Contextual equivalences in configuration structures and reversibility

Clément Aubert, Ioana Cristescu

Contextual equivalences in configuration structures and reversibility[☆,☆☆]

Clément Aubert[a,b,1,*], Ioana Cristescu[c,**]

[a] *INRIA*
[b] *Université Paris-Est, LACL (EA 4219), UPEC, F-94010 Créteil, France*
[c] *Diderot, Sorbonne Paris Cité, P.P.S., UMR 7126, F-75205 Paris, France*

Abstract

Contextual equivalence equate terms that have the same observable behaviour in any context. A standard contextual equivalence for CCS is the strong barbed congruence. Configuration structures are a denotational semantics for processes in which one define equivalences that are more discriminating, i.e. that distinguish the denotation of terms equated by barbed congruence. Hereditary history preserving bisimulation (HHPB) is such a relation. We define a strong back-and-forth barbed congruence on RCCS, a reversible variant of CCS. We show that the relation induced by the back-and-forth congruence on configuration structures is equivalent to HHPB, thus providing a contextual characterization of HHPB.

Keywords: Formal semantics, Process algebras and calculi, Reversible CCS, Hereditary history preserving bisimulation, Strong barbed congruence, Contextual caracterisation

Introduction

Reversibility

Being able to reverse a computation is an important feature of computing systems. Reversibility is a key aspect in every system that needs to achieve

[☆] Extended version of a work presented at ICE 2015 [1]. A part of this work appears, with much more contextual material, in the second author's Ph.D Thesis [2].

[☆☆] This work was partly supported by the ANR-14-CE25-0005 ELICA and the ANR-11-INSE-0007 REVER.

[*] Corresponding author
[**] Principal Corresponding author
Email addresses: aubertc@appstate.edu (Clément Aubert), ioana.cristescu@pps.univ-paris-diderot.fr (Ioana Cristescu)
URL: http://lacl.fr/~caubert/ (Clément Aubert), http://www.pps.univ-paris-diderot.fr/~ioana/ (Ioana Cristescu)
[1] Present address: Department of Computer Science, Appalachian State University, Boone, NC 28608, USA.

distributed consensus [3] to escape local states where the consensus cannot be found. In such problems, multiple computing agents have to reach a common solution. Allowing independent agents to backtrack and explore the solution space enables them to reach a globally accepted state if given enough time and if a common solution exists. For example, the dining philophers problem [4] requires a backtracking mechanism to prevent deadlocks. Rewinding a computation step by step is also a common way to debug programs. In such settings the step by step approach is often more useful than restarting the program from an initial state.

Importantly, the backtracking mechanism can be integrated to the operational semantics of a programming language, instead of adding a tailor-made implementation on top of each program. A formal model for reversible concurrent systems needs to address two challenges at the same time: (i) how to compute without forgetting and (ii) what is an optimal notion of legitimate backward moves. Roughly speaking, the first point is about syntax: processes need to carry a memory that keeps track of everything that has been done (and of the choices that have not been made). Importantly the needed information to backtrack is recorded in a *distributed* fashion instead of using a centralized store, which could be a bottleneck for the computation. The second point is tied to the choice of the computation's semantics. In a sequential program, one backtracks computations in the opposite order to the execution. However, in a concurrent setting, we do not want to undo the actions precisely in the opposite order than the one in which they were executed, as this order may not materialise. The concurrency relation between actions has to be taken into account. It can be argued that the most liberal notion of reversibility is the one that just respects causality: an action can be undone precisely after all the actions that causally depend on it have also been undone. Then an acceptable backward path is *causally consistent* with the forward computation.

There are different accounts of reversible operational semantics, RCCS [5, 6] and CCSK [7] being the two main propositions for a reversible CCS. In these works, reversibility is embedded into a (classical) process calculus.

Causal models

In interleaving models, the internal relations between different events cannot be observed. In particular, causality is not treated as a primitive concept. On the other hand, non-interleaving semantics have a primitive notion of *concurrency* between computation events. As a consequence one can also derive a *causality* relation, generally defined as the complement of concurrency. These models are therefore sometimes called *true-concurrent* or *causal* or, if causality is represented as a partial order on events, *partial order* semantics.[2]

[2]Event and configuration structures were introduced to define domains for concurrency [8]. Causal models are thus often, but inaccurately, called *denotational*: a denotational interpretation is supposed to be invariant by reductions, a property that event structures do not have.

A causal model is often an alternative representation of an existing interleaving semantics that helps in understanding the relations between computations in the latter. Usually in such models, sets of events are considered computational states. Each set, called a *configuration*, represents a reachable state in the run of the process. The behaviour of a system is encoded as a collection of such sets. The set inclusion relation between the configurations stands for the possible paths followed by the execution. Concurrency and causality are derivable from set inclusion. In their generality, such models are called *configuration structures* [9], they are a syntax-free and causal model that can interpret multiple calculi.

Stable families [10] are configuration structures equipped with a set of axioms, that capture the intended behaviour of a CCS process. Morphisms of stable families capture sub-behaviours of processes and form a category of stable families. Process combinators correspond then to universal constructions in this category. The correspondence with CCS is established through an operational semantics defined on stable families, that we abusively name in that context configurations structures as well.

Behavioural equivalence

Behavioural equivalences are a major motivation in the study of formal semantics. For instance, one wants to verify that the execution of a program satisfies its expected behaviour, or that binaries obtained from the same source code, but with different compilation techniques, behave the same. Thus the interesting equivalences equate terms that behave the same. Moreover the equivalence should be a congruence: two processes are equivalent if they behave similarly in any context. Loosely speaking it aims at identifying process that have a common external behaviour in any environment.

Equivalences defined on reduction semantics are often hard to prove. A proof technique in this case is to define a LTS-based equivalence that is equivalent with the reduction-based one and carry the proofs in LTS semantics.

Behavioural equivalences are defined on the operational semantics and thus cannot access the structure of a term. The observations one do during the execution of a process are called the *observables* of the relation. For instance one observes whether the process terminates or whether it interacts with the environment [11].

Causality and reversibility

Causality and reversibility are tightly connected notions [5, 12]. Causal consistency is a correctness criterion for reversible computations. Therefore whenever a reversible semantics is proposed, the calculus has to be equipped first with a causal semantics.

Prime LTS are known [13] to generates a prime event structure. Since a specific reversible LTS [7] is indeed prime, and moreover since the forward and backward reductions correspond to reductions in its causal representation, reversible models and causal ones are easily derivable from each other.

Notably the connection between reversibility and causality is useful to define meaningful reversible equivalences. Causal equivalences are more discriminating

than the traditional operational ones. However on a reversible operational semantics one define equivalences of the same expressivity. Causal equivalences have been extensively studied [14–17]. Of particular interest is the hereditary history preserving bisimulation, which was shown to correspond to a LTS-based equivalence for a reversible CCS [7].

Equivalences on configuration structures

In CCS equivalences are defined only on forward transitions and are therefore inappropriate to study reversible processes.

A reversible bisimulation [18] is more adapted but it is not contextual. We introduce a contextual equivalence on RCCS by adapting the notions of contexts and barbs to the reversible setting. The resulting relation, called *barbed back-and-forth congruence* is defined similarly to the barbed congruence of CCS except that the backwards reductions are also observed.

Configuration structures provide a causal semantics for CCS. Equivalences on configuration structures are more discriminating than the ones on the operational setting. It is possible to move up and down in the lattice, whereas in the operational semantics, only forward transitions have to be simulated. As an example, consider the processes $a \mid b$ and $a.b + b.a$ that are bisimilar in CCS but whose causal relations between events differ.

In particular we are interested in hereditary history preserving bisimulation (HHPB) in Definition 23, which equates configuration structures that simulate each others' forward and backward moves. Phillips and Ulidowski [14] showed that the back-and-forth bisimulation corresponds to HHPB, that can be defined in an operational setting thanks to reversibility. Allowing both forward and backward transitions gives to the operational world the discriminating power of causal models. We show that HHPB also corresponds to a congruence on RCCS, the barbed back-and-forth congruence. It is the a contextual characterisation of HHPB which implies a contextual equivalences in configuration structures.

Outline

We begin by recalling notions on LTS and CCS, as well as their so-called reversible variants (Sect. 1). RCCS (Sect. 1.2) is then proven to be a conservative extension of CCS over the traces: their is a strong bisimulation between a reversible process and a "classical", memory-less, process (Lemma 3). Lastly, we adapt the usual CCS notions of contexts, barbs, and barbed congruence to RCCS (Sect. 1.3), thus introducing the back-and-forth barbed congruence (Definition 13).

We next introduce the interpretation of reversible process on configuration structures (Sect. 2). We recall the classical definitions (Sect. 2.1) as well as the encoding of CCS terms in configuration structures (Sect. 2.2). Encoding of RCCS terms is built on top of it (Sect. 2.3), and an operational correspondence between reversible processes and their interpretations is proven (Lemma 6).

Finally, we introduce a notion of context for configuration structures (Sect. 3.1) and study the relation induced on configuration structures by the barbed back-and-forth congruence (Sect. 3.2). In Sect. 3.3 we define the hereditary history

preserving bisimilarity and provide a characterisation by inductive relations. Lastly, we show in Sect. 3.4 that HHPB is a congruence (Proposition 8) and that whenever two configuration structures are barbed back-and-forth congruent, they also are hereditary history preserving bisimilar (Theorem 2).

Our main contribution is proving that barbed congruence in RCCS corresponds to hereditary history preserving bisimulation, which is defined on configuration structures As a consequence, it provides a contextual characterization of equivalences defined in non-interleaving semantics.

Limitations

Our work is restrained to processes that forbid "auto-concurrency" and "auto-conflict" (Remark 4). We do not cover recursion, though a treatment of recursion in configuration structures exists [10]. "Irreversible" action is a feature of RCCS [5] that is absent of our work.

We tried to stick to canonical notations and to remind of common definitions. However, we consider the reader familiar with the syntax, congruence relation and reduction rules of CCS. If not, a quick glance at a textbook [19] or at lectures notes [20] should help the reader uneasy with them.

1. Contextual equivalences in reversibility

Reversibility provides an implicit mechanism to undo computations. Interleaving semantics use a Labeled Transition System (LTS) to represent computations, henceforth refered to as the *forward* LTS. In a reversible semantics a second LTS is defined that represents the *backward* moves (Sect. 1.1).

RCCS [5, 21, 22] (Sect. 1.2) is a reversible variant of CCS, that allows computations to *backtrack*, hence introducing the notions of *forward* and *backward* transitions. Memories attached to processes store the relevant information to eventually do backward steps. Without this memory, RCCS terms are essentially CCS terms (Lemma 3), but their presence forces to be precise when defining contexts and contextual equivalence for the reversible case (Sect. 1.3).

1.1. (Reversible) labelled transition systems

A labelled transition system is a multi-graph where the nodes are called *states* and the edges, *transitions*. Transitions are labelled by *actions* and may be fired non-deterministically.

Definition 1 (Labelled Transition System)**.** A *labelled transition system* is a tuple $(\rightarrow, S, \mathrm{Act})$ made of a set S of *states*, a set Act of *actions* (or labels) and a relation $\rightarrow \subseteq S \times \mathrm{Act} \times S$.

For $s, s' \in S$ and $a, b \in \mathrm{Act}$, we write $s \xrightarrow{a} s'$ for $(s, a, s') \in \rightarrow$ and $s \rightarrow s'$ if $s \xrightarrow{a} s'$ for some $a \in \mathrm{Act}$.

Elements $t : s \xrightarrow{a} s'$ of $\longrightarrow$ are called transitions. Two transitions, t and t' are *composable*, written $t; t'$, if the target of t is the source of t'. The empty trace is denoted ϵ.

Definition 2 (Trace)**.** A *trace*, denoted by $\sigma : t_1; \dots; t_n$ is a sequence of composable transitions. Except for the empty trace, all traces have a source and a target.

Define $\longrightarrow^\star \subseteq S \times \mathrm{Act}^\star \times S$ the *reachability* relation as follows:

$$s \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} s' \iff \exists t_1, \dots, t_n \text{ and } s_1, \dots, s_{n+1} \text{ such that } t_i : s_i \xrightarrow{\alpha_i} s_{i+1} \text{ and } s_1 = s, s_{n+1} = s'.$$

We say in that case that s' is *reachable* from s, that s' is a *derivative* of s, and that s is an *ancestor* of s'.

Definition 3 (Reversible LTS)**.** Given $(\longrightarrow, S, \mathrm{Act})$ and $(\rightsquigarrow, S, \mathrm{Act})$ two labelled transition systems defined on the same set of states and actions, we define $(\twoheadrightarrow, S, \mathrm{Act})$ a third LTS by taking $\twoheadrightarrow = \longrightarrow \cup \rightsquigarrow$. By convention, a transition $s \longrightarrow t$ is said to be *forward*, whereas a transition $t \rightsquigarrow s$ is said to be *backward*. In $t \rightsquigarrow s$, s is an *ancestor* of t.

A variety of semantically different backtracking mechanisms exists, for instance,

- taking $\rightsquigarrow = \emptyset$ models a language with only irreversible moves,
- in a sequential setting, if $\longrightarrow$ draws a tree, taking $\rightsquigarrow = \{(t, a, s) \mid s \xrightarrow{a} t\}$ forces the backward traces to follow exactly the forward execution.

In concurrency, backward traces are allowed if their source and target are respectively the target and source of a forward trace.

