process.

Lemma 1 ([7, Lemma 5]). If \( R \xrightarrow{\cdot} S \) and \( R \) is coherent then so is \( S \).

Corollary 1. For every reachable \( m \triangleright P \) and \( i \in I(m) \), \( i \) occurs once in \( m \).

Note that the property above holds for reversible threads, and not for RCCS processes in general: indeed, we actually want memory events to have the same identifiers if they result from a synchronization or a fork.

Definition 13 (Back-and-forth bisimulation). A back-and-forth bisimulation in RCCS is a relation \( R \subseteq R \times R \) such that if \( (R_1, R_2) \in R \)

\[ \forall S_1, R_1 \xrightarrow{i\alpha} S_1 \Rightarrow \exists S_2, R_2 \xrightarrow{j\alpha} S_2, (S_1, S_2) \in R \]

\[ \forall S_2, R_2 \xrightarrow{i\alpha} S_2 \Rightarrow \exists S_1, R_1 \xrightarrow{j\alpha} S_1, (S_1, S_2) \in R \]

\[ \forall S_1, R_1 \xrightarrow{\emptyset} S_1 \Rightarrow \exists S_2, R_2 \xrightarrow{\emptyset} S_2, (S_1, S_2) \in R \]

\[ \forall S_2, R_2 \xrightarrow{\emptyset} S_2 \Rightarrow \exists S_1, R_1 \xrightarrow{\emptyset} S_1, (S_1, S_2) \in R \]

Example 5. The processes \( \emptyset \triangleright (a.a \mid b) \) and \( \emptyset \triangleright (a \mid a \mid b) \) are in a back-and-forth bisimulation.

Theorem 1 ([1, Theorem 2],[10]). Back-and-forth bisimulation on RCCS processes without auto-concurrency corresponds to HHPB on their encoding.

Note that this result does not hold for the processes in Example 5 (see also Example 2). This does not contradict the theorem, as the processes in Example 5 are with auto-concurrency.

3 Lifting the Restrictions

To define a bisimulation on RCCS (with auto-concurrency) that corresponds to HHPB we first have to encode the memories of a reversible process into a structure similar to the configuration structures, called identified configuration structures (Sect. 3.1). We can then define the encoding (Sect. 3.2), and the equivalences in RCCS that use this encoding of memories (Sect. 3.3).
3.1 Identified Configuration Structures

Definition 14 (Identified configuration structure). An identified configuration structure, or \( \mathcal{D} \)-structure, \( \mathcal{D} = (E, C, \ell, I, m) \) is a configuration structure \( \mathcal{C} = (E, C, \ell) \) endowed with a set of identifiers \( I \) and a function \( m : E \to I \) such that,

\[
\forall x \in C, \forall e_1, e_2 \in x, m(e_1) \neq m(e_2)
\]

(Collision Freeness)

We call \( \mathcal{C} \) the underlying configuration structure of \( \mathcal{D} \) and write \( \mathcal{D} = \mathcal{C} + m \). We write \( \emptyset \) for the identified configuration structure with \( E = \emptyset \).

For the rest of this paper, we omit \( I \), and assume that we are always given \( \mathcal{D} = (E, C, \ell, m) \) and \( \mathcal{D}_i = (E_i, C_i, \ell_i, m_i) \), for \( i = 1, 2 \).

Example 6. Fig. 2c, with \( m(a) = 1, m(\overline{a}) = 2, m(b) = 3 \) and \( m(\tau) = 4 \), is a \( \mathcal{D} \)-structure. Note that it is possible to have fewer identifiers than events: take \( I = \{1, 2, 3\} \) and \( m(a) = m(\tau) = 1, m(\overline{a}) = 2 \) and \( m(b) = 3 \).

For the following remark, we need to suppose that every configuration structure is endowed with a total ordering on its events.

Remark 1. Every configuration structure can be mapped to a \( \mathcal{D} \)-structure.

The mapping is trivial: take \( I \) to be \( \{1, \ldots, \text{Card } E\} \), and define \( m : E \to I \) to follow the ordering on \( E \). Note that, in this case, \( m \) is a bijection.

Definition 15 (Category of \( \mathcal{D} \)-structures). We define \( \mathbb{D} \) the category of identified configuration structure, where objects are \( \mathcal{D} \)-structures, and a morphism \( f \) from \( \mathcal{D}_1 \) to \( \mathcal{D}_2 \) is a triple \( (f_E, f_C, f_m) \) such that

- \((f_E, f_C)\) is a morphism in \( \mathcal{C} \) from \((E_1, C_1, \ell_1)\) to \((E_2, C_2, \ell_2)\);
- \(f_m : I_1 \to I_2\) preserves identifiers: \( f_m(m_1(e)) = m_2(f_E(e)) \).

We denote \( \mathcal{F} : \mathbb{D} \to \mathbb{C} \) the forgetful functor.

Definition 16 (Operations on \( \mathcal{D} \)-structures).

The product of \( \mathcal{D}_1 \) and \( \mathcal{D}_2 \) is \( \mathcal{D}_1 \times \mathcal{D}_2 = (E, C, \ell, m) \):

- \((E_1, C_1, \ell_1) \times (E_2, C_2, \ell_2) = (E, C, \ell) \) is the product in the category of configuration structures with projections \( \pi_i : (E, C, \ell) \to (E_i, C_i, \ell_i) \);
- \(m : E_1 \times_\times E_2 \to (I_1 \cup \{m_\ast\}) \times (I_2 \cup \{m_\ast\})\), for \( m_\ast \notin I_1 \cup I_2 \), is defined as

\[
m(e) = \begin{cases} 
(m_1(\pi_1(e)), m_\ast) & \text{if } \pi_2(e) = \ast \\
(m_1(\pi_1(e)), m_2(\pi_2(e))) & \text{if } \pi_1(e) = \ast \\
(m_1(\pi_1(e)), m_2(\pi_2(e))) & \text{otherwise}
\end{cases}
\]

with the projections \( \pi_i : m \to m_i \cup \{m_\ast\} \).

Define the projections \( \gamma_i : \mathcal{D}_1 \times \mathcal{D}_2 \to \mathcal{D}_i \) as the pair \((\pi_i, \pi_i)\).

The relabeling of \( \mathcal{D}_1 \) along \( r : E_1 \to L \) is \( r \circ \mathcal{D}_1 = (E_1, C_1, r, m_1) \).