1.2. Reversible CCS

A RCCS term, also called a *monitored process*, is a CCS process equipped with a memory. A *thread* is a CCS term P guarded by a memory m and denoted $m \rhd P$. Processes can be composed of multiple threads. The memory acts as a stack for the previous computations. Each entry in the memory is called a *(memory) event* and has a unique identifier. The forward transitions push events to the memories while the backward moves pop them out.

Definition 4 (Names, labels and actions)**.** We define $\mathsf{N} = \{a, b, c, \dots\}$ to be the set of *names* and $\bar{\mathsf{N}} = \{\bar{a}, \bar{b}, \bar{c}, \dots\}$ its *co-names*. The complement of a (co-)name is given by a bijection $\bar{[\cdot]} : \mathsf{N} \to \bar{\mathsf{N}}$, whose inverse is also denoted by $\bar{[\cdot]}$, so that $\bar{\bar{a}} = a$.

A *synchronisation* is a pair of names that complement each other, as $(a, \bar{a})$, and that is denoted with the special symbol τ, whose complement is undefined.

Actions are labelled using the set $\mathsf{L} = \mathsf{N} \cup \bar{\mathsf{N}} \cup \{\tau\}$ of (event) labels defined by the following grammar:

$$\mathsf{N} \cup \bar{\mathsf{N}} : \lambda, \pi ::= a \parallel \bar{a} \parallel \dots \qquad \text{(CCS prefixes)}$$

$$\mathsf{L} : \alpha, \beta ::= \tau \parallel a \parallel \bar{a} \parallel \dots \qquad \text{(Event labels)}$$

As it is common, we will sometimes use a and b to range over names, and call the set of names and co-names simply the set of names.

Transitions in both directions are decorated by the identifier of the associated event. Identifiers on the (partial) events are used to remember their synchronisation partners. Thus to combine into a τ, the transitions need complementary labels and the same identifier.

Grammar. Consider the following *process constructors*, also called *combinators* or *operators*:

$$\begin{aligned} e &\coloneqq \langle i, \alpha, P\rangle && \text{(memory events)}\\ m &\coloneqq \varnothing \parallel \Upsilon\, .m \parallel e.m && \text{(memory stacks)}\\ P, Q &\coloneqq \lambda.P \parallel P \mid Q \parallel \lambda.P + \pi.Q \parallel P\backslash a \parallel 0 && \text{(CCS processes)}\\ R, S &\coloneqq m \rhd P \parallel R \mid S \parallel R\backslash a && \text{(RCCS processes)} \end{aligned}$$

A (memory) event $e = \langle i, \alpha, P\rangle$ is made of:

- An event identifier $i \in \mathsf{I}$ that *tags* transitions. We may think of them as `pid`, in the sense that they are a centrally distributed identifier attached to each transition.
- A label α that marks which action has been fired (in the case of a forward transition), or what action should be restored (in the case of a backward move).
- A backup of the whole process P that has been erased when firing a sum, or 0 otherwise.

In the memory stack, the fork symbol Υ marks a parallel composition. The memory is then copied in two local memories, as despicted in the congruence rule called "distribution memory" in Definition 5).

Lastly the null process, denoted 0, cannot perform any transition. We will often omit it, so for example we write $a \mid b$ instead of $a.0 \mid b.0$.

Notations 1.

- We use $\mathbb{N}$ for the set of *event identifiers* I and let i, j, k range over elements of I. Forward and backward transitions will be tagged with such identifiers, and so we write $\xrightarrow{i:\alpha}$ and $\overset{i:\alpha}{\rightsquigarrow}$. We use $\overset{i:\alpha}{\twoheadrightarrow}$ as a wildcard for $\xrightarrow{i:\alpha}$ or $\overset{i:\alpha}{\rightsquigarrow}$, and if there are indices $i_1, \ldots, i_n$ and labels $\alpha_1, \ldots, \alpha_n$ such that $R_1 \overset{i_1:\alpha_1}{\twoheadrightarrow} \cdots \overset{i_n:\alpha_n}{\twoheadrightarrow} R_n$, then we write $R_1 \twoheadrightarrow^\star R_n$. We sometimes omit the identifier or the label in the transition.
- For R a reversible process and m a memory, we denote $\mathsf{I}(m)$ (resp. $\mathsf{I}(R)$) the set of identifiers occurring in m (resp. in R).
- The sets $\mathrm{nm}(R)$ of names in R, $\mathrm{fn}(R)$ of free names in R and $\mathrm{bn}(R) = \mathrm{nm}(R) \setminus \mathrm{fn}(R)$ of bound (or private) names in R are defined by extending

the definition of free names on CCS terms to memories and RCCS terms:

$$\begin{aligned} \mathrm{fn}(P\backslash a) &= \mathrm{fn}(P) \setminus \{a\} \\ \mathrm{fn}(a.P) &= \mathrm{fn}(\bar{a}.P) = \{a\} \cup \mathrm{fn}(P) \\ \mathrm{fn}(P \mid Q) &= \mathrm{fn}(P + Q) = \mathrm{fn}(P) \cup \mathrm{fn}(Q) \\ \mathrm{fn}(0) &= \emptyset \end{aligned} \qquad \text{(CCS rules)}$$

$$\begin{aligned} \mathrm{fn}(R\backslash a) &= \mathrm{fn}(R) \setminus \{a\} \\ \mathrm{fn}(R \mid S) &= \mathrm{fn}(R) \cup \mathrm{fn}(S) \\ \mathrm{fn}(m \rhd P) &= \mathrm{fn}(P) \end{aligned} \qquad \text{(RCCS rules)}$$

Remark 1 (On recording the past)**.** To store the information needed to backtrack, RCCS attaches local memories to each thread. CCSK [7], a variant of CCS, simulates reductions by movings a pointer in the term, that is left unchanged. Reversible higher-order π [18] uses a centralised, global memory to store the process before a reduction. Keys are associated to each reduction, thus reverting a transition with key k consists in restoring the process associated to k from the global memory. The exact mechanism used for recording does not have an impact on the theory except for the structural rules, as we note in Remark 2.

The labelled transition system for RCCS is given by the rules of Figure 1.

The prefix constructor $a.P$ stands for sequential composition, the process interacts on a before continuing with P. Rules In+ (for the input) and Out+ (for the output) consumes a prefix by adding in the memory the corresponding event. The backward moves, described by the rules In− and Out−, remove an event at the top of a memory and restores the prefix and the non-deterministic sum. Those rules are presented with a (guarded) sum, but we consider for instance $\varnothing \rhd a.P \xrightarrow{1:a} \langle 1, a, 0\rangle.\varnothing \rhd P$ to be a legal transition, taking $P + 0$ (which is not syntactically correct) to be P.

Parallel composition $P \mid Q$ employ the four rules of Figure 1b to derive a transition. Rules Com+ and Com− depicts two process agreeing to synchronize or to undo a synchronization by providing two dual prefixes,[3] agreeing on the event identifier and triggering the transitions simultaneously. Rules ParL and ParR allow respectively the left or the right process to compute independently of the rest of the process. In those two later rules, the side condition $i \notin \mathsf{I}(S)$ ensures, in the forward direction, the uniqueness of the event identifiers and it prevents, in the backward direction, a part of a previous synchronisation to backtrack alone.

Once the name a is "hidden in P", that is, made private to the process P, it cannot be used to interact with the environment. This situation is denoted with $P\backslash a$ and illustrated in rule Hide. Whenever the private name a is encountered

[3]Notice that since the complement of τ is not defined, only inputs and outputs synchronize.

$$\textsc{In}+\ \frac{}{m \rhd a.P + Q \xrightarrow{i:a} \langle i, a, Q\rangle.m \rhd P}\ i \notin \mathsf{I}(m)$$

$$\textsc{Out}+\ \frac{}{m \rhd \bar{a}.P + Q \xrightarrow{i:\bar{a}} \langle i, \bar{a}, Q\rangle.m \rhd P}\ i \notin \mathsf{I}(m)$$

$$\textsc{In}-\ \frac{}{\langle i, a, Q\rangle.m \rhd P \overset{i:a}{\rightsquigarrow} m \rhd a.P + Q}\ i \notin \mathsf{I}(m)$$

$$\textsc{Out}-\ \frac{}{\langle i, \bar{a}, Q\rangle.m \rhd P \overset{i:\bar{a}}{\rightsquigarrow} m \rhd \bar{a}.P + Q}\ i \notin \mathsf{I}(m)$$

(a) Prefix and sum rules

$$\textsc{Com}+\ \frac{R \xrightarrow{i:\alpha} R' \quad S \xrightarrow{i:\bar{\alpha}} S'}{R \mid S \xrightarrow{i:\tau} R' \mid S'} \qquad \textsc{ParL}\ \frac{R \overset{i:\alpha}{\twoheadrightarrow} R'}{R \mid S \overset{i:\alpha}{\twoheadrightarrow} R' \mid S}\ i \notin \mathsf{I}(S)$$

$$\textsc{Com}-\ \frac{R \overset{i:\alpha}{\rightsquigarrow} R' \quad S \overset{i:\bar{\alpha}}{\rightsquigarrow} S'}{R \mid S \overset{i:\tau}{\rightsquigarrow} R' \mid S'} \qquad \textsc{ParR}\ \frac{R \overset{i:\alpha}{\twoheadrightarrow} R'}{S \mid R \overset{i:\alpha}{\twoheadrightarrow} S \mid R'}\ i \notin \mathsf{I}(S)$$

(b) Parallel constructions

$$\textsc{Hide}\ \frac{R \overset{i:\alpha}{\twoheadrightarrow} R'}{R\backslash a \overset{i:\alpha}{\twoheadrightarrow} R'\backslash a}\ a \notin \{\alpha, \bar{\alpha}\} \qquad \textsc{Congr}\ \frac{R \equiv R' \overset{i:\alpha}{\twoheadrightarrow} S' \equiv S}{R \overset{i:\alpha}{\twoheadrightarrow} S}$$

(c) Hiding and congruence

Figure 1: Rules of the RCCS LTS

in the environment, α-renaming of a is done inside P:

$$P\backslash a =_\alpha (P[b/a])\backslash b$$

where $P[b/a]$ stands for process P in which b substitutes a. We say that the hiding operator is a binder for the private name a.

The structural congruence, whose definition follows, is applied on terms by the rule Congr. It is built on top of some of the corresponding rules for CCS, and rewrites the terms under the memory or distributes it between two forking processes.

Definition 5 (Structural congruence)**.** Structural congruence on monitored processes is the smallest equivalence relation up to uniform renaming of identifiers generated by the:

$$m \rhd (P + Q) \equiv m \rhd (Q + P) \tag{1}$$

$$m \rhd ((P + Q) + R) \equiv m \rhd (P + (Q + R)) \tag{2}$$

$$\frac{P =_\alpha Q}{m \rhd P \equiv m \rhd Q} \qquad (\alpha\text{-conversion})$$

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

Adding a *scope of restriction* rule $m \rhd P\backslash a \equiv (m \rhd P)\backslash a$ could be done at the price of a cumbersome definition of what a free name in a memory is.

Remark 2 (On reduction semantics)**.** Correctness criteria for reversible semantics mostly relate it with its *only-forward* counterpart. However one may be interested in defining a reduction semantics for the LTS of Figure 1 if only to relate RCCS with other reversible semantics for CCS. One, then needs a congruence relation on RCCS terms that has the monoid structure for parallel composition and the null process 0. However, due to the fork constructor, the associativity does not hold:

$$(R_1|R_2)|R_3 \not\equiv R_1|(R_2|R_3).$$

Other reversible calculi, in particular the reversible higher-order π-calculus [18] fares better: its congruence relation respects associativity, thanks to a mechanism that uses bounded names for forking processes. Then α-renaming can be applied on these forking names.

Alternatively, one could use "at distance rewriting" [23] to bypass the lack of flexibility of our structural congruence.

In RCCS not all syntactically correct processes have an operational meaning. Consider for instance

$$\curlyvee.\langle i, \alpha, 0\rangle.\varnothing \rhd P \mid \varnothing \rhd Q.$$

To make a backward transition, one should first apply the congruence rule called "distribution memory" and then look for a rule of the LTS to apply. But this is impossible, since the memory on the right-hand side of the parallel operator

does not contain a fork symbol ($\curlyvee$) at its top. The distributed memory does not agree on a common past, blocking the execution, but this term is correct from a syntactical point of view. In the following, we will consider only the semantically correct terms, called *coherent*.

Definition 6 (Coherent process)**.** A RCCS process R is *coherent* if there exists a CCS process P such that $\varnothing \rhd P \longrightarrow^\star R$.

Coherent terms are also called *reachable*, as they are obtained by a forward execution from a process with an empty memory. Coherence of terms can equivalently be defined in terms of coherence on memories [5, Definition 1].

Backtracking is non-deterministic because backtracking is possible on different threads. However, it is noetherian and confluent as backward synchronisations are deterministic [21, Lemma 11].

Lemma 1 (Unique origin)**.** *If R is a coherent process, then $\forall R'$ such that $R \equiv R'$ or $R \twoheadrightarrow R'$, then R' is also coherent. Up to structural congruence, there exists a unique process P such that $R \rightsquigarrow^\star \emptyset \rhd P$, we call it* the origin of R *and denote it O_R.*

Lastly, we recall a useful result, that asserts that every reversible trace can be rearranged as a sequence of only-backward moves followed by a sequence of only-forward moves.

Lemma 2 (Parabolic traces, [5, Lemma 10])**.** *If $R \twoheadrightarrow \cdots \twoheadrightarrow S$, then there exists R' such that $R \rightsquigarrow^\star R' \longrightarrow^\star S$.*

It is natural to wonder if our reversible syntax is a conservative extension of CCS. We will make sure in the following that the forward rules in the reversible LTS correspond to the LTS of the natural semantics.