The restriction of \( \mathcal{D}_1 \) to \( E \subseteq E_1 \) is \( \mathcal{D}_1 \mid_E = ((E_1, C_1, \ell_1) \mid_E + m_1 \mid_E) \).
The parallel composition of $D_1$ and $D_2$ is $D_1 \parallel D_2 = (r \circ (D_1 \times D_2)) | F$, with $\neg D_1 \times D_2 = D_3 = (E_3, C_3, \ell_3, m_3)$ is the product of $D$-structures. 

$\neg r \circ D_3$, with $r : E_3 \rightarrow L \cup \{ \bot \}$ defined as follows,

$$
r(e) =
\begin{cases}
\bot & \text{if } \pi_1(e) = e_1 \neq * \land \pi_2(e) = e_2 \neq * \land m_1(e_1) \neq m_2(e_2) \\
\text{or if } \pi_1(e) = e_1 \neq * \land \pi_2(e) = * \\
\land (\exists e_2 \in E_2, \text{ s.t. } m_1(e_1) = m_2(e_2)) \\
\text{or if } \pi_2(e) = e_2 \neq * \land \pi_1(e) = * \\
\land (\exists e_1 \in E_1, \text{ s.t. } m_1(e_1) = m_2(e_2))
\end{cases}
$$

$\tau$ if $\pi_1(e) = e_1 \neq * \land \pi_2(e) = e_2 \neq * \land m_1(e_1) = m_2(e_2)$ and $\ell_3(e) = (\alpha, \pi)$ (Valid Synchronisations)

$\alpha$ if $\pi_1(e) = e_1 \neq * \land \pi_2(e) = e_2 \neq * \land m_1(e_1) = m_2(e_2)$ and $\ell_3(e) = (\alpha, \alpha)$ (Valid Forks) otherwise

$-(r \circ D_3)|_F$, where $F = \{ e \in E_3 \mid r(e) \neq \bot \}$.

In the definition of parallel composition, $r$ detects the wrong synchronization (or fork) pairs: if two events occur at the same time, then they must have the same identifier. And if an event occurs alone, then no other event can occur with the same identifier.

### 3.2 Encoding the Memory of Reversible Processes

**Definition 17 (Encoding a RCCS memory).** The encoding of the memory of a RCCS process in a $D$-structure is defined\(^4\) by induction on the process:

- $[m \triangleright P] = [m]$
- $[R_1 | R_2] = [R_1] | [R_2]$
- $[R \setminus a] = [R]$
- $[(i, \alpha, P).m] = (E, C, m)$
- $[\forall. m] = [m]$
- $[\emptyset] = 0$

For $[(i, \alpha, P).m]$, let $[m] = (E_m, C_m, \ell_m, m_m)$ and $e \notin E_m$, we let $(E, C, \ell, m)$ be $E = E_m \cup e$, $C = C_m \cup \{ x \cup \{ e \} \mid x \in C_m \text{ is maximal} \}$, $\ell = \ell_m \cup \{ e \mapsto \alpha \}$ and $m(e') = m_m(e')$ if $e' \neq e$, and $m(e) = i$ otherwise.

The memories of any RCCS process could be encoded into $D$-structures, but we will encode only reachable (and thus coherent) processes (Definition 12).

**Example 7.** Consider the following transitions:

$$
\emptyset \triangleright (a.b \mid c.\overline{a}) \equiv (\forall. \emptyset \triangleright a.b) \mid (\forall. \emptyset \triangleright c.\overline{a})
$$

$$
1 \xrightarrow{c} (\forall. \emptyset \triangleright a.b) \mid (\langle 1, c, 0 \rangle. \forall. \emptyset \triangleright \overline{c})
$$

$$
2 \xrightarrow{\pi} (\langle 2, a, 0 \rangle. \forall. \emptyset \triangleright b) \mid (\langle 2, \pi, 0 \rangle. \langle 1, c, 0 \rangle. \forall. \emptyset \triangleright 0)
$$

$$
3 \xrightarrow{b} (\langle 3, b, 0 \rangle. \langle 2, a, 0 \rangle. \forall. \emptyset \triangleright 0) \mid (\langle 2, \pi, 0 \rangle. \langle 1, c, 0 \rangle. \forall. \emptyset \triangleright 0)
$$

\(^4\) We make the abuse of notation of writing $[\cdot]$ for the encoding of both a reversible process and a memory, into a $D$-structure.
Fig. 6: (Identified) Configurations Structures of Examples 7 and 8

Fig. 7: Configurations Structure of Example 7 (Continued)
Now, observe that:

\[
[(3, b, 0). (2, a, 0). \gamma . \emptyset \triangleright 0) | (2, \overline{a}, 0). (1, c, 0). \gamma . \emptyset \triangleright 0)]
\]
\[
= [(3, b, 0). (2, a, 0). \gamma . \emptyset \triangleright 0 | [(2, \overline{a}, 0). (1, c, 0). \gamma . \emptyset \triangleright 0]
\]
\[
= [(3, b, 0). (2, a, 0). \gamma . \emptyset | [(2, \overline{a}, 0). (1, c, 0). \gamma . \emptyset]
\]

If we let (with the obvious labeling functions \( \ell(x) = x \) for \( x \in \{a, \overline{a}, b, c\} \)):

\[
[(3, b, 0). (2, a, 0). \gamma . \emptyset] = (\{a, b\}, \{\emptyset, \{a\}, \{b, a\}\}, \{a \mapsto 2, b \mapsto 3\}]
\]
\[
[(2, \overline{a}, 0). (1, c, 0). \gamma . \emptyset] = (\{c, \overline{a}\}, \{\emptyset, \{c\}, \{c, \overline{a}\}\}, \{c \mapsto 1, \overline{a} \mapsto 2\})
\]

Then we respectively get the structures of Figures 6a and 6b.

The product of those two configurations results in the following set of events (where we keep naming the events after their label):

\[
(a, \ast) \quad (\text{ev. 1}) \quad (\ast, c) \quad (\text{ev. 3}) \quad (a, c) \quad (\text{ev. 5}) \quad (b, c) \quad (\text{ev. 7})
\]
\[
(b, \ast) \quad (\text{ev. 2}) \quad (\ast, \overline{a}) \quad (\text{ev. 4}) \quad (a, \overline{a}) \quad (\text{ev. 6}) \quad (b, \overline{a}) \quad (\text{ev. 8})
\]

When doing the parallel composition, the relabeling labels with \( \perp \) events ev. 5, ev. 7 and ev. 8, since their event identifier do not match, as well as ev. 1 and ev. 4, since they have the same identifiers but occur unsynchronized. Hence, we obtain the \( \mathcal{D} \)-structure of Fig. 6c, with \( m(\ast, c) = 1 \), \( m(a, \overline{a}) = 2 \) and \( m(b, \ast) = 3 \).