Definition 7 (Map from RCCS to CCS)**.** We define inductively a map $\varepsilon(\cdot)$ from RCCS terms to CCS terms by erasing the memory:

$$\varepsilon(m \rhd P) = P \qquad \varepsilon(R|S) = \varepsilon(R)|\varepsilon(S) \qquad \varepsilon(R\backslash a) = (\varepsilon(R))\backslash a$$

In the following lemma, we denote $\xrightarrow{\alpha}$ the standard rewriting rule on CCS terms.

Lemma 3 (Strong "forward" bisimulation between R and $\varepsilon(R)$)**.** *For all R and S, $R \xrightarrow{i:\alpha} S$ for some i iff $\varepsilon(R) \xrightarrow{\alpha} \varepsilon(S)$.*

1.3. Contextual equivalences

Contextual equivalence for CCS terms [24] is now standard, but its extension to RCCS is not straightforward, since contexts needs to be properly defined (Definition 11). As usual, reductions are part of the observables, but observing only them results in a too coarse relation, and adding termination is not relevant in concurrency. *Barbed congruence* (Definition 10) has proven to be the right notion for CCS, and we revisit it for RCCS terms. We begin by recalling definitions of context and observables for CCS.

Definition 8 (Context)**.** A context is a process with a hole $[\cdot]$ defined formally by the grammar:

$$C[\cdot] \coloneqq [\cdot] \parallel \lambda.C[\cdot] \parallel P \mid C[\cdot] \parallel C[\cdot]\backslash a$$

Definition 9 (Barbs)**.** Write $P \downarrow_\alpha$ if there exists P' such that $P \xrightarrow{\alpha} P'$.

We now define a contextual equivalence where reductions and barbs are the observables.

Definition 10 (Barbed congruence)**.** The *barbed bisimulation* is a symmetric relation $\mathcal{R}$ on CCS processes such that whenever $P \mathcal{R} Q$ the following holds:

$$P \longrightarrow P' \implies \exists Q' \text{ s.t. } Q \longrightarrow Q' \text{ and } Q \mathcal{R} Q' \qquad \text{(closed by reduction)}$$
$$P \downarrow a \implies Q \downarrow a \qquad \text{(barb preserving)}$$

If there exists a barbed bisimulation between P and Q we write $P \dot{\sim}^\tau Q$ and say that P and Q are *barbed bisimilar*.

If $\forall C[\cdot]$, $C[P] \dot{\sim}^\tau C[Q]$, we write $P \sim^\tau Q$ and say that P and Q are *barbed congruent*.

An interesting proposition allows to restrict the grammar of contexts in the following.

Proposition 1. $\forall a, P_1, P_2, Q, \lambda, a,$

$$P_1 \dot{\sim}^\tau P_2 \implies \begin{cases} \lambda.P_1 \dot{\sim}^\tau \lambda.P_2 \\ P_1\backslash a \dot{\sim}^\tau P_2\backslash a \\ P_1 + Q \dot{\sim}^\tau P_2 + Q \end{cases}$$

Proof.

1. $P_1 \dot{\sim}^\tau P_2 \implies \lambda.P_1 \dot{\sim}^\tau \lambda.P_2$. From CCS's grammar, $\lambda \neq \tau$, hence $\nexists P_1', P_2'$ such that $P \xrightarrow{\tau} P_1'$ or $P_2 \xrightarrow{\tau} P_2'$. The relation $\{\lambda.P_1, \lambda.P_2\}$ is trivially a barbed bisimulation.
2. $P_1 \dot{\sim}^\tau P_2 \implies P_1\backslash a \dot{\sim}^\tau P_2\backslash a$. Let us denote $\mathcal{R}_1$ the largest barbed bisimulation for P_1 and P_2. We show that the relation $\mathcal{R}_2 = \{P_1\backslash a, P_2\backslash a \mid P_1 \mathcal{R}_1 P_2\}$ is a barbed bisimulation. We have to show that:
   - $\forall b$ such that $P_1\backslash a \downarrow \beta$ then $P_2\backslash a \downarrow \beta$. It follows from $P_1\backslash a \downarrow \beta \implies P_1 \downarrow \beta$ and $\beta \neq a$.
   - $P_1\backslash a \xrightarrow{\tau} P_1'$ implies that $P_2\backslash a \xrightarrow{\tau} P_2'$ and $P_1' \mathcal{R}_2 P_2'$. By structural induction on the transition $P_1\backslash a \xrightarrow{\tau} P_1'$ we have that $\exists P_1''$ such that $P_1 \xrightarrow{\tau} P_1''$ and $P_1''\backslash a = P_1'$. As $P_1 \mathcal{R}_1 P_2$ there exists P_2'' such that $P_2 \xrightarrow{\tau} P_2''$ and we apply the rule HIDE we get $P_2\backslash a \xrightarrow{\tau} P_2''\backslash a$. Thus there exists $P_2' = P_2''\backslash a$ and $P_1' \mathcal{R}_2 P_2'$.

   It follows similarly for the barbs and reductions on P_2.

3. $P_1 \dot{\sim}^{\tau} P_2 \implies P_1 + Q \dot{\sim}^{\tau} P_2 + Q$. Let us denote $\mathcal{R}_1$ the largest barbed bisimulation for P_1 and P_2. We show that the relation $\mathcal{R}_2 = \mathcal{R}_1 \cup \{P_1, P_2\}$ is a barbed bisimulation. As above, we show that:
   - $\forall \alpha$ such that $(P_1 + Q) \downarrow \alpha$ then $(P_2 + Q) \downarrow \alpha$. It follows from the subcases $P_1 \downarrow \alpha$ (hence $P_2 \downarrow \alpha$) or $Q \downarrow \alpha$.
   - $P_1 + Q \xrightarrow{\tau} P_1'$. From rules SumL and SumR either $Q \xrightarrow{\tau} P_1'$ or $P_1 \xrightarrow{\tau} P_1'$. In the first case we deduce, using rule SumL that $P_1 + Q \xrightarrow{\tau} P_1'$, and therefore $P_1' \mathcal{R}_2 P_1'$. In the second case, we apply rule SumR and have that $P_2 \xrightarrow{\tau} P_2'$ and $P_1' \mathcal{R}_1 P_2'$, which concludes our proof.

   It follows similarly for the barbs and reductions on P_2. □

Corollary 1. *If a context $C[\cdot]$ does not contain the parallel operator, then for all P, Q, $C[P] \not\dot{\sim}^{\tau} C[Q]$ implies $P \not\dot{\sim}^{\tau} Q$*

Stated differently, this implies that discriminating contexts regarding barbed congruence involve parallel composition. As we will focus on this relation, we will only consider in the following the contexts to be parallel compositions:

$$C[\cdot] \coloneqq [\cdot] \parallel P \mid [\cdot]$$

This is handy to define RCCS context, but some subtleties remain. A context has to become an executable process regardless of the process instantiated with it. We say that a context has a coherent memory, it may backtrack up to the context with an empty memory (similar to the Definition 6 of coherent processes). We distinguish three types of contexts, depending on their memories:

- Contexts with an empty memory.
- Contexts with a non-empty memory that is coherent on its own.[4] The process that we instantiate with it can be
  - incoherent,[4] in which case we conjecture that the term obtained after instantiation is also incoherent,
  - coherent on their own,[4] in which case it is possible to backtrack the memory of the context up to the empty memory.

Hence w.l.o.g. we consider contexts without memory and contexts with coherent memories to be equivalent. These are the types of contexts that we use throughtout the article. However, a third case remains:

- Contexts that have a non-coherent memory. There exists incoherent terms whose instantiation with an incoherent context is coherent. For instance, $C = \langle 1, a, 0 \rangle.\ \Upsilon\ .\varnothing \rhd P \mid [\cdot]$ and $R = \langle 1, \bar{a}, 0 \rangle.\ \Upsilon\ .\varnothing \rhd P'$ are incoherent individually, but $C[R]$ is coherent and can backtrack to $\Upsilon.\varnothing \rhd P \mid \Upsilon.\varnothing \rhd P'$. We leave this case as future work.

[4] Up to minor addition of Υ symbols, as explained later on.

The "up to minor addition of Υ symbols" comes from a simple consideration on the parallel composition in RCCS. A process with an empty memory compose with a RCCS term if fork symbols are added to reflect the parallel composition. For instance, two processes with an empty memory $\varnothing \rhd P$ and $\varnothing \rhd P'$ compose and we obtain

$$\Upsilon.\varnothing \rhd P \mid \Upsilon.\varnothing \rhd P' \equiv \varnothing \rhd (P \mid P')$$

instead of $\varnothing \rhd P \mid \varnothing \rhd P'$, an incoherent process.

We define a rewriting function on RCCS processes, that adds a fork symbol at the beginning of a memory. It allows a process with a memory to compose with a context.

Definition 11 (RCCS context)**.** Define $\Upsilon(R)$ the operator that adds a fork symbol at the beginning of the memory of each thread in R:

$$\begin{aligned} \Upsilon(R_1 \mid R_2) &= \Upsilon(R_1) \mid \Upsilon(R_2) \\ \Upsilon(R\backslash a) &= (\Upsilon(R))\backslash a \\ \Upsilon(m \rhd P) &= m'.\ \Upsilon\ .\varnothing \rhd P \text{ where } m = m'.\varnothing \\ \Upsilon(0) &= 0 \end{aligned}$$

Define $C_\Upsilon[R]$ as follows

$$C_\Upsilon[R] = \begin{cases} R & \text{if } C[\cdot] = [\cdot] \\ \Upsilon.\varnothing \rhd P \mid \Upsilon(R) & \text{if } C[\cdot] = P \mid [\cdot] \end{cases}$$

RCCS context are basically CCS context with additional fork symbols in the memory of the context and in the memory of the process instantiated. We now verify that $C_\Upsilon[R]$ is a coherent process, using the function $\varepsilon(\cdot)$ that erases the memories from a term (Definition 7).

Proposition 2. *For all R and $C_\Upsilon[\cdot]$, $\varnothing \rhd C[\varepsilon(O_R)] \longrightarrow^\star C_\Upsilon[R]$.*

Proof. Let $C[\cdot] = P \mid [\cdot]$ and $O_R = \varnothing \rhd P'$. By definition and application of the congruence rules, we have that

$$\begin{aligned} \varnothing \rhd C[\varepsilon(O_R)] &= \varnothing \rhd \big(P \mid \varepsilon(O_R)\big) \\ &= \varnothing \rhd \big(P \mid P'\big) \\ &\equiv (\Upsilon.\varnothing \rhd P) \mid (\Upsilon.\varnothing \rhd P') \end{aligned}$$

We have from the trace $O_R \longrightarrow^\star R$ that

$$(\Upsilon.\varnothing \rhd P) \mid (\Upsilon.\varnothing \rhd P') \longrightarrow^\star (\Upsilon.\varnothing \rhd P) \mid \Upsilon(R) = C_\Upsilon[R]. \qquad \square$$

Example 1. Let $R = \Upsilon.m.\varnothing \rhd P_1 \mid \Upsilon.m.\varnothing \rhd P_2$ and $C[\cdot] = P \mid [\cdot]$. Let us rewind R to its origin:

$$\begin{aligned} R &= \Upsilon.m.\varnothing \rhd P_1 \mid \Upsilon.m.\varnothing \rhd P_2 \\ &\equiv m.\varnothing \rhd (P_1 \mid P_2) \\ &\rightsquigarrow^\star \varnothing \rhd P' \\ &= O_R \end{aligned}$$

We instantiate the context with O_R and redo the execution from the origin of R up to R:

$$\begin{aligned} C_\Upsilon[O_R] &= (\Upsilon.\varnothing \rhd P) \mid (\Upsilon.\varnothing \rhd P') \\ &\longrightarrow^\star (\Upsilon.\varnothing \rhd P) \mid \big((m.\ \Upsilon\ .\varnothing \rhd P_1) \mid (m.\ \Upsilon\ .\varnothing \rhd P_2)\big) \\ &= (\Upsilon.\varnothing \rhd P) \mid \Upsilon(R) \\ &= C_\Upsilon[R] \end{aligned}$$

Hence we have that $C_\Upsilon[O_R] \longrightarrow^\star C_\Upsilon[R]$.

Once this delicate notion of context for reversible process is settled, extending the CCS barbs (Definition 9) and barbed congruence (Definition 10) are straightforward.

Definition 12 (RCCS barbs)**.** We write $R \downarrow_\alpha$ if there exists $i \in I$ and R' such that $R \xrightarrow{i:\alpha} R'$.

Definition 13 (Back-and-forth barbed congruence)**.** A *back-and-forth bisimulation* is a symmetric relation on coherent processes $\mathcal{R}$ such that if $R\ \mathcal{R}\ S$, then

$$R \overset{i:\tau}{\rightsquigarrow} R' \implies \exists S' \text{ s.t. } S \overset{i:\tau}{\rightsquigarrow} S' \text{ and } R'\ \mathcal{R}\ S'; \qquad \text{(back)}$$

$$R \xrightarrow{i:\tau} R' \implies \exists S' \text{ s.t. } S \xrightarrow{i:\tau} S' \text{ and } R'\ \mathcal{R}\ S'; \qquad \text{(forth)}$$

and it is a *back-and-forth* barbed *bisimulation* if, additionally,

$$R \downarrow_a \implies S \downarrow_a . \qquad \text{(barbed)}$$

We write $R \dot{\sim}^\tau S$ if there exists $\mathcal{R}$ a back-and-forth barbed bisimulation such that $R\ \mathcal{R}\ S$.

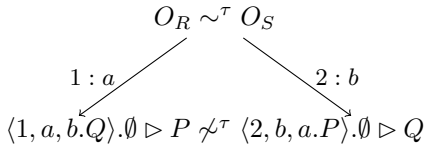
The *back-and-forth barbed congruence*, denoted $R \sim^\tau S$, holds if for all context $C_\Upsilon[\cdot]$, $C_\Upsilon[O_R] \dot{\sim}^\tau C_\Upsilon[O_S]$.

From the definition of $R \sim^\tau S$, the following lemma trivially holds.

Lemma 4. *For all R and S, $R \sim^\tau S \implies O_R \sim^\tau O_S$.*

However, the converse does not hold as R and S can be any derivative of the same origin, as illustrated below.

Example 2. Let $R = \langle 1, a, b.Q\rangle.\emptyset \rhd P$ and $S = \langle 2, b, a.P\rangle.\emptyset \rhd Q$, with $P \not\dot{\sim}^\tau Q$. We have that $O_R \sim^\tau O_S$, as $O_R = O_S$, but $R \not\sim^\tau S$, as $P \not\dot{\sim}^\tau Q$:



Note that even if the context is defined for any reversible process, we instantiate the context with processes with an empty memory in Definition 13. If instead we had defined $R \sim^{\tau} S$ iff for all contexts, there exists $\mathcal{R}$ such that $C_{\Upsilon}[R]\ \mathcal{R}\ C_{\Upsilon}[S]$, then Lemma 4 would not hold. We highlight this in the following example.

Example 3. Let us consider the processes $\varnothing \rhd a.P + Q$ and $\varnothing \rhd a.P$ which can do a transition on a to obtain $R = \langle 1, a, Q\rangle \rhd P$ and $S = \langle 1, a\rangle \rhd P$. We have that R and S are back-and-forth barbed bisimular. As we are using contexts without memory, there is no context able to backtrack on a.

$$\begin{array}{ccc} O_R = \varnothing \rhd a.P + Q & \dot{\not\sim}^{\tau} & O_S = \varnothing \rhd a.P \\ \big\downarrow 1 : a & & \big\downarrow 1 : a \\ R = \langle 1, a, Q\rangle \rhd P & \sim^{\tau} & S = \langle 1, a\rangle \rhd P \end{array}$$

Remark 3 (On backward barbs)**.** Let us informally argue that backward barbs are not an interesting addition to a contextual equivalence. One can always define ad-hoc barbs that potentially change the equivalence relations, however we end up with relations that have no practical meaning. We consider below another definition of barb [25, Definition 2.1.3], which gives an intuitive reading and is not syntax-specific.

Let the tick ($\checkmark \notin \mathsf{N}$) be a special symbol denoting termination. A *barb* is an interaction with a context that can do a tick immediately after:

$$P \downarrow_{\alpha} \iff P \mid \bar{\alpha}.\checkmark \xrightarrow{\tau} Q \mid \checkmark \text{ for some } Q.$$

Note that the definition above implies that (i) the barb is an interaction with a context that can terminate immediately after and (ii) the interaction *blocks* the termination on the context side, i.e. no further transition is possible on that side.

If we try to apply this definition to a backward barb then the tick has to be in the memory of the context and blocked by another action, i.e. the context has to be of the form $C[\cdot] \equiv [\cdot] \mid (\langle i, \bar{\alpha}, 0\rangle.\langle\checkmark\rangle.\varnothing \rhd 0)$. This raises multiple problems:

1. Syntactically, $\checkmark$ becomes a prefix, rather than a "terminal process", i.e. terms of the form $\checkmark.a.P$ appear. This contradict the intuition that this symbol stands for termination.
2. In a situation where $C[R] \overset{i:\tau}{\rightsquigarrow} R' \mid \langle\checkmark\rangle.\varnothing \rhd 0$, the $\checkmark$ symbol is not observable, and R' could continue its computation before $\checkmark$ is popped from the context's memory. So we would have to add the content of the memory to what is observable. But in that case, one might as well look directly in the process' memory if a label is present.
3. Lastly, defining the backward barb as the capability to do a backward step, and having immediately after the forward barb, seems to be equivalent to any reasonable definition of backward barb.

Thus we argue that the backward barbs are a contrived notion.

2. Configuration structures as a model of reversibility

Causal models take causality and concurrency between events as primitives. In configuration structures, configurations stands for computational states and the set inclusion represents the executions, so that in each state one can infer a local order on the events based on the set inclusion. We introduce them and their categorical representation modeling operations from process (Sect. 2.1).

One can also obtain a causal semantics of a process calculus, by decorating its LTS. In Sect. 2.2 we briefly show how to interpret CCS terms in configuration structures and how to decorate its LTS to derive causal information from the transitions.

Lastly, we introduce configuration structure for a restricted class of RCCS process, called *singly labelled* (Definition 24). They are essentially an address in the configuration structure of the underlying, "original" CCS term. We then introduce the LTS of those configuration structures and prove their operational correspondence with the reversible syntax (Lemma 5).

2.1. Configuration structures as a causal model

Let E be a set, $\subseteq$ be the usual set inclusion and C be a family of subsets of E. For $X \subseteq C$, X is *compatible*, denoted $X\uparrow$, if $\exists y \in C$ finite such that $\forall x \in X$, $x \subseteq y$.

Definition 14 (Configuration structures)**.** A *configuration structure* $\mathcal{C}$ is a triple $(E, C, \subseteq)$ where E is a set of events, $\subseteq$ is the set inclusion and $C \subseteq \mathcal{P}(E)$ is a set of subsets satisfying:

- *finiteness*: $\forall x \in C, \forall e \in x, \exists z \in C$ finite such that $e \in z$ and $z \subseteq x$,
- *coincidence freeness*: $\forall x \in C, \forall e, e' \in x, e \neq e' \Rightarrow \big(\exists z \in C, z \subseteq x$ and $(e \in z \iff e' \notin z)\big)$,
- *finite completness*: $\forall X \subseteq C$ if $X\uparrow$ then $\bigcup X \in C$,
- *stability*: $\forall x, y \in C$ if $x \cup y \in C$ then $x \cap y \in C$.

We denote $\mathbf{0}$ the configuration structure with $E = \emptyset$.

Intuitively, events are the actions occurring during the run of a process, while configurations represents computational states. The first axiom, *finiteness*, guarantees that for each event the set of causes is finite. *Coincidence freeness* states that each computation step consists of a single event. Axioms *finite completness* and *stability* are more abstract and are better explained on some examples. Consider the structures in Figure 2: the structure 2a does not satisfy the second axiom, as two events occur in a single step. The structure 2b does not satisfy finite completeness. Intuitively, configuration structures cannot capture "pairwise" concurrence. Finally, the structure 2c does not satisfies stability and the intuition is that the causes of event e_3 are *either* e_1 *or* e_2, but not both.

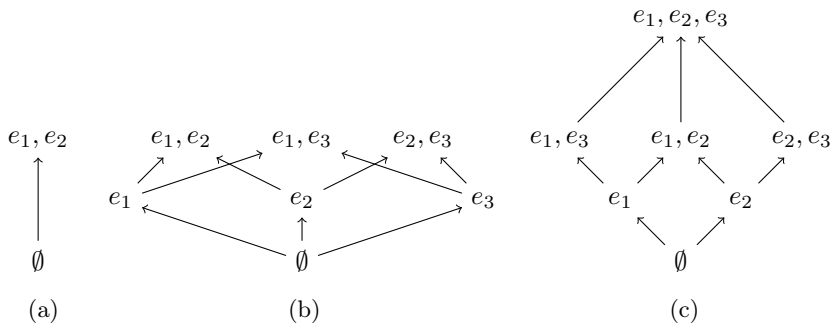


Figure 2: Structures that are not coincidence free, finite complete and stable, respectively

Notations 2. In a configuration $\mathcal{C}C$, if $x, x' \in C$, $e \in E$, $e \notin x$ and $x' = x \cup \{e\}$, then we write $x \xrightarrow{e} x'$.

Definition 15 (Labelled configuration structure)**.** A *labelled configuration structure* $\mathcal{C} = (E, C, \ell)$ is a configuration structure endowed with a *labelling function* from events to labels $\ell : E \to \mathsf{L}$.

From now on, we will only consider configurations structures that are labelled, so we omit that adjective in the following.

Now we define morphisms on configurations structures that permits to form a category whose objects are configuration structures. Intuitively, morphisms model the inclusion or refinement relations between processes. Process algebras' operators are then extended to configuration structures, which makes it a modular model.

Definition 16 (Category of configuration structures)**.** A morphism of configurations structures $f : (E_1, C_1, \ell_1) \to (E_2, C_2, \ell_2)$ is a partial function on the underlying sets $f : E_1 \rightharpoonup E_2$ that is:

- *configurations preserving*: $\forall x \in C_1, f(x) = \{f(e) \mid e \in x\} \in C_2$,
- *local injective*: $\forall x \in C_1, \forall e_1, e_2 \in x, f(e_1) = f(e_2) \implies e_1 = e_2$,
- *label preserving*: $\forall x \in C_1, \forall e \in x, \ell_1(e) = \ell_2(f(e))$.

An isomorphism on configuration structures is denoted $\cong$.

Definition 17 (Operations on configuration structures)**.** Let $\mathcal{C}_1 = (E_1, C_1, \ell_1)$, $\mathcal{C}_2 = (E_2, C_2, \ell_2)$ be two configuration structures and set $E^\star = E \cup \{\star\}$.

Product Let $\star$ denote *undefined* for a partial function. Define *the product of* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ as $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2$, for $\mathcal{C} = (E, C, \ell)$, where $E = E_1 \times_\star E_2$ is the product

in the category of sets and partial functions[5]:

$$E_1 \times_\star E_2 = \{(e_1, \star) \mid e_1 \in E_1\} \cup \{(\star, e_2) \mid e_2 \in E_2\} \\ \cup \{(e_1, e_2) \mid e_1 \in E_1, e_2 \in E_2\}$$

with the projections $p_1 : E \to E_1 \cup \{\star\}$, $p_2 : E \to E_2 \cup \{\star\}$. Define the projections $\pi_1 : (E, C) \to (E_1, C_1)$, $\pi_2 : (E, C) \to (E_2, C_2)$ such that the following holds, for $e \in E$ and $x \in C$:

- $\pi_1(e) = p_1(e)$ and $\pi_2(e) = p_2(e)$;
- $\pi_1(x) \in C_1$ and $\pi_2(x) \in C_2$;
- $\forall e, e' \in x$, if $\pi_1(e) = \pi_1(e') \neq \star$ or $\pi_2(e) = \pi_2(e') \neq \star$ then $e = e'$;
- $\forall e \in x, \exists z \subseteq x$ finite s.t. $\pi_1(x) \in C_1$, $\pi_2(x) \in C_2$ and $e \in z$;
- $\forall e, e' \in x, e \neq e' \Rightarrow \exists z \subseteq x$ s.t. $\pi_1(z) \in C_1$, $\pi_2(z) \in C_2$ and $(e \in z \iff e' \notin z)$.

The labelling function ℓ is defined as follows:

$$\ell(e) = \begin{cases} \ell_1(e_1) & \text{if } \pi_1(e) = e_1, \pi_2(e) = \star \\ \ell_2(e_2) & \text{if } \pi_1(e) = \star, \pi_2(e) = e_2 \\ (\ell_1(e_1), \ell_2(e_2)) & \text{otherwise} \end{cases}$$

Coproduct Define *the coproduct of $\mathcal{C}_1$ and $\mathcal{C}_2$* as $\mathcal{C} = \mathcal{C}_1 + \mathcal{C}_2$, for $\mathcal{C} = (E, C, \ell)$, 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 labelling function ℓ is defined as $\ell(e) = \ell_i(e_i)$ when $e_i \in E_i$ and $\pi_i(e_i) = e$.

Restriction Let $E' \subseteq E$ and define *the restriction of a set of events* as $(E, C, \ell) \restriction E' = (E', C', \ell')$ where $x \in C' \iff x \in C$ and $x \subseteq E'$.

The restriction of a name is then $(E, C, \ell) \restriction a \coloneqq (E, C, \ell) \restriction E_a$ where $E_a = \{e \in E \mid \ell(e) \in \{a, \bar{a}\}\}$. For $a_1, \ldots, a_n$ a list of names, we define similarly $\restriction \cup_{1 \leqslant i \leqslant n} E_{a_i}$.

Prefix Let λ be the label of an event and define *the prefix operation on configuration structures* as $\alpha.(E, C, \ell) = (e \cup E, C', \ell')$, for $e \notin E$ where $x' \in C' \iff \exists x \in C$, $x' = x \cup e$ and $\ell'(e) = \alpha$, and $\forall e' \neq e$, $\ell'(e') = \ell(e')$.

Relabelling Define *the relabelling of a configuration structure* as $\mathcal{C}_1 \circ \ell = (E_1, C_1, \ell)$, where $\mathcal{C}_1 = (E_1, C_1, \ell_1)$ and $\ell : E_1 \to \mathsf{L}$ is a labelling function.

Parallel composition Define parallel composition $\mathcal{C} = \big((\mathcal{C}_1 \times \mathcal{C}_2) \circ \ell\big) \restriction E$ as the application of product, relabelling and restriction, with ℓ and E defined below.

[5]The category of sets and partial functions has sets as objects and functions that can take the value $\star$ as morphisms [26, Appendix A].