Observe the encoding of the origin of this process, \( a.b \mid c.\overline{a} \), in Fig. 7: the underlying configuration of the encoding of the memory previously exposed is just a particular path in that configuration structure. This remark is made formal in Lemma 2.

Example 8. Similarly, we can encode the execution

\[
\emptyset \triangleright (a.b \mid c) \xrightarrow{1c} \xrightarrow{2a} \xrightarrow{3b} ((3, b, 0). (2, a, 0). \gamma . \emptyset \triangleright 0) | ((1, c, 0). \gamma . \emptyset \triangleright 0)
\]

and obtain the structure of Fig. 7, with \( m(c) = 1, m(a) = 2 \) and \( m(b) = 3 \).

For the rest of this subsection, we assume given a coherent reversible process \( R \) whose origin is written \( O_R \). As expected the underlying configuration of \( [R] \) is included in \([O_R]\) (this will be a direct consequence of Lemma 4):

**Lemma 2.** \( \mathcal{F}([R]) \subseteq [O_R] \).\(^5\)

The following lemma states that all memory events in \( R \) are either causally linked or concurrent. Therefore their encoding results in partially ordered set (poset) with one maximal element (Definition 2), linked by the subset relation. This is illustrated by Examples 7 and 8, and proved by induction on the RCCS process.

**Lemma 3.** \( [R] \) is a poset with one maximal configuration.

\(^5\) Formally, \( C_1 = (E_1, C_1, \ell_1) \subseteq C_2 = (E_2, C_2, \ell_2) \) iff \( E_1 \subseteq E_2 \); \( C_1 \subseteq C_2 \) and \( \ell_1 \subseteq \ell_2 \).
Another way to encode a reachable memory in a configuration structure is to encode $R$ as a configuration, called the address of $R$, inside $[O_R]$:

$$[R] = ([O_R], x)$$

where $x$ is a configuration in $[O_R]$.

**Definition 18 (Generation).** Given $(E, C, \ell)$ and $x \in C$, the configuration structure generated by $x$ is $x \downarrow = (x, C_x, \ell \upharpoonright x)$, where $C_x = \{ y \mid y \in C, y \subseteq x \}$.\(^7\)

The encoding of the memory of $R$, $[R]$, is the configuration structure generated by the configuration $x$. In other words, the memory of $R$ is “everything under” the address of $R$ in $[O_R]$. We state this formally in the following lemma:

**Lemma 4.** For $[R] = ([O_R], x)$, $\mathcal{F}([R]) \cong x \downarrow$.

We can now make the intuitions of Fig. 1 formal, letting $R \equiv (\prod_i m_i \rhd P_i) \rhd a$ and $O_R$ be its origin. Informally speaking, the memories of $R$, i.e. the $m_i$, is the “past” of $R$, and the process $(\prod_i P_i) \rhd a$ is its “future”: in the configuration structure $[O_R]$, they are respectively represented by $\mathcal{F}([R])$ and $[[\prod_i P_i] \rhd a]$. Consider Example 7: the past of the process is the structure shown in Fig. 6c, which corresponds to the gray part in Fig. 6d. The future of this process is empty, as all actions have been consumed.

However, we cannot recover the origin process from the encoding of the past and future of a reversible process. There is a loss of information that occurs for the synchronisation events when encoding memories into identified configuration structures: we label $\tau$ such events, whereas in the memories of a RCCS process the synchronised events keep their input (or output) label.

### 3.3 (Hereditary) History-Preserving Bisimulations on CCS

We adapt the (hereditary) history-preserving bisimulations of Sect. 2.3 to CCS, making the bijections becomes isomorphisms between memory encodings.

Below, we let $P_1$ and $P_2$ be two CCS processes (possibly with auto-concurrency, Definition 7), and $[P_i] = (E_i, C_i, \ell_i)$, for $i = 1, 2$.

**Definition 19 (HPB and HHPB on CCS).** A relation $R \subseteq \mathbb{R} \times \mathbb{R} \times (E_1 \rightarrow E_2)$ such that $(\emptyset \rhd P_1, \emptyset \rhd P_2, \emptyset) \in R$ and if $(R_1, R_2, f) \in R$ then $f$ is an isomorphism between $\mathcal{F}([R_1])$ and $\mathcal{F}([R_2])$ and (6) and (7) (resp. (6–9)) hold is called a

---

\(^6\) By abuse of notation, $[\cdot]$ denotes two different encodings, on CCS and RCCS processes.

\(^7\) For the reader familiar with event structures, a configuration $x$ defines an event structure $(x, \leq_x, \ell)$ for $\leq_x$ the causality in $x$. The construction here mirrors the transformation from an event structures to a configuration structure [15].
and we defined a series of bisimulations $(\text{wfHPB})$ on configuration structures. We managed therefore to define an equivalence on CCS which distinguishes for instance $a.a | b$ from $a | a | b$, whose encodings are not HHPB. We would like to conclude with three observations.

Fist, we should stress that our relations are defined in terms of CCS processes: on the surface, this paper offers a new result on non-reversible CCS, using tools stemming from the study of reversible computation. We believe this is an interesting contribution, that witnesses the relevance of studying concurrent reversible computation.

Our paper also introduces a natural technical tool, identified configuration structure, to encode information on memory events. It should be noted that a memory event is made of three elements, $(i, \alpha, Q)$, and that configuration structures encode names $\alpha$, and $D$-structures furthermore encode identifiers $i$. We
could take a step further and also enclose information about the residual process $Q$ in a sum. Representing the “whole memory” in denotational model could lead to interesting new bisimulation on processes, but we leave this for a future possible extension.

Other CCS bisimulations such as the pomset bisimulations [5] or the localities bisimulations [4] are known to be different from the history-preserving ones. However, these bisimulations add some information on auto-concurrent events, that can be used to distinguish them. A possible direction of future work is then to adapt these bisimulations to the reversible setting to maybe capture HHPB.

References

A Appendix

The appendix is divided in three subsections.

Sect. A.1 gathers results about identified configuration structures, introduced in Sect. 3.1. They are not required to understand the rest of the technical development, but illustrates why we believe this notion is “solid”, and gives some insights on how to manipulate it.