- First, $\mathcal{C}_1 \times \mathcal{C}_2 = \mathcal{C}_3$ is the product with $\mathcal{C}_3 = (E_3, C_3, \ell_3)$;
- Then, $\mathcal{C}' = \mathcal{C}_3 \circ \ell$ with ℓ defined as follows:

$$\ell(e) = \begin{cases} \ell_3(e) & \text{if } \ell_3(e) \in \{a; \bar{a}\} \\ \tau & \text{if } \ell_3(e) \in \{(a, \bar{a}); (\bar{a}, a)\} \\ 0 & \text{otherwise} \end{cases}$$

- Finally, $\mathcal{C} = (E_1 \times_\star E_2, C_3, \ell) \upharpoonright E$ is the resulted configuration structure, where $E = \{e \in E_3 \mid \ell(e) \neq 0\}$.

In the definition of the product, the conditions guarantee that $\mathcal{C}_1 \times \mathcal{C}_2$ is the product in the category of configuration structures and that the projections π_1, π_2 are morphisms. In particular, the third condition ensures that the projections are local injective, the fourth and fifth enforce finiteness and coincidence-freeness axioms in the resulted configuration structure.

Definition 18 (Causality)**.** Let $x \in C$ and $e_1, e_2 \in x$ for (E, C, ℓ) a configuration structure. Then we say that e_1 *happens before* e_2 *in* x or that e_1 *causes* e_2 *in* x, written $e_1 \leqslant_x e_2$, iff $\forall x_2 \in C, x_2 \subseteq x, e_2 \in x_2 \implies e_1 \in x_2$.

Configurations can also be interpreted as *temporal* observations [2, Chap. 5], instead of causal orders present in the structure of a term. Refering to the order as *happens before* instead of causality highlights the observational nature of the order.

Morphisms on configuration structures reflect causality. Let $f : \mathcal{C}_1 \to \mathcal{C}_2$ a morphism and $x \in C_1$ a configuration. Then

$$\forall e_1, e_2 \in x, \text{ if } f(e_1) \leqslant_{f(x)} f(e_2) \text{ then } e_1 \leqslant_x e_2.$$

However, morphisms do not preserve causality in general. In the case of a product we can show that all *immediate* causalities are due to one of the two configurations structures. Stated differenlty, a context can add but cannot remove causality in the process.

Definition 19 (Immediate causality)**.** Let e, e' be two events in a configuration x for a configuration structure (E, C, ℓ). Denote $e \to_x e'$ if e is an *immediate cause* for e' in x, that is $e <_x e'$ and $\nexists e''$ such that $e <_x e'' <_x e'$.

Note that we overload the notation $e \to_x e'$ however as it is defined on events only, it is not ambiguous.

Proposition 3. *Let $x \in \mathcal{C}_1 \times \mathcal{C}_2$. Then $e_1 \to_x e_2 \iff$ either $\pi_1(e_1) <_{\pi_1(x)} \pi_1(e_2)$ or $\pi_2(e_1) <_{\pi_2(x)} \pi_2(e_2)$.*

Proof. The proof [2, Proposition 6] follows by contradiction, using that if x is a configuration in $\mathcal{C}$ and if $e \in x$ is such that $\forall e' \in x$, $e \not<_x e'$, then $x \setminus e$ is a configuration in $\mathcal{C}$. □

2.2. Operational semantics, correspondence and equivalences

Configuration structures are a causal model for CCS [27] in which the computational states are the configurations and the forward executions are dictated by set inclusion. To show the correspondence with CCS (Lemma 5), one defines an operational semantics on configurations structures (Definition 21) that erases the part of the structure that is not needed in future computations. In order to define a reversible semantics on configurations structures a second LTS is introduced (Definition 22), that instead of being defined on configurations structures is defined on the configurations of a stable family. Thus forward and backward moves are simply the set inclusion relation and its opposite, respectively.

The soundness of the model is proved by defining an operational semantics on configurations structures and showing an operational correspondence between the two worlds.

Definition 20 (Encoding a CCS term)**.** Given P a CCS term, its encoding $[\![P]\!]$ as a configuration structure is built inductively, using the operations of Definition 17:

$$[\![a.P]\!] = a.[\![P]\!] \qquad [\![\bar{a}.P]\!] = \bar{a}.[\![P]\!]$$
$$[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!] \qquad [\![P + Q]\!] = [\![P]\!] + [\![Q]\!]$$
$$[\![P\backslash a]\!] = [\![P]\!] \upharpoonright E_a \qquad [\![0]\!] = \mathbf{0}$$

Definition 21 (LTS on configurations structures)**.** Let $\mathcal{C} = (E, C, \ell)$ be a configuration structure. Define $\mathcal{C} \setminus e = (E \setminus e, C', \ell')$ where ℓ' is the restriction of ℓ to the set $E \setminus e$ and

$$x \in C' \iff x \cup \{e\} \in C.$$

We easily make sure that $\mathcal{C} \setminus e$ is also a configurations structures.

We define a LTS on configurations structures thanks to the relation $\mathcal{C} \xrightarrow{e} \mathcal{C}\setminus e$, and we extend the notation to $\mathcal{C} \xrightarrow{\ell(e)} \mathcal{C}\setminus e$ and to $\mathcal{C} \xrightarrow{x} \mathcal{C}\setminus x$, for x a configuration.

Lemma 5 (Operational correspondence between a process P and its encoding $[\![P]\!]$)**.** *Let P a process and $[\![P]\!] = (E, C, \ell)$ its interpretation.*

1. *$\forall \alpha$, P' such that $P \xrightarrow{\alpha} P'$, $\exists e \in E$ such that $\ell(e) = \alpha$ and $[\![P]\!] \setminus e \cong [\![P']\!]$;*
2. *$\forall e \in E$, if $\{e\} \in C$ then $\exists P'$ such that $P \xrightarrow{\ell(e)} P'$ and $[\![P]\!] \setminus e \cong [\![P']\!]$.*

The above lemma shows that *labelled* transitions are in correspondence, but labels are just a tool for compositionality. The main result is that a process and its encoding simulate each others *reductions*.

Theorem 1 (Operational correspondence with CCS)**.** *Let P a process and $[\![P]\!] = (E, C, \ell)$ its encoding.*

1. *$\forall P'$ such that $P \xrightarrow{\tau} P'$, $\exists \{e\} \in C$ closed such that $[\![P]\!] \setminus e \cong [\![P']\!]$;*

2. $\forall e \in E$, $\{e\} \in C$ *closed,* $\exists P'$ *such that* $P \xrightarrow{\tau} P'$ *and* $[\![P]\!] \setminus e \cong [\![P']\!]$.

Multiple equivalence relations on configuration structures have been defined and studied [12, 14, 16, 17, 28]. Among them, hereditary history preserving bisimulation (HHPB) [15] equates structures that simulate each others' forward and backward moves and thus connects configuration structures to reversibility. It is considered a canonical equivalence on configuration structures as it respects the causality and concurrency relations between events and admits a categorical representation [29].

Those connections between reversibility and causal models sheds a new light on what help are the meaningful equivalences on reversible processes. Consequently, one apply them in the operational setting. We begin by modifying the definition of our LTS on configuration structures to include backward moves as well.

Definition 22 (A reversible LTS on configuration structures)**.** Consider (E, C, ℓ) a configuration structure. For $x \in C$, $e \in E$ define $x \xrightarrow{e} x'$ iff $x' = x \cup \{e\}$ and $x \xrightarrow{\alpha} x'$ if additionally, $\ell(e) = \alpha$. The backward moves are defined as $x \overset{e}{\rightsquigarrow} x'$ and $x \overset{\alpha}{\rightsquigarrow} x'$ if $x = x' \cup \{e\}$ and $\ell(e) = \alpha$.

Denote $x \overset{e}{\twoheadrightarrow} x'$ when either $x \xrightarrow{e} x'$ or $x \overset{e}{\rightsquigarrow} x'$.

Such a LTS naturally satisfies elementary criterion that one wold expect from a LTS [2, Chap. 2].

Definition 23 (HHPB [15, Definition 1.4])**.** A *hereditary history preserving bisimulation* on labelled configuration structures is a symmetric relation $\mathcal{R} \subseteq C_1 \times C_2 \times \mathcal{P}(E_1 \times E_2)$ such that $(\emptyset, \emptyset, \emptyset) \in \mathcal{R}$ and if $(x_1, x_2, f) \in \mathcal{R}$, then

$$
\begin{aligned}
& f \text{ is a label and order preserving bijection between } x_1 \text{ and } x_2 \\
x_1 \xrightarrow{e_1} x_1' \implies & \exists x_2' \in C_2 \text{ s.t. } x_2 \xrightarrow{e_2} x_2' \text{ and } f = f' \restriction x_1, (x_1', x_2', f') \in \mathcal{R} \\
x_1 \overset{e_1}{\rightsquigarrow} x_1' \implies & \exists x_2' \in C_2 \text{ s.t. } x_2 \overset{e_2}{\rightsquigarrow} x_2' \text{ and } f' = f \restriction x_2, (x_1', x_2', f') \in \mathcal{R}
\end{aligned}
$$

It is known [7] that hereditary history preserving bisimulation corresponds to the back-and-forth bisimulation (Definition 13), in the following sense: CCSK [30] is proven to satisfy the "the axioms of reversibility" [31, Definition 4.2]), so that its LTS is *prime*. Then, this LTS is represented as a process graph, on which the forward-reverse bisimulation [7, Definition 5.1]—our back-and-forth bisimulation (Definition 13)— is defined. Finally, configuration graphs and hereditary history-preserving bisimulation are defined from configuration structures, and both relation are proven to coincide. [7, Theorem 5.4, p. 105].

2.3. Configuration structures for RCCS

All the possible future behaviours of a process without memory are present in its encoding as a configuration structure. All alike, we want our encoding of processes with memory to record both their past *and* their future, so that they can evolve in both directions, as process do. To this end, we encode RCCS terms

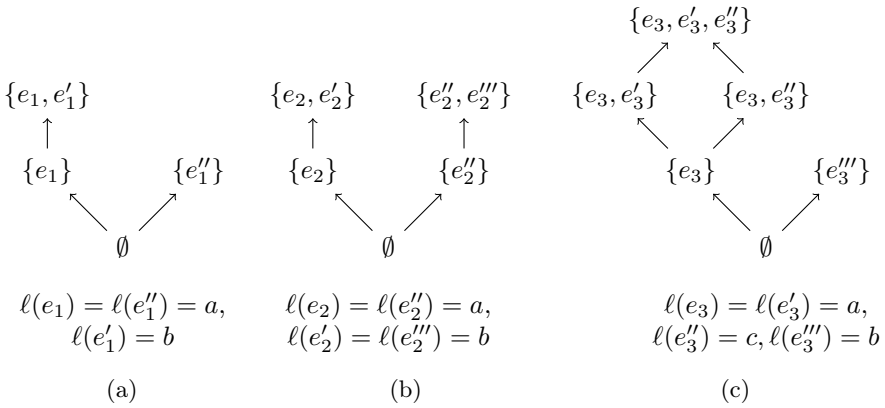


Figure 3: Encoding RCCS in configurations structures

as a configuration in the configuration structure of their origins (Definition 25). Then, we show an operational correspondence between RCCS terms and their encoding.

To determine which configuration corresponds to the computational state of the term we are encoding, we need to uniquely identify a process from its past and future. However, as the following example illustrates, this is not always possible: Remark 4 explains the limitations of the encoding we are going to develop.

Example 4.
1. The process $P = a.b + a$ is interpreted as the configuration structure in Figure 3a. Let us consider the execution $\varnothing \rhd P \xrightarrow{a} R$. To determine which of the configurations labelled a correspond to R we have to consider the future of R as well.
2. Hence we choose a configuration that respects the past and the future of R, but such a configuration might not be unique. Let $Q = a.b + a.b$ be a process whose configuration structure is in Figure 3b. For the trace $P \xrightarrow{a} b$ there is no way to choose between the two configurations labelled a.

The situation of example 4 is generalizable to any process whose reduction may lead to a process of the form $a.P + a.P$ or $a.P \mid a.P$. We consider then a restricted class of processes, as discussed in the following remark.

Remark 4 (On auto-concurrency and others limitations)**.** In the following, we need to uniquely identify configurations based solely on the labels and orders of the *open* (i.e. non-synchronized) events. As seen in example 4, this is not possible in encoding of processes that may reduce to the form $a.P \mid a.Q$ or $a.P + a.Q$.

The first kind of process is characterised by an *auto-concurrency* condition [32, Definition 9.5]. We need a stronger condition, a sort of *auto-conflict*, to forbid the second type of process.

Definition 24 (Singly labelled configuration structures and processes)**.** A configuration structure $\mathcal{C}$ is *singly labelled*, or *without auto-concurrency nor auto-conflict* if $\forall x \in \mathcal{C}$ and $\forall e, e' \notin x$ we have that

$$\big(x \xrightarrow{e} y, x \xrightarrow{e'} y' \text{ and } \ell(e) = \ell(e')\big) \implies e = e'.$$

A process is singly labelled if its encoding as a configuration structure is.

Remark that being singly labelled does not mean that each label has to occurs only once in a process: whereas $a \mid b.a$ is not, since after firing b two transitions labelled a can be fired, $a.a$ and $a.b + b$ are singly labelled. However, a syntactical definition of this restriction cannot be inductively defined, since P and Q might be singly labelled, but not $P \mid Q$ nor $P + Q$.

The following encoding, and all the results that use it, require the process to be singly labelled (on top of being coherent, if they are reversible). This restriction could probably be removed at the price of a *tagging* of the occurrences of names, maybe in the spirit of the *localities* [33].