The main purpose of Sect. A.2 is to prove Lemma 4, stated in Sect. 3.2. This apparently intuitive result actually requires a heavy machinery to be proven: not only do we prove Lemma 3, but we also state and prove some intermediate lemmas.

The immediate advantage of Lemma 4 is that it makes the proof of Theorem 2, in Sect. A.3, almost straightforward.

Before doing so, we need to prove the only new result of Sect. 2:

Proof (Corollary 1, page 9). Since \( m \rhd P \) is reachable, \( m \rhd P \) is coherent by Lemma 1, and hence \( m \rhd \emptyset \) by Definition 12. Looking at Definition 11, the only way to derive \( m \rhd \emptyset \) is to start with \( \text{em.} \), and then to apply \( \text{ev.} \) to “stack” the memory events. If an identifier were to appear in two memory event in \( m \), we would not be able to use \( \text{ev.} \) to add the second memory event, since the side-condition would forbid it.

A.1 On the Robustness of Identified Configuration Structures

This subsection gathers elementary results on the structures introduced in Sect. 3.1. These results, e.g., that the category of \( \mathcal{D} \)-structures of Definition 15 is indeed a category (Lemma 5) or that the operations on \( \mathcal{D} \)-structures of Definition 16 are well-defined (Lemma 7), were not stated in the body of the document, but are implicitly used.

Lemma 5. Identified configuration structure and their morphisms (Definition 15, page 10) form a category .

Proof. Identity For every \( \mathcal{D} \)-structure \( \mathcal{D} = (E, C, \ell, I, m) \), \( \text{id}_\mathcal{D} : \mathcal{D} \to \mathcal{D} \) is defined to be the identity on the underlying configuration structure \( \text{id} : (E, C, \ell) \to (E, C, \ell) \) from \( C \), that trivially preserves identifiers. For any morphism \( f : \mathcal{D}_1 \to \mathcal{D}_2 \), \( f \circ \text{id}_{\mathcal{D}_1} = f = \text{id}_{\mathcal{D}_2} \circ f \) is trivial.

Associativity for \( f : \mathcal{D}_1 \to \mathcal{D}_2 \), \( g : \mathcal{D}_2 \to \mathcal{D}_3 \) and \( h : \mathcal{D}_3 \to \mathcal{D}_4 \), \( h \circ (g \circ f) = (h \circ g) \circ f \) is inherited from the associativity in \( C \), and since \( f, g \) and \( h \) all preserves identifiers.

Hence \( \mathcal{D} \) is a category. 

Lemma 6. The product of \( \mathcal{D} \)-structures of Definition 16, page 10, is the product in \( \mathcal{D} \).

Proof. First note that \( \mathcal{D}_1 \times \mathcal{D}_2 \) is a \( \mathcal{D} \)-structures as it is a configuration structure and from the definition of \( m \) every event in a configuration has a unique label. Secondly, it is easy to show that the projections are morphisms. Lastly to show that the structure \( \mathcal{D}_1 \times \mathcal{D}_2 \) has the universal property, we proceed in two steps:
the underlying configuration structure is the product of the underlying configuration structures, by definition:

\[ F(D_1 \times D_2) = F(D_1) \times F(D_2); \]

– for any \( D' \) which projects into \( D_1 \) and \( D_2 \), then \( F(D') \) projects into \( F(D_1) \) and \( F(D_2) \) and therefore there exists a unique morphism \( h : F(D') \to F(D_1 \times D_2) \). It is easy to show that since the projections preserve identifiers, then so does \( h \) which concludes our proof.

This lemma also follows from [15, Proposition 85]. \( \Box \)

**Lemma 7.** The operations of Definition 16 (product, relabeling, restriction and parallel composition), page 10, preserve \( D \)-structures.

**Proof.** Let us note that (i) the product, relabeling, restriction, and parallel composition on configuration structures from Definition 4 preserve configuration structures and that (ii) any configuration structure endowed with a valid identifier function (i.e., such that no two events in the same configuration have the same identifier, cf. Collision Freeness) is a valid \( D \)-structure.

For the product, it follows trivially from Lemma 6.

Relabeling does not change anything but the labels, so we have nothing to prove.

The restriction only removes events in configurations and keeps the identifier function intact. Hence if the initial structure has a valid identifier function, then the identifier function of the new structure is a valid one by assumption.

Let us now consider the parallel composition of two \( D \)-structures, denoted \( D_1 \upharpoonright D_2 = (E, C, \ell, m) \). Proving that the identifier function is valid follows from a case analysis. Given a configuration \( x \in C \) and two events \( e, e' \in x \), these are the possible cases:

– \( \pi_2(e) = * \) and \( \pi_2(e') = * \). In this case, looking at the definition of the product in \( D \)-structures, \( m(e) = (m_1(\pi_1(e)), m_\ast) \) and \( m(e') = (m_1(\pi_1(e')), m_\ast) \). If \( m(e) = m(e') \), then \( m_1(\pi_1(e)) = m_1(\pi_1(e')) \) in the configuration \( \pi_1(x) \) in \( D_1 \). But that’s a contradiction, since \( \pi_1(e) \) and \( \pi_1(e') \) are in the same configuration and the identifier function of \( D_1 \) is valid.

– \( \pi_1(e) = * \) and \( \pi_1(e') = * \). This case is similar as the previous one, except that it uses that the identifier function of \( D_2 \) is valid.

– \( \pi_1(e) \neq * \) and \( \pi_2(e') \neq * \) (with either \( \pi_1(e') = * \) or \( \pi_1(e') \neq * \)). If \( m(e) = m(e') \), then \( m_1(\pi_i(e)) = m_i(\pi_i(e')) \) for \( i = 1, 2 \). Then in this case,

• either one of them, say, \( e \), is a synchronisation or a fork: in this case, \( m_1(\pi_1(e)) = m_2(\pi_2(e)) = m_2(\pi_2(e')) \), and \( e' \) was relabeled \( \perp \) at the relabeling stage of the parallel composition, and then removed during the restriction. Hence a contradiction: \( e \) and \( e' \) can’t be two events in the same configuration.

• or none of them is a synchronisation, in which case both events were removed by the restriction. Hence, again, a contradiction: \( e \) and \( e' \) can’t be two events in the same configuration.
The symmetric case (where $\pi_2(e) \neq \star$ and $\pi_1(e') \neq \star$) is similar.

The following lemma makes more formal the intuition of Remark 1, page 10. Remember that we assumed that for every configuration structure $(E, C, \ell)$, $E$ was endowed with a total order, that we write $\preceq$.

**Lemma 8.** $F : \mathcal{D} \to \mathcal{C}$, defined by
- $F(E, C, \ell, I, m) = (E, C, \ell)$
- $F(f_E, f_C, f_m) = (f_E, f_C)$

and $S : \mathcal{C} \to \mathcal{D}$, defined by
- $S(E, C, \ell) = (E, C, \ell, I, m)$, where $I = \{1, \ldots, \text{Card } E\}$ and $m(e) = \begin{cases} 1 & \text{if } \forall e', e \preceq e' \\ i + 1 & \text{if } \exists e', e' \preceq e, m(e') = i \text{ and there is no } e'' \text{ s.t. } e' \preceq e'' \preceq e \end{cases}$

- For $(f_E, f_C) : (E_1, C_1, \ell_1) \to (E_2, C_2, \ell_2)$, $S(f_E, f_C) = (f_E, f_C, f_m)$, where $f_m(m_1(e)) = m_2(f_E(e_2))$.

are functors.

**Proof.** Proving that $F$ is a functor is immediate.

Proving that $S(C)$ is a $\mathcal{D}$-structure is immediate, since our construction of Collision Freeness. For $(f_E, f_C) : C_1 \to C_2$, proving that $S(f_E, f_C)$ is a morphism between $S(C_1)$ and $S(C_2)$ is also immediate. For the preservation of the identity, we compute:

$S(\text{id}_C) = S(\text{id}_E, \text{id}_C)$
$= (\text{id}_E, \text{id}_C, f_m)$

where $f_m(m(e)) = m(\text{id}_E(e)) = m(e)$, hence $f_m = \text{id}_I : I \to I$,

$= (\text{id}_E, \text{id}_C, \text{id}_I)$
$= \text{id}_{S(C)}$

For the composition of morphisms, given $f = (f_C, f_E) : C_1 \to C_2$ and $g = (g_C, g_E) : C_2 \to C_3$, we write $S(C_i) = (E_i, C_i, \ell_i, I_i, m_i)$ and we compute:

$S(g) \circ S(f) = (g_C \circ f_C, g_E \circ f_E, g_m \circ f_m)$

where, for all $e \in E_1$, we compute:

$(g_m \circ f_m)(m_1(e)) = g_m(f_m(m_1(e)))$
$= g_m(m_2(f_E(e)))$
$= m_3(g_E(f_E(e)))$

And hence we can conclude:

$S(g) \circ S(f) = S(g \circ f)$
A.2 Proofs for Sect. 3.2

In the following, we start by observing that Lemma 2 follows from Lemma 4. Lemma 4, on its side, requires a bit of work: on top of proving Lemma 3, we state and prove some intermediate lemmas (Lemma 9, 10 and 11) needed to obtain it.

Proof (Lemma 2, page 13). Follows from Lemma 4.

Lemma 9. For all RCCS process $R$, letting $|R| = (E, C, m)$, for all $e_1, e_2 \in E$, $m(e_1) = m(e_2)$ implies $e_1 = e_2$.

Proof. We proceed by structural induction on $R$. From Corollary 1 the only interesting case is the parallel composition, i.e. $R = R_1 \mid R_2$. From the definition of parallel composition in $D$-structures (Definition 16), it follows that $m(e_1) = m(e_2)$ implies $e_1 = e_2$.

Proof (Lemma 3, page 13). We proceed by induction on $R$.

If $R$ is $m \triangleright P$, we prove that $|m|$ is a poset with one maximal element by induction on $m$. The base case, if $m$ is $\emptyset$, is trivial, since $\emptyset = \emptyset$ is a poset with one maximal element. If $m$ is $\bigvee .m'$, then it follows by induction hypothesis, since $\bigvee .m' = \bigvee .m'$. If $m$ is $(i, \alpha, P).m'$, then by induction hypothesis, $[m']$ is a poset with one maximal element, and the construction of $[(i, \alpha, P).m']$ detailed in Definition 17 preserves that property.

If $R$ is $R' \setminus a$, then it trivially follows by induction hypothesis.

Finally, if $R$ is $R_1 \mid R_2$, then by induction hypothesis we get that $|R_1|$ and $|R_2|$ are both posets with a maximal configuration. We also know by Lemma 9 that they have disjoint identifiers. Looking at the definition of parallel composition for $D$-structures (Definition 16), we may observe that $|R| = |R_1| \mid |R_2| = (R_1 \mid R_2) = (r \circ ([R_1] \times [R_2]))|_F$.

We show that there exists more than one maximal configurations in $|R_1| \times |R_2|$ and that all but one are removed by the restriction.

We show this by first showing that there exists more than one maximal configurations in $F([R_1]) \times F([R_2])$, denoted here with $C$. From the definition of product (Definition 4), we have that there exists $y_1, \ldots, y_n$ maximal configurations in $C$ such that $\pi_1(y_i)$ and $\pi_2(y_i)$ are maximal in $C_1$ and $C_2$, respectively. As $|R_1| \times |R_2| = (F([R_1]) \times F([R_2])) + m$ it follows that the maximal configurations of $C$ are preserved in $|R_1| \times |R_2|$.

A second step is then to show that the restriction keeps only one maximal configuration. Let $y_i, y_j$ be two maximal configurations. As they are maximal it implies that $y_i \cup y_j \notin C$, for $i \neq j \leq n$. In turn, this implies that there exists $e_i \in y_i$ and $e_j \in y_j$ such that $\pi_1(e_i) = \pi_1(e_j)$ and $e_i \neq e_j$, as otherwise $y_i \cup y_j$ would be defined. Here we assume that $\pi_1(e_i) = \pi_1(e_j)$ but we could also take $\pi_2(e_i) = \pi_2(e_j)$ and the argument still holds.

Let us now take $d$ an event in $C_1$ and take $e_1, \ldots, e_m$ the subset of events in $E$ where $\pi_1(e_i) = d$. The restriction in the parallel composition of $|R_1| \mid |R_2|$ keeps only one such event $e_i$ and removes the rest. Therefore, from all maximal configurations $y_1, \ldots, y_m$ such that $e_i \in y_i$, $i \leq m$, only one remains.
By applying the argument above to all events in \([R_1]\) (and \([R_2]\)), we have that the restriction removes all but one \(y_i\), which is then the maximal configuration in \([R_1] \mid [R_2]\).

\[\square\]

Let us write \(x_{\text{MAX}}\) the maximal configuration from Lemma 3.