Definition 25 (Encoding RCCS processes in configurations structures)**.** Let R be a singly labelled process and $\mathcal{C} = [\![\varepsilon(O_R)]\!]$ the encoding (Definition 20) of its "memory-less" origin (Definition 7).

We first need the function $\mathrm{ad}_{\mathcal{C}}$, defined as:

$$\mathrm{ad}_{\mathcal{C}}(x, f, R_1 \xrightarrow{i:\alpha} R_2 \longrightarrow^\star R_3) = \mathrm{ad}_{\mathcal{C}}(x \cup \{e\}, f \cup \{e \leftrightarrow i\}, R_2 \longrightarrow^\star R_3) \quad (3)$$

$$\mathrm{ad}_{\mathcal{C}}(x, f, R_2 \longrightarrow^\star R_3) = x \text{ if } R_2 = R_3 \quad (4)$$

Where in (3) e is such that

- $\ell(e) = \alpha$;
- $x \cup \{e\} \in \mathcal{C}$;
- $j <_{R_2} i \iff f(j) <_{x \cup \{e\}} e$;
- and $[\![\varepsilon(R_2)]\!] = \big(\mathcal{C} \setminus (x \cup \{e\})\big)$.

Now we define the encoding of R in configuration structure by induction on the trace (Definition 2) $\sigma : O_R \longrightarrow^\star R$, as $[\![R]\!]_\sigma = (\mathcal{C}, \mathrm{ad}_{\mathcal{C}}(\emptyset, \emptyset, \sigma))$.

We show in Proposition 4 that the function is well defined, i.e. for every singly labelled process R and for every trace $\sigma : O_R \longrightarrow^\star R$ there exists a unique configuration in $[\![\varepsilon(O_R)]\!]_\sigma$ defined as above.

Example 5. A first simple example is the encoding of a process with an empty memory. Let $S = \varnothing \rhd P$, $\varepsilon(O_S) = P$ and $[\![S]\!]_\epsilon = ([\![P]\!], \emptyset)$.

Let us show how to compute the encoding of the process

$$R = \langle 2, a, 0\rangle.\ \Upsilon\ .\langle 1, a, b\rangle \rhd 0 \mid \Upsilon.\langle 1, a, b\rangle \rhd a.$$

We backtrack to its origin and obtain $O_R = \varnothing \rhd a.(a \mid c) + b$. The term is encoded in the configuration structure in Figure 3c. We apply the function $\mathrm{ad}_{\mathcal{C}}(\emptyset, O_R \longrightarrow^\star R)$ on the trace

$$\begin{aligned}\varnothing \rhd a.(a \mid c) + b &\xrightarrow{1:a} \langle 1, a, b\rangle \rhd (a \mid c)\\ &\equiv (\curlyvee.\langle 1, a, b\rangle \rhd a) \mid (\curlyvee.\langle 1, a, b\rangle \rhd c)\\ &\xrightarrow{2:a} \langle 2, a, 0\rangle.\ \curlyvee\ .\langle 1, a, b\rangle \rhd 0 \mid \curlyvee.\langle 1, a, b\rangle \rhd c\\ &= R'\end{aligned}$$

The configuration corresponding to R is then $\{e_3, e_3'\}$.

Let us show that the encoding is correct, and in particular that the function $\mathrm{ad}_{\mathcal{C}}$ is well defined.

Proposition 4 (Soundness of the RCCS encoding)**.** *Let P be a singly labelled process and $\mathcal{C} = [\![P]\!]$ its encoding. Then for any R reachable from $\varnothing \rhd P$ there exists a unique $x \in \mathcal{C}$ such that* $\mathrm{ad}_{\mathcal{C}}(\emptyset, \emptyset, \varnothing \rhd P \longrightarrow^\star R) = x$.

Proof. From Lemma 2, we consider the trace $O_R \longrightarrow^\star R$ to be only forward. We proceed by induction on the trace $O_R \longrightarrow^\star R$. For the inductive case we have the trace $O_R \longrightarrow^\star R_n$ and $\mathrm{ad}_{\mathcal{C}}(\emptyset, f_n, O_R \longrightarrow^\star R_n) = x_n$, for $x_n \in \mathcal{C}$, f_n a label and order preserving bijection between x_n and R_n, and such that $[\![\varepsilon(R_n)]\!] = \mathcal{C} \setminus x_n$. We have to show that for the trace $O_R \longrightarrow^\star R_n \xrightarrow{i:a} R_{n+1}$ there exists a unique configuration $x_{n+1} \in \mathcal{C}$ such that

$$\mathrm{ad}_{\mathcal{C}}(\emptyset, \emptyset, O_R \longrightarrow^\star R_n \xrightarrow{i:\alpha} R_{n+1}) = x_{n+1}$$

and

$$[\![\varepsilon(R_{n+1})]\!] = \mathcal{C} \setminus x_{n+1}.$$

We have that

$$\mathrm{ad}_{\mathcal{C}}(\emptyset, \emptyset, O_R \longrightarrow^\star R_n \xrightarrow{i:\alpha} R_{n+1}) = \mathrm{ad}_{\mathcal{C}}(x_n, f_n, R_n \xrightarrow{i:\alpha} R_{n+1})$$

Hence we have that $x_{n+1} = x \cup \{e\}$, $f_{n+1} = f_n \cup \{e \leftrightarrow i\}$ and we have to show that there exists a unique $e \in \mathcal{C}$ such that $\ell(e) = \alpha$ and

$$[\![\varepsilon(R_{n+1})]\!] = \mathcal{C} \setminus (x_n \cup \{e\}).$$

However, if such an e exists then $e \in [\![\varepsilon(R_n)]\!]$ and

$$\mathcal{C} \setminus (x_n \cup \{e\}) = [\![\varepsilon(R_n)]\!] \setminus e.$$

Hence we reason on the transition $R_n \xrightarrow{i:\alpha} R_{n+1}$ to show that there exists a unique $\{e\} \in [\![\varepsilon(R_n)]\!]$ such that $[\![\varepsilon(R_{n+1})]\!] = [\![\varepsilon(R_n)]\!] \setminus e$. We consider only the case $\alpha = a$, the rest being similar. Using structural congruence it is possible to rewrite R_n and R_{n+1} as follows

$$R_n \equiv (m_1 \rhd a.P_1 \mid R_2)\backslash(b_1 \dots b_n) \qquad R_{n+1} \equiv (m_1 \rhd P_1 \mid R_2)\backslash(b_1 \dots b_n)$$

and hence, for $\varepsilon(R_2) = P_2$,

$$\varepsilon(R_n) = (a.P_1 \mid P_2)\backslash(b_1 \dots b_n) \qquad \varepsilon(R_{n+1}) = (P_1 \mid P_2)\backslash(b_1 \dots b_n).$$

We have then to show that

$$[\![(P_1 \mid P_2)\backslash(b_1 \dots b_n)]\!] = [\![(a.P_1 \mid P_2)\backslash(b_1 \dots b_n)]\!] \setminus e.$$

From Lemma 5 such an event exists. To show its uniqueness, consider $R_n \equiv m_1 \rhd a.P_1 \mid \big(m_2 \rhd a.P_2 \mid R_2\big)$. Either $m_1 = m_2$ in which case the process exhibits auto-concurrency, or $m_1 \neq m_2$ and in this case the condition $j <_{R_{n+1}} i \iff f_n(j) <_{x_n \cup \{e\}} e$ from the definition of the encoding, points to either m_1 or m_2.

Let us prove that $\forall x \in [\![(P_1 \mid P_2)\backslash(b_1 \dots b_n)]\!]$, $x \in [\![(a.P_1 \mid P_2)\backslash(b_1 \dots b_n)]\!] \setminus e$. The other direction is similar. Let us unfold the encoding of Definition 20 using the operations on configurations structures of Definition 17.

$$[\![(P_1 \mid P_2)\backslash(b_1 \dots b_n)]\!] = ([\![P_1]\!] \times [\![P_2]\!]) \upharpoonright \cup_{1 \leqslant i \leqslant n} E_{b_i}$$
$$[\![(a.P_1 \mid P_2)\backslash(b_1 \dots b_n)]\!] = ([\![a.P_1]\!] \times [\![P_2]\!]) \upharpoonright \cup_{1 \leqslant i \leqslant n} E_{b_i}$$

If $x \in ([\![P_1]\!] \times [\![P_2]\!]) \upharpoonright \cup_{1 \leqslant i \leqslant n} E_{b_i}$ then

$$\forall e \in x, \ell(e) \notin \{b_i, \bar{b_i}, 0\}. \tag{5}$$

Hence $x \in ([\![P_1]\!] \times [\![P_2]\!])$. Let π_1, π_2 be the two projections defined by the product. Then

$$\pi_1(x) \in [\![P_1]\!] \text{ and } \pi_2(x) \in [\![P_2]\!]. \tag{6}$$

As $\pi_1(x) \in [\![P_1]\!]$, and from the definition of $[\![a.P_1]\!]$ we have that $\exists e_1$, $\ell(e_1) = a$ and such that $\{e_1\} \cup \pi_1(x) \in a.[\![P_1]\!]$. From Equation 6 we have that $\exists x_2 \in a.[\![P_1]\!] \times [\![P_2]\!]$ such that $\pi_1(x_2) = \{e_1\} \cup \pi_1(x)$ and $\pi_2(x_2) = \pi_2(x)$. Hence $\exists! e$ such that $\pi_1(e) = e_1$, $\pi_2(e) = \star$ and $x_2 = \{e\} \cup x$. From Equation 5 we have that $x_2 \in (a.[\![P_1]\!] \times [\![P_2]\!]) \upharpoonright \cup_{1 \leqslant i \leqslant n} E_{b_i}$. We infer that if $x \cup \{e\} \in (b_1 \dots b_n)(a.[\![P_1]\!] \times [\![P_2]\!])$ then $x \in [\![(b_1 \dots b_n)(a.P_1 \mid P_2)]\!] \setminus e$.

From $[\![\varepsilon(R)]\!] = \mathcal{C} \setminus x_n$, we have that $\forall y \in [\![\varepsilon(R)]\!]$, $y \cup x_n \in \mathcal{C}$. In particular $x_n \cup \{e\} \in \mathcal{C}$.

Let us denote $x_{n+1} = x_n \cup \{e\}$. Remains to show that $j <_{R_{n+1}} i \iff f(j) <_{x_{n+1}} e$. We show the implication $j <_{R_{n+1}} i \implies f_n(j) <_{x_{n+1}} e$ and consider the immediate order for $<_{R_{n+1}}$, as the order is transitive. From $j <_{R_{n+1}} i$, we have that $\langle i, a\rangle.\langle j, b\rangle \in R_{n+1}$, hence we retrieve a process R_k where $b.a.P' \in R_k$. Hence the events $f_n(j)$ and $f_n(i)$ are causaly dependent in the configuration structure of $[\![\varepsilon(R_k)]\!]$, and therefore causally dependent in $\mathcal{C}$. For the other direction $f_n(j) <_{x_{n+1}} e \implies j <_{R_{n+1}} i$ we show that $\ell(f(j))$ and $\ell(e)$ are causal in the origin process P, hence they are causally dependent in the memory of R_{n+1}.

Hence $\mathrm{ad}_{\mathcal{C}}(\emptyset, O_R \longrightarrow^\star R_n \xrightarrow{a} R_{n+1}) = x_n \cup \{e\}$ with $[\![\varepsilon(R_{n+1})]\!] = [\![O_{R_n}]\!] \setminus (x_n \cup \{e\})$. □

Remark 5 (On encoding RCCS)**.** Another encoding exists [14], but it is not compositional, since $[\![P_1 \mid P_2]\!]$ is not defined as an operation on $[\![P_1]\!]$ and $[\![P_2]\!]$. Compositionality is important for the definition of contexts in configurations structures, in particular for the definition of congruence (Definition 28).

Let us now define a transition relation on configurations structures, useful in showing the operational correspondence between terms of RCCS and their encoding.

Definition 26 (Reversible LTS in configurations structures)**.** Define $([\![P]\!], x) \xrightarrow{\ell(e)} ([\![P]\!], x \cup \{e\})$ for $x \cup \{e\} \in [\![P]\!]$. Similarly to Definition 3 we define $([\![P]\!], x) \overset{\ell(e)}{\rightsquigarrow} ([\![P]\!], x \setminus e)$, for some e such that $x \setminus e \in [\![P]\!]$.

We defined in Definition 25 the encoding of a process parametrically on a trace. The following proposition shows that any trace from O_R up to R leads to the same encoding.

Proposition 5. *For all singly labelled processes R there exists x a configuration in $[\![\varepsilon(O_R)]\!]$ such that $\forall \sigma : O_R \longrightarrow^\star R$, $[\![R]\!]_\sigma = x$ holds.*

Proof. Denote $\mathcal{C} = [\![\varepsilon(O_R)]\!] = (E, C, \ell)$. From the definition of $[\![R]\!]_\sigma$ it suffices to show that for any configurations $x, y \in \mathcal{C}$ such that there exists a label and order preserving bijection between the two and such that $\mathcal{C} \setminus x = \mathcal{C} \setminus y$, $x = y$ holds.

We prove it by induction on the size of x and y. Suppose that there exists two events e, e' such that $y = x \cup \{e\}$ and $z = x \cup \{e'\}$ are configurations of $\mathcal{C}$ as well, with $f : y \leftrightarrow z$. Since R is singly labelled (Definition 24), if $\ell(e) = \ell(e')$, then $e = e'$. □

Example 6. Consider the configuration structure in Figure 3c, encoding the process $P = a.(a \mid c) + b$. The process

$$S = \big(\langle 2, a, 0\rangle.\ \curlyvee .\langle 1, a, b\rangle \rhd 0\big) \mid \big(\langle 3, c, 0\rangle.\ \curlyvee .\langle 1, a, b\rangle \rhd 0\big)$$

can be reached on the trace $\sigma_1 : \varnothing \rhd P \xrightarrow{1:a} \xrightarrow{2:a} \xrightarrow{3:c} S$ or $\sigma_1 : \varnothing \rhd P \xrightarrow{1:a} \xrightarrow{3:c} \xrightarrow{2:a} S$. However both traces lead to the same encoding of S.

Hence we write $[\![R]\!]$ instead of $[\![R]\!]_\sigma$. It is an essential property to prove the existence of a bisimulation relation between a process and its encoding.

Lemma 6 (Operational correspondence between a R and $[\![R]\!]$)**.** *Let R a process and $[\![R]\!] = (\mathcal{C}, x)$ its interpretation.*

1. *$\forall \alpha$, S and $i \in \mathsf{I}$ such that $R \xrightarrow{i:\alpha} S$ then $[\![R]\!] \xrightarrow{\alpha} [\![S]\!]$;*
2. *$\forall \alpha$, S and $i \in \mathsf{I}$ such that $R \overset{i:\alpha}{\rightsquigarrow} S$ then $[\![R]\!] \overset{\alpha}{\rightsquigarrow} [\![S]\!]$;*
3. *$\forall e \in E$, $(\mathcal{C}, x) \xrightarrow{\ell(e)} (\mathcal{C}, x \cup \{e\})$ then $\exists S$, such that for some $i \in \mathsf{I}$, $R \xrightarrow{i:\alpha} S$ and $[\![S]\!] = (\mathcal{C}, x \cup \{e\})$.*
4. *$\forall e \in E$, $(\mathcal{C}, x) \overset{\ell(e)}{\rightsquigarrow} (\mathcal{C}, x \setminus e)$ then $\exists S$, such that for some $i \in \mathsf{I}$, $R \overset{i:\alpha}{\rightsquigarrow} S$ and $[\![S]\!] = (\mathcal{C}, x \setminus e)$.*

Proof. 1. As $R \xrightarrow{i:\alpha} S$, $O_R = O_S$, we have that $[\![S]\!] = (\mathcal{C}, x_s)$, where $x_S = \mathrm{ad}_{\mathcal{C}}(\emptyset, O_R \longrightarrow^\star S) = \mathrm{ad}_{\mathcal{C}}(\emptyset, O_R \longrightarrow^\star R \xrightarrow{\alpha} S) = x_R \cup \{e\}$ by Proposition 4. As $[\![R]\!] = (\mathcal{C}, x_R)$ it follows that $(\mathcal{C}, x_R) \xrightarrow{\alpha} (\mathcal{C}, x_S)$.

2. The proof for the backward direction is similar except that it uses the trace up to R. It uses Proposition 5, which allows us to backtrack on any path from the emptyset and leading to x_R.

3. From $(\mathcal{C}, x) \xrightarrow{\ell(e)} (\mathcal{C}, x \cup \{e\})$ we have that $x \cup \{e\} \in \mathcal{C}$. Then $\{e\} \in \mathcal{C} \setminus x$. From $[\![R]\!] = (\mathcal{C}, x)$ we have that $\mathcal{C} \setminus x = [\![\varepsilon(R)]\!]$, hence $\{e\} \in [\![\varepsilon(R)]\!]$. We use Lemma 5 and obtain that $\exists P$ such that $\varepsilon(R) \xrightarrow{\ell(e)} P$. Then due to the strong bisimulation between a RCCS term and its corresponding CCS term in Lemma 3, we have that, for some i, $R \xrightarrow{i:\alpha} S$, where $\varepsilon(S) = P$. That $[\![S]\!] = (\mathcal{C}, x \cup \{e\})$ follows from a similar argument to above and from Proposition 5. □

4. It is similar to the case above.

3. Contextual equivalence on configuration structures

In this section we introduce a notion of context for the configurations structures and then adapt the back-and-forth barbed bisimulation to configurations structures (Definition 28). We define hereditary history preserving bisimulation and use two families of relations, denoted F_i and B_i, to inductively approximate the bisimulation (Lemma 8). We use these relations to show that two processes are barbed congruent whenever their denotations are in the HHPB relation (Theorem 2). Once the HHPB has been probed to be a congruence (Proposition 8), one direction is straightforward , whereas the other is more technical and, as in CCS [24], follows by contradiction. It uses the relations F_i and B_i (Definition 31) to build contexts that discriminate processes that are not bisimilar.

3.1. Contexts for configurations structures

Contexts for configurations structures have never been defined as it is not clear what a configuration structure with a hole could be. However, if a structure $\mathcal{C}$ has an operational meaning, i.e. there exists P a process such that $\mathcal{C} = [\![P]\!]$, we use a CCS context $C[\cdot]$ to build a configuration structure $[\![C[P]]\!]$.

When analysing the reductions of a process in context, we need to know the contribution the process and the context have in the reduction. To this aim we associate to the context $C[\cdot]$ instantiated by a process P a projection morphism $\pi_{C,P} : [\![C[P]]\!] \to [\![P]\!]$ that retrieve in $[\![C[P]]\!]$ the parts of a configuration belonging to $[\![P]\!]$.

Following Proposition 1, we continue to consider only context made of parallel compositions, but the following definition can be extended to arbitrary contexts [2, Definition 46].

Definition 27. Let $C[\cdot]$ a context, and P a process. The *projection* $\pi_{C,P} : [\![C[P]]\!] \to [\![P]\!]$ is defined on the structure of C as follows:

- if $C[\cdot] = C'[\cdot] \mid P'$ then $\pi_{C,P} : [\![C'[P] \mid P']\!] \to [\![P]\!]$ is defined as $\pi_{C,P}(e) = \pi_{C',P}(\pi_1(e))$, where $\pi_1 : [\![C'[P] \mid P']\!] \to [\![C'[P]]\!]$ is the projection morphism defined by the product in Definition 17;
- if $C[\cdot] = [\cdot]$ then $\pi_{C,P} : [\![C[P]]\!] \to [\![P]\!]$ is the identity.

We naturally extend $\pi_{C,P}$ to configurations, and prove by case analysis that $\pi_{C,P} : [\![C[P]]\!] \to [\![P]\!]$ is a morphism.

3.2. Relation induced by barbed congruence on configurations structures

We define a relation on configurations structures that have an operational meaning and we show it is the relation induced by the barbed congruence in RCCS (Definition 13). We call the relation barbed back-and-forth congruence, to highlight its meaning, though it is not strictly speaking a congruence on configurations structures.

Definition 28 (Back-and-forth barbed congruence on configurations structures)**.** A *back-and-forth barbed bisimulation on configurations structures* is a symmetric relation $\mathcal{R} \subseteq C_1 \times C_2$ such that $(\emptyset, \emptyset) \in \mathcal{R}$, and if $(x_1, x_2) \in \mathcal{R}$, then

$$x_1 \overset{e_1}{\rightsquigarrow} x_1' \implies \exists x_2' \in C_2 \text{ s.t. } x_2 \overset{e_2}{\rightsquigarrow} x_2', \text{ with } \ell_1(e_1) = \ell_2(e_2) = \tau \text{ and } (x_1', x_2') \in \mathcal{R}; \quad \text{(back)}$$

$$x_1 \xrightarrow{e_1} x_1' \implies \exists x_2' \in C_2 \text{ s.t. } x_2 \xrightarrow{e_2} x_2', \text{ with } \ell_1(e_1) = \ell_2(e_2) = \tau \text{ and } (x_1', x_2') \in \mathcal{R}; \quad \text{(forth)}$$

$$\text{if } \exists e_1 \in E_1 \text{ s.t. } \ell_1(e_1) \neq \tau \text{ and } x_1 \xrightarrow{e_1} x_1' \text{ then } \exists x_2' \in C_2 \text{ s.t. } x_2 \xrightarrow{e_2} x_2', \text{ with } \ell_1(e_1) = \ell_2(e_2). \quad \text{(barbed)}$$

Let $\mathcal{C}_1 \dot{\sim}^{\tau} \mathcal{C}_2$ if and only if there exists a back-and-forth barbed bisimulation between $\mathcal{C}_1$ and $\mathcal{C}_2$.

Define $\sim^{\tau}$ the *back-and-forth barbed congruence* induced on configurations structures as a symmetric relation on configurations structures that have an operational meaning such that

$$[\![P_1]\!] \sim^{\tau} [\![P_2]\!] \iff \forall C, [\![C[P_1]]\!] \dot{\sim}^{\tau} [\![C[P_2]]\!].$$

We now prove that this relation is the relation induced on the encoding of processes by the barbed back-and-forth congruence (Definition 13). We begin by proving it in the non-contextual case. We remind the reader that, as we are going to manipulate encoding of RCCS terms, some restrictions on the terms applies (Remark 4).

Proposition 6. *For all P and Q, $\varnothing \rhd P \dot{\sim}^{\tau} \varnothing \rhd Q \iff [\![P]\!] \dot{\sim}^{\tau} [\![Q]\!]$.*

Proof.
- $\varnothing \rhd P \dot{\sim}^{\tau} \varnothing \rhd Q \implies [\![P]\!] \dot{\sim}^{\tau} [\![Q]\!]$. Let $\mathcal{R}_{\text{CCS}}$ be a back-and-forth barbed bisimulation between P and Q. We show that the following relation

$$\mathcal{R} = \{(x_1, x_2) \mid x_1 \in [\![P]\!], x_2 \in [\![Q]\!], \exists R, S \text{ s.t. } O_R = P, O_S = Q, R \,\mathcal{R}_{\text{CCS}}\, S \text{ and } [\![R]\!] = ([\![P]\!], x_1), [\![S]\!] = ([\![Q]\!], x_2)\}$$

is a back-and-forth barbed bisimulation between $[\![P]\!]$ and $[\![Q]\!]$.

We have that $(\emptyset, \emptyset) \in \mathcal{R}$, let $(x_1, x_2) \in \mathcal{R}$. We have to show that the conditions in Definition 28 hold. Suppose that $x_1 \xrightarrow{e_1} x_1' = x_1 \cup \{e_1\}$, for $\ell(e_1) = \tau$.

$$\begin{aligned}
x_1 \xrightarrow{e_1} x_1' &\implies ([\![P]\!], x_1) \xrightarrow{\ell(e_1)} ([\![P]\!], x_1') && \text{(From Definition 26)}\\
&\implies R \xrightarrow{i:\ell(e_1)} R' \text{ s.t. } [\![R']\!] = ([\![P]\!], x_1') && \text{(From Lemma 6)}\\
&\implies S \xrightarrow{i':\tau} S' && \text{(From } R \,\mathcal{R}_{\text{CCS}}\, S\text{)}\\
&\implies ([\![Q]\!], x_2) \xrightarrow{\ell(e_2)} ([\![Q]\!], x_2') && \text{(From Lemma 6)}
\end{aligned}$$

with $\ell(e_2) = \tau$. We have then $(x_1', x_2') \in \mathcal{R}$.

We proceed in a similar manner to show that conditions on the backward transitions and on the barbs hold.

- $[\![P]\!] \dot{\sim}^{\tau} [\![Q]\!] \implies \varnothing \rhd P \dot{\sim}^{\tau} \varnothing \rhd Q$. Let $\mathcal{R}_{\text{Conf}}$ be a back-and-forth barbed bisimulation between $[\![P]\!]$ and $[\![Q]\!]$. We show that the following relation

$$\begin{aligned}
\mathcal{R} = \{(R, S) \mid \varepsilon(O_R) = P, \varepsilon(O_S) = Q \text{ and } [\![R]\!] = ([\![P]\!], x_1),\\
[\![S]\!] = ([\![Q]\!], x_2), \text{ with } (x_1, x_2) \in \mathcal{R}_{\text{Conf}}\}
\end{aligned}$$

is a back-and-forth barbed bisimulation between P and Q. Let $(R, S) \in \mathcal{R}$, the following holds:

$$\begin{aligned}
R \xrightarrow{i:\tau} R' &\implies ([\![P]\!], x_1) \xrightarrow{\ell(e_1)} ([\![P]\!], x_1') && \text{(From Lemma 6)}\\
&\implies ([\![Q]\!], x_2) \xrightarrow{\ell(e_2)} ([\![Q]\!], x_2') && \text{(From } (x_1, x_2) \in \mathcal{R}_{\text{Conf}}\text{)}\\
&\implies S \xrightarrow{i':\tau} S' && \text{(From Lemma 6)}
\end{aligned}$$

where $x_1' = x_1 \cup \{e_1\}$, $x_2' = x_2 \cup \{e_2\}$ and $\ell(e_1) = \ell(e_2) = \tau$. We have that $O_{R'} = P$, $O_{S'} = Q$, $[\![R']\!] = ([\![P]\!], x_1')$, $[\![S']\!] = ([\![Q]\!], x_2')$ and $(x_1', x_2') \in \mathcal{R}_{\text{Conf}}$. Hence $(R', S') \in \mathcal{R}$.

To prove that the remaining conditions on the pair (R, S) holds as well is similar.

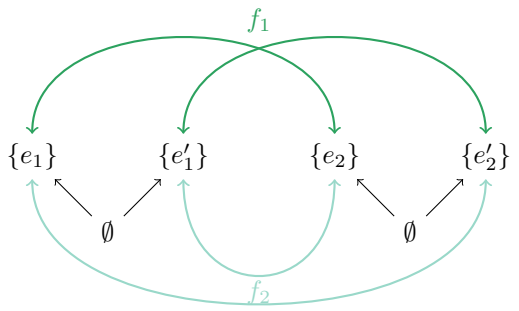
□

The contextual version of the proposition is straightforward.