For the following proof, we need to introduce the causality relation on memory events and on transitions from [7]. We write \(i_1 <_R i_2\) for \(R\) a process and \(i_1, i_2 \in I\) if there exist a memory stack \(m.(i_1, \alpha_1, P_1), m'.(i_2, \alpha_2, P_2), m''\) in \(R\). We write \(i_1 \leq_R i_2\) for the transitive closure of \(<\) over all memories of \(R\).

**Lemma 10.** Let \(R\) be a reversible process and \([R] = (E, C, \ell, m)\) be the encoding of its memory, with \(x_{\text{MAX}}\) the maximal configuration in \(C\). Let \((i_1, \alpha_1, P_1)\) and \((i_2, \alpha_2, P_2)\), \(i_1 \neq i_2\) be two memory events in \(R\) and let \(e_1, e_2 \in E\) be the two corresponding events, i.e. \(m(e_j) = i_j, j \in \{1, 2\}\). Then \(i_1 \leq_R i_2 \iff e_1 \leq_{x_{\text{MAX}}} e_2\).

**Proof.** Follows by a structural induction on \(R\).

We also import from [7, Definition 1] the definition of causality on transitions \(t_1\) and \(t_2\), of a trace \(\theta\), that we denote here with \(t_1 \leq_{\theta} t_2\). We do not give the definition formally, as it is lengthy, but intuitively, it is the causality relation on memory events lifted to transitions.

Let \(R\) be an RCCS process and \(\theta : O_R \rightarrow R\) a trace. Also, let \([O_R]\) = \((E, C, \ell)\) and let \(x\) be a configuration such that \([R] = ([O_R], x)\). We define a bijection from transitions in \(\theta\) to events in \(x\), that we write \(\text{TE}\). We define the bijection by induction on the trace \(\theta\) using the fact for each transition \(t : R' \xrightarrow{\ell, \alpha} R''\) in \(\theta\), there exists \(x'\) a configuration and \(e'\) an event in \([O_R]\) such that \([R'] = ([O_R], x')\) and \([R''] = ([O_R], x' \cup \{e'\})\). Then \(\text{TE}(t) = e'\).

**Lemma 11.** Let \(t_1\) and \(t_2\) be two transitions of a trace \(\theta : O_R \rightarrow R\) and let \([R] = ([O_R], x)\). Then \(t_1 \leq_{\theta} t_2 \iff \text{TE}(t_1) \leq_x \text{TE}(t_2)\).

**Proof.** Follows by induction on the trace \(\theta\).

**Proof (Lemma 4, page 14).** We reformulate the hypothesis and show a stronger (in the sense that it is more specific) result from which Lemma 4 follows.

As \(R\) is reachable there exists a forward-only trace \(O_R \xrightarrow{i_1, \alpha_1} \cdots \xrightarrow{i_n, \alpha_n} R\) [1, Lemma 1], denoted by \(\theta\). We write \([R_j]\) = \((E_j, C_j, \ell_j, m_j)\) for \(R_j\) a process in the trace above and \(\theta_j : O_R \xrightarrow{i_1, \alpha_1} \cdots \xrightarrow{i_j, \alpha_j} R_j\) a subtrace of \(\theta\).

Let \([O_R]\) = \((E, C, \ell)\). From [1] we have that \([R_j]\) = \(([O_R], x_j)\) for \(x_j \in C\).

We show that there exists three bijections:

- \(\text{TE}_j\) between transitions in the trace \(\theta_j\) and events in \([O_R]\), i.e. \(\text{TE}_j : \theta_j \rightarrow E\);
- \(\text{MT}_j\) between events in \([R_j]\) and transitions, i.e. \(\text{MT}_j : E_j \rightarrow \theta_j\);
- \(\text{ME}_j\) between events in \([R_j]\) and events in \([O_R]\), i.e. \(\text{ME}_j : E_j \rightarrow E\), such that \(\text{ME}_j = \text{TE}_j \circ \text{MT}_j\).
Moreover, all three bijections preserve the labels and the causality relations. In particular, $\me$ is label and order preserving for all events in $x_{\text{MAX}}$, the maximal configuration in $[R_j]$: \begin{align}
\me_j(x_{\text{MAX}}) &= x_j \quad (10) \\
e_1 \leq x_{\text{MAX}} e_2 &\iff \me_j(e_1) \leq x_j \me_j(e_2). \quad (11)
\end{align}

From Definition 18, $x \downarrow = (x, C_x, \ell|x)$, for $C_x = \{y \mid y \in C, y \subseteq x\}$. We can write $[R_j] = x_{\text{MAX}} \downarrow$, and therefore have $\mathcal{F}([R_j]) \cong (\me_j(x_{\text{MAX}})) \downarrow$.

We proceed by induction on the trace $\theta$, and at each step we extend the three bijections above.

For a transition $R_j \xrightarrow{i,\alpha} R_{j+1}$ and a trace $\theta_j$ we have by induction that

- $\mathcal{F}([R_j]) \cong x_j \downarrow$, for $[R_j] = ([O_R], x_j)$;
- there exists $\text{TE}_j : \theta_j \rightarrow E$, $\text{MT}_j : E_j \rightarrow \theta_j$ and $\me_j : E_j \rightarrow E$

as defined above.

There is an operational correspondence between $R_j$ and its encoding [1, Lemma 6], and therefore there exists $e$ an event in $[[O_R]]$ such that

\[(([O_R], x_j) \xrightarrow{e} ([O_R], x_j \cup \{e\}) \quad (12)
\]

with $\ell(e) = \alpha$ and where we write $x_{j+1} = x_j \cup \{e\}$.

We have to show that $\mathcal{F}([R_{j+1}]) \cong x_{j+1} \downarrow$. More specifically we show that $E_{j+1} \setminus E_j = \{e_{j+1}\}$ and that we can extend the bijections such that

\[
\begin{align}
\text{TE}_{j+1} &= \text{TE}_j \cup \{(R_j \xrightarrow{i,\alpha} R_{j+1}) \mapsto e\} \quad (13) \\
\text{MT}_{j+1} &= \text{MT}_j \cup \{e_{j+1} \mapsto (R_j \xrightarrow{i,\alpha} R_{j+1})\} \quad (14) \\
\me_{j+1} &= \me_j \cup \{e_{j+1} \mapsto e\} \quad (15)
\end{align}
\]

As $\me_{j+1}$ is a label and order preserving bijection on $x_{\text{MAX}}$, the maximal configuration in $[R_{j+1}]$, it follows that $\me_{j+1}(x_{\text{MAX}}) = x_{j+1}$. As $[R_{j+1}] \cong x_{\text{MAX}} \downarrow$ it follows $\mathcal{F}([R_{j+1}]) \cong x_{j+1} \downarrow$.

We now proceed by cases on the transition $R_j \xrightarrow{i,\alpha} R_{j+1}$. We distinguish two cases: $\alpha$ is an unsynchronized input or output, and $\alpha = \tau$.

- Let us suppose w.l.o.g. that $\alpha = a$. Then we can rewrite the transition as follow:

$$R_j = (\overrightarrow{b})(S \mid m \triangleright a.P + Q) \xrightarrow{i,\alpha} R_{j+1} = (\overrightarrow{b})(S \mid \langle i, \alpha, Q \rangle m \triangleright P)$$

Let us also define the following projection on events: $\pi_S(e) = e_s$ if there exists $e_s \in [S]$ such that $e, e_s$ have the same identifiers and undefined otherwise. Similarly define $\pi_m$ for the projections of events from $[R_j]$ to $[m]$.

Note that we can extend $\text{TE}_{j+1}$ as in Equation 13. We show that there exists an event in $[R_{j+1}]$ which corresponds to the transition $R_j \xrightarrow{i,\alpha} R_{j+1}$. Let
Lemma 3 and we there exists a memory $\text{Lemma 10}$ which shows that we can unfold we have that there $\text{Definition 17}$ a, $Q$, consider $15$ Equation 11 it follows that. To show $\text{Suppose that}$ we are assuming here, for simplification, that both thread involved in the restriction. The event $(\star, e)$ removed by the second restriction as $\text{we reason for}$ similarly, we reason for $\text{have then that}$ $\text{Lemma 10}$ we have that there exists a single maximal configuration in $C_m$, denoted with $x_{\text{max}}^m$. Using Definition 17 we can unfold $[(i, \alpha, Q).m]$ and write $\left[ R_j \right] = (r \circ ([S] \times (E_m, C_m, \ell_m, m_m))) \upharpoonright \downarrow \varnothing$ $\left[ R_{j+1} \right] = (r \circ ([S] \times (E_m, C_m, \ell_m, m_m))) \upharpoonright \downarrow \varnothing$ \begin{align*} \left[ R_j \right] &= (r \circ ([S] \times (E_m, C_m, \ell_m, m_m))) \upharpoonright \downarrow \varnothing \\ \left[ R_{j+1} \right] &= (r \circ ([S] \times (E_m, C_m, \ell_m, m_m))) \upharpoonright \downarrow \varnothing \end{align*} for some event $e_m \notin E_m$. From rules act. and par. of Fig. 4, $i$ is not in the domain of $m_m$. Therefore all synchronisations in $[S] \times [(i, \alpha, Q).m]$ of the form $(e_s, e_m)$, with $e_s \in [S]$, are relabeled $\bot$ and removed by the first restriction. The event $(\star, e_m)$ is preserved by the first restriction. It is not removed by the second restriction as $a \notin \overline{b}$. Remember that from Lemma 3 we have that there exists a single maximal configuration in $[R_{j+1}]$ and from the definition of the parallel composition there is only one event in $x_{\text{max}}$ with the first projection equal to $e_m$, denoted $e_{j+1}$: $e_{j+1} \in x_{\text{max}}$ with $\pi_m(e_{j+1}) = e_m$. It follows that $E_{j+1} = E_j \cup \{e_{j+1}\}$. We extend then the bijections as in Equations 14 and 15. Moreover, $\ell(e_{j+1}) = \ell_m(e_m)$, and from Equation 12 it follows that $\text{ME}_{j+1}$ is label preserving. The last part is to show Equation 11. We only have to show that $$e' \leq_{x_{\text{max}}} e_{j+1} \iff \text{ME}_{j+1}(e') \leq_{x_{j+1}} \text{ME}_{j+1}(e_{j+1}),$$ (16) as the rest follows by induction on $\text{ME}_j$ and from Equation 15. To show Equation 16, consider $e' \leq_{x_{\text{max}}} e_{j+1}$. From Definition 17 there exists a memory event $d = (i', \alpha', P')$ in $[R_{j+1}]$ such that $m_j(e') = i'$. From Lemma 10 we have then that $i' \leq_R i$ and using the definition of causality on transitions ([7, Definition 1]), $\text{MT}_{j+1}(e') \leq_{\emptyset} \text{MT}_{j+1}(e_{j+1})$. We conclude using Lemma 11 which shows that $\text{TE}_{j+1}(\text{MT}_{j+1}(e')) \leq_{x_{j+1}} \text{TE}_{j+1}(\text{MT}_{j+1}(e_{j+1}))$. Similarly, we reason for $e'$ concurrent with $e_{j+1}$. Lastly, note that there are no events in conflict with $e_{j+1}$ (or with $e$ in $x_{j+1}\downarrow$), as there is a single maximal configuration in both $[R_{j+1}]$ and in $x_{j+1}\downarrow$. - Suppose that $\alpha = \tau$ and let us write the transitions as follows: $$R_j = (\overline{b})(S \mid m_1 \triangleright a.P_1 + Q_1 \mid m_2 \triangleright \overline{\pi}.P_2 + Q_2)$$ $$\xrightarrow{i, \tau} R_{j+1} = (\overline{b})(S \mid (i, \alpha, Q_1).m_1 \triangleright P_1 \mid (i, \alpha, Q_2).m_2 \triangleright P_2)$$ We are assuming here, for simplification, that both thread involved in the synchronisation are under the same set of restricted names. The more general case, does not change the reasoning here, just adds in technicality.
We show that the transition adds a single event \( e_{j+1} \) in \( [R_{j+1}] \) and that the bijection \( \alpha \) defined on \( [R_j] \) extends to \( e_{j+1} \) such that it remains a label and order preserving bijection between the maximal configuration in \( [R_{j+1}] \) and \( x_{j+1} \). The proof follows the reasoning above.

\[ \square \]

### A.3 Proof of Theorem 2

Before proving the main theorem, let us make the following observation. Let \( f : (E_1, C_1, \ell_1) \rightarrow (E_2, C_2, \ell_2) \) be a morphism, which is defined as function on events: \( f = (f_E, f_C) \). As \( f_C \) is defined by \( f_E \), we can w.l.o.g. write \( f = f_E \).

Saying that \( f \) is an isomorphism is equivalent then to saying that \( f : E_1 \rightarrow E_2 \) is a label- and order-preserving bijection. Therefore the functions \( f \) of Definition 9 and Definition 20 are of “the same nature”.

**Proof (Theorem 2, page 15).** Let us prove the HHPB case, the other three cases being similar, and actually simpler.

\[
\Rightarrow \text{Let } \mathcal{R}_{\text{RCCS}} \text{ be a HHPB between } \emptyset \rhd P_1 \text{ and } \emptyset \rhd P_2. \text{ We show that the following relation}
\]

\[
\mathcal{R} = \{(x_1, x_2, f) \mid x_1 \in [P_1], x_2 \in [P_2], \exists R_1, R_2 \text{ s.t. } O_{R_1} = P_1,
\]

\[
O_{R_2} = P_2, (R_1, R_2, f) \in \mathcal{R}_{\text{RCCS}} \text{ and }
\]

\[
[R_1] = ([P_1], x_1), [R_2] = ([P_2], x_2)\}
\]

is a HHPB between \( [P_1] \) and \( [P_2] \).

First note that \( (\emptyset, \emptyset, \emptyset) \in \mathcal{R} \): indeed \( (\emptyset \rhd P_1, \emptyset \rhd P_2, \emptyset) \in \mathcal{R}_{\text{RCCS}} \) and \( [\emptyset \rhd P_1] = ([P_1], \emptyset) \), for \( i \in \{1, 2\} \).

Let us suppose that \( (x_1, x_2, f) \in \mathcal{R} \) for \( [R_i] = ([P_i], x_i) \), for \( i \in \{1, 2\} \) and \( f : x_1 \rightarrow x_2 \) an isomorphism. Moreover, note that \( \mathcal{F}([R_i]) \cong x_i \downarrow \) from Lemma 4, and that \( [R_i] \cong x_i \downarrow + m \), for some function \( m \), from Definition 14.

To show that \( \mathcal{R} \) is a HHPB we have to show that if \( x_1 \xrightarrow{\ell_1} y_1 \) (or \( x_1 \xrightarrow{c_1} y_1 \)) then there exists \( y_2 \) such that \( x_2 \xrightarrow{\ell_2} y_2 \) (or \( x_2 \xrightarrow{c_2} y_2 \)) respectively and such that \( (y_1, y_2, f') \in \mathcal{R} \) for some \( f' \).

Let \( x_1 \xrightarrow{c_1} y_1 \), hence by definition, \( y_1 = x_1 \cup \{e_1\} \). From the correspondence between RCCS and their encodings (from \([1, \text{Lemma 6}])\), it follows that \( R_1 \xrightarrow{\ell_1} S_1 \) such that \( [S_1] = ([P_1], y_1) \). We therefore deduce that \( [S_1] \cong y_1 \downarrow + (m \cup \{e_1 \mapsto i\}) \).

As \( (R_1, R_2, f) \in \mathcal{R}_{\text{RCCS}} \) and as \( R_1 \xrightarrow{\ell_1} S_1 \), it follows that there exists a transition \( R_2 \xrightarrow{\ell_2} S_2 \) with \( f = f' \mid {\mathcal{F}([R_1])} \) and \( (S_1, S_2, f') \in \mathcal{R}_{\text{RCCS}} \). Again from the correspondence between \( R_2 \) and \( [R_2] \) we have that \( x_2 \xrightarrow{e_2} y_2 \) such that \( y_2 = x_2 \cup \{e_2\} \) and \( [S_2] = ([P_2], y_2) \). Then we have that \( (y_1, y_2, f') \in \mathcal{R} \).

We treat similarly the cases where \( x_2 \) does a transition, or when the transitions are backwards.
Let $R_{\text{CONF}}$ be a HHPB between $\llbracket P_1 \rrbracket$ and $\llbracket P_2 \rrbracket$. We show that the following relation

$$R = \{ (R_1, R_2, f) \mid O_{R_1} = P_1, O_{R_2} = P_2 \text{ and } \llbracket R_1 \rrbracket = ([P_1], x_1), \llbracket R_2 \rrbracket = ([P_2], x_2), \text{ with } (x_1, x_2, f) \in R_{\text{CONF}} \}$$

is a HHPB between $\emptyset \rhd P_1$ and $\emptyset \rhd P_2$.
We have that $(\emptyset \rhd P_1, \emptyset \rhd P_2, \emptyset) \in R$ as $(\emptyset, \emptyset, \emptyset) \in R_{\text{CONF}}$ and $[\emptyset \rhd P_i] = ([P_i], \emptyset)$, for $i \in \{1, 2\}$.

We suppose now that $(R_1, R_2, f) \in R$, with $f : F(\lfloor R_1 \rfloor) \rightarrow F(\lfloor R_2 \rfloor)$. It implies that $[R_i] = ([P_i], x_i)$, $i \in \{1, 2\}$, we have that $(x_1, x_2, f) \in R_{\text{CONF}}$.

To show that $R$ is a HHPB we have to show that if $R_1 \overset{i\alpha}{\rightarrow} S_1$ (or $R_1 \overset{\sim\alpha}{\rightarrow} S_1$) then there exists $S_2$ such that $R_2 \overset{j\alpha}{\rightarrow} S_2$ (or $R_2 \overset{\sim\alpha}{\rightarrow} S_2$ respectively) and such that $(S_1, S_2, f') \in R$ for some $f'$.

Let $R_1 \overset{i\alpha}{\rightarrow} S_1$. We use again the correspondence between RCCS and their encodings (from [1, Lemma 6]) from which we have that there exists $e_1$ and $y_1 = x_1 \cup \{e_1\}$ such that $x_1 \overset{e_1}{\rightarrow} y_1$ and $[S_1] = ([P_1], y_1)$. As $(x_1, x_2, f) \in R_{\text{CONF}}$ it implies that there exists $e_2, y_2$ and $f'$ such that $x_2 \overset{e_2}{\rightarrow} y_2$ and $(y_1, y_2, f') \in R_{\text{CONF}}$. Again, from the correspondence between RCCS and configuration structures we have that from $x_2 \overset{e_2}{\rightarrow} y_2$, there exists $S_2$ and some $j$ such that $R_2 \overset{j\alpha}{\rightarrow} S_2$ with $[S_2] = ([P_2], y_2)$. Hence $F([S_2]) = y_2 \downarrow$.

We conclude therefore that $(S_1, S_2, f') \in R$.

Similarly we show the cases where $R_1$ does a backward transition, or if $R_2$ does a forward or backward transition.

\[\square\]