Lemma 7. *For all singly labelled processes R and S, $O_R \sim^{\tau} O_S \iff [\![\varepsilon(O_R)]\!] \sim^{\tau} [\![\varepsilon(O_S)]\!]$.*

Proof.

$$\begin{aligned}
O_R \sim^{\tau} O_S &\iff \forall C[\cdot], C_{\Upsilon}[O_R] \dot{\sim}^{\tau} C_{\Upsilon}[O_S] && \text{(From Definition 13)}\\
&\iff \forall C[\cdot], \varnothing \rhd C[\varepsilon(O_R)] \dot{\sim}^{\tau} \varnothing \rhd C[\varepsilon(O_S)] && \text{(From Definition 11)}\\
&\iff \forall C[\cdot], [\![C[\varepsilon(O_R)]]\!] \dot{\sim}^{\tau} [\![C[\varepsilon(O_S)]]\!] && \text{(From Proposition 6)}\\
&\iff [\![\varepsilon(O_R)]\!] \sim^{\tau} [\![\varepsilon(O_S)]\!] && \text{(From Definition 28)}
\end{aligned}$$



$$\ell_1(e_1) = \ell_1(e'_1) = a \qquad \ell_2(e_2) = \ell_2(e'_2) = a$$

Figure 4: Two possible hereditary history preserving bisimulations

□

3.3. Inductive characterisation of HHPB

Similarly to the proof in CCS, the correspondence between a contextual equivalence and a non-contextual one necessitates to approximate HHPB with (a family of) inductive relations defined on configuration structures. If we are interested only in the forward direction (as in CCS), the inductive reasoning starts with the empty set, and constructs the bisimilarity relation by adding pairs of configurations reachable in the same manner from the empty set. However, to approximate HHPB, we need to have an inductive reasoning on the backward transition as well (Definition 31). These relations are of major importance to prove our main theorem (Theorem 2), as they re-introduce the possibility of an inductive reasoning thanks to a stratification of the HHPB relation.

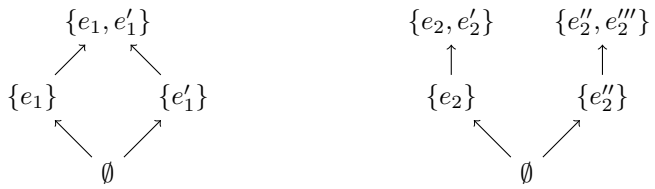
Definition 29 (Hereditary history preserving bisimilarity)**.** The hereditary history preserving bisimilarity, denoted $\sim$, is the union of all HHPB relations (Definition 23).

Remark 6 (On the uniqueness of hereditary history preserving bisimilarity)**.** Writing $\mathcal{C}_1 \sim \mathcal{C}_2$ is an abuse of notation as hereditary history preserving *bisimulations* are defined on $C_1 \times C_2 \times \mathcal{P}(E_1 \times E_2)$. Also the union of all bisimulations may contain triples that do not have "compatible" bijections. For instance, we have two possible bisimulations between the configurations structures of Figure 4:

$$f_1 = \{e_1 \leftrightarrow e_2, e'_1 \leftrightarrow e'_2\} \qquad f_2 = \{e_1 \leftrightarrow e'_2, e'_1 \leftrightarrow e_2\}$$

However, the bisimilarity relation contains both tuples $(\{e_1, e_2\}, \{e'_1, e'_2\}, f_1)$ and $(\{e_1, e_2\}, \{e'_1, e'_2\}, f_2)$.

We give an inductive characterisation of HHPB by reasoning on the structures up to a level: we ignore the configurations that have greater cardinality then



$\ell_1(e_1) = a$, $\ell_1(e_1') = b$

(a) $a \mid b$

$\ell_2(e_2) = \ell_2(e_2''') = a$, $\ell_2(e_2') = \ell_2(e_2'') = b$

(b) $a.b + b.a$

Figure 5: Encoding parallel and sum in configurations structures

the considered level. HHPB is then the relation obtained when we reach the top level. Hence we are able to detect, whenever two configurations structures are not HHPB, at which level the bisimulation does no longer hold.

In the following, denote $\mathrm{Card}(x)$ the cardinality of a set x.

Definition 30 (Maximal and top configurations)**.** A configuration $x \in \mathcal{C}$ is *maximal* if there is no configuration $y \in \mathcal{C}$ such that $x \subsetneq y$. If moreover $\forall y \in \mathcal{C}$, $\mathrm{Card}(y) \leqslant \mathrm{Card}(x)$ then x is a *top configuration*.

Definition 31 (F_i, B_i)**.** Given $\mathcal{C}_1$, $\mathcal{C}_2$ two configurations structures define, for all $x_1 \in C_1$, $x_2 \in C_2$ and f a label and order-preserving function:

$$(x_1, x_2, f) \in F_i \iff \begin{cases} \mathrm{Card}(x_1) = \mathrm{Card}(x_2) = i, & \text{if } x_1 \text{ and } x_2 \text{ are maximal} \\ \forall x_1', \exists x_2', x_1 \xrightarrow{e_1} x_1', x_2 \xrightarrow{e_1} x_2' \text{ and} & \\ f = f' \restriction x_1 \text{ s.t. } (x_1', x_2', f') \in F_{i+1} & \text{otherwise} \end{cases}$$

$$(x_1, x_2, f) \in B_i \iff \begin{cases} (x_1, x_2, f) \in F_i & \text{if } i = 0 \\ \forall x_1', \exists x_2', x_1 \overset{e_1}{\rightsquigarrow} x_1', x_2 \overset{e_1}{\rightsquigarrow} x_2' \text{ and} & \\ f' = f \restriction x_2 \text{ s.t. } (x_1', x_2', f') \in F_{i-1} \cap B_{i-1} & \text{otherwise} \end{cases}$$

The label and order-preserving function in the two relations helps to ensure that configurations that are in relation have the same labels and causal structure.

The relation B_i is built on top of F_i: it tests for the backward steps all the couples that passed the forward test. It should be remarked that, with this definition, $B_i \subseteq F_i$, but, at the price of slight modifications, one could have defined F_i on top of B_i.

Example 7. Consider the configurations structures in Figures 3b and 3c, the relations F_n are enough to discriminate them:

$$F_2 = \big(\{e_1, e_1'\}, \{e_2, e_2'\}\big); \big(\{e_1, e_1'\}, \{e_2'', e_2'''\}\big)$$
$$F_1 = \big(\{e_1\}, \{e_2\}\big); \big(\{e_1\}, \{e_2''\}\big)$$
$$F_0 = \emptyset$$

This intuitively is due to the fact that forward transitions are enough to discriminate $a + a.b$ and $a.b + a.b$. However for comparing the processes $a \mid b$ and $a.b + b.a$ whose configurations are in Figures 5a and 5b, we need the backward moves as well. Let us first build the F_n relations:

$$\begin{aligned} F_2 &= \big(\{e_1, e_1'\}, \{e_2, e_2'\}\big); \big(\{e_1, e_1'\}; \{e_2'', e_2'''\}\big) \\ F_1 &= \big(\{e_1\}, \{e_2\}\big); \big(\{e_1'\}; \{e_2''\}\big) \\ F_0 &= \big(\emptyset, \emptyset\big) \end{aligned}$$

We first construct the B_0 relation and then move up in the structures. In our example, the B_2 relation breaks the HHPB.

$$\begin{aligned} B_0 = F_0 &= \big(\emptyset, \emptyset\big) \\ B_1 &= \big(\{e_1\}, \{e_2\}\big); \big(\{e_1'\}; \{e_2''\}\big) \\ B_2 &= \emptyset \end{aligned}$$

The following proposition states that pairs of configurations are in a bisimulation relation if they have the same cardinality. It follows from the fact that any configuration is reachable from the empty set and that they have to mimick each other's step in the backward direction.

Proposition 7. *Let $\mathcal{C}_1 \sim \mathcal{C}_2$ be two configuration structures in a hereditary history preserving bisimulation and $x_1 \in \mathcal{C}_1$, $x_2 \in \mathcal{C}_2$ be two configurations.*

If $\exists f$ such that $(x_1, x_2, f) \in \{\sim\}$ then $\mathrm{Card}(x_1) = \mathrm{Card}(x_2)$.

Proof. It follows by induction on the trace $\emptyset \longrightarrow^\star x_1$. For every event in x_1, we have to add an event in x_2 in order to obtain that the pair x_1 and x_2 are in a HHPB relation. □

The following lemma, that will be handy to prove Theorem 2, implies that if for all $n \leqslant k$ the maximum cardinal considered, $F_n \cap B_n \neq \emptyset$, then $\cup_{n \leqslant k}(F_n \cap B_n)$ is a bisimulation.

Lemma 8. *For all $\mathcal{C}_1$, $\mathcal{C}_2$, if $\mathcal{C}_1 \sim \mathcal{C}_2$, then $\forall x_1 \in \mathcal{C}_1 (\exists x_2 \in \mathcal{C}_2, \exists f, (x_1, x_2, f) \in F_n \cap B_n) \iff (\exists x_2 \in \mathcal{C}_2, \exists f, (x_1, x_2, f) \in \sim)$, where* $\mathrm{Card}(x_1) = n$.

Proof. Let us denote $\mathcal{R}$ the relation $\sim$. One should first remark that $\mathcal{C}_1 \sim \mathcal{C}_2$ implies that $\forall x_1 \in \mathcal{C}_1, \exists x_2 \in \mathcal{C}_2$, and $\exists f$ such that $(x_1, x_2, f) \in \mathcal{R}$, as $(\emptyset, \emptyset, \emptyset) \in \mathcal{R}$ and all configurations are reachable from the empty set. The reader should notice that the $x_2 \in \mathcal{C}_2$ and f on both sides of the $\iff$ symbols may be different.

We prove that statement by induction on the cardinal of x_1.

$\mathrm{Card}(x_1) = 0$.

$\Rightarrow$ $x_2 \in \mathcal{C}_2$ s.t. $(\emptyset, x_2, f) \in \mathcal{R}$ follows by the definition of the bisimulation from $x_2 = \emptyset$ and $f = \emptyset$.

$\Leftarrow$ By definition, $F_0 \cap B_0 = F_0$. Since there exists $x_2 \in \mathcal{C}_2$ such that $(\emptyset, x_2, f) \in \mathcal{R}$, we know that any forward transition made by $\emptyset$ can be simulated by a forward transition from x_2, and that the elements obtained are in the relation $\mathcal{R}$. By an iterated use of this notion, we find top configurations $x_1^m \in \mathcal{C}_1$ and $x_2^m \in \mathcal{C}_2$ (that is, elements of maximum cardinality, k) such that $(x_1^m, x_2^m, f^m) \in \mathcal{R}$. By Proposition 7, x_1^m and x_2^m have the same cardinality, and $(x_1^m, x_2^m, f^m) \in F_k$. By just reversing the trace, we go backward and stay in relation F_i until $i = 0$, hence we found the x_2 and f we were looking for.

$\mathrm{Card}(x_1) = k+1$. As $\mathrm{Card}(x_1) > 0$, we know there exists x_1' such that $x_1 \overset{e_1}{\rightsquigarrow} x_1'$.

$\Rightarrow$ Let x_2 and f such that $(x_1, x_2, f) \in F_{k+1} \cap B_{k+1}$. We know that

$$\forall x_1', \exists x_2' \text{ and } f', x_1 \overset{e_1}{\rightsquigarrow} x_1', x_2 \overset{e_1}{\rightsquigarrow} x_2' \text{ and } (x_1', x_2', f') \in B_k$$
(By Definition of B_k)
$$\exists x_2'', f'', (x_1', x_2'', f'') \in \mathcal{R}$$
(By induction hypothesis)

And as $x_1' \xrightarrow{e_1} x_1$, there exists x_2''' and f''' such that $(x_1, x_2''', f''') \in \mathcal{R}$.

$\Leftarrow$ We prove it by contraposition: suppose that $\exists x_2, f$ such that $(x_1, x_2, f) \in \mathcal{R}$, we prove that $\forall x_2$, $(x_1, x_2, f) \notin F_{k+1} \cap B_{k+1}$ leads to a contradiction.

As $(x_1, x_2, f) \in \mathcal{R}$, we know that there exists x_1' and x_2', f' such that $x_1 \overset{e_1}{\rightsquigarrow} x_1'$, $x_2 \overset{e_1}{\rightsquigarrow} x_2'$ and $(x_1', x_2', f') \in \mathcal{R}$. By induction hypothesis, $\exists x_2''$ and $\exists f''$ such that $(x_1', x_2'', f'') \in F_k \cap B_k$. As $x_1' \xrightarrow{e_1} x_1$, $\exists x_2'''$ and $\exists f'''$ such that $x_2'' \xrightarrow{e_1} x_2'''$ and $(x_1, x_2''', f''') \in F_{k+1}$, by definition of F_k.

So $(x_1, x_2''', f''') \notin B_{k+1}$, but as $x_1 \overset{e_1}{\rightsquigarrow} x_1'$ and $x_2''' \overset{e_1}{\rightsquigarrow} x_2''$, and as moreover $(x_1', x_2'', f'') \in F_k \cap B_k$, we have that $(x_1, x_2''', f''') \in B_{k+1}$.

From this contradiction we know that we found the right element (x_2''') that is in relation with x_1 according to $F_{k+1} \cap B_{k+1}$. □

3.4. Contextual characterisation of HHPB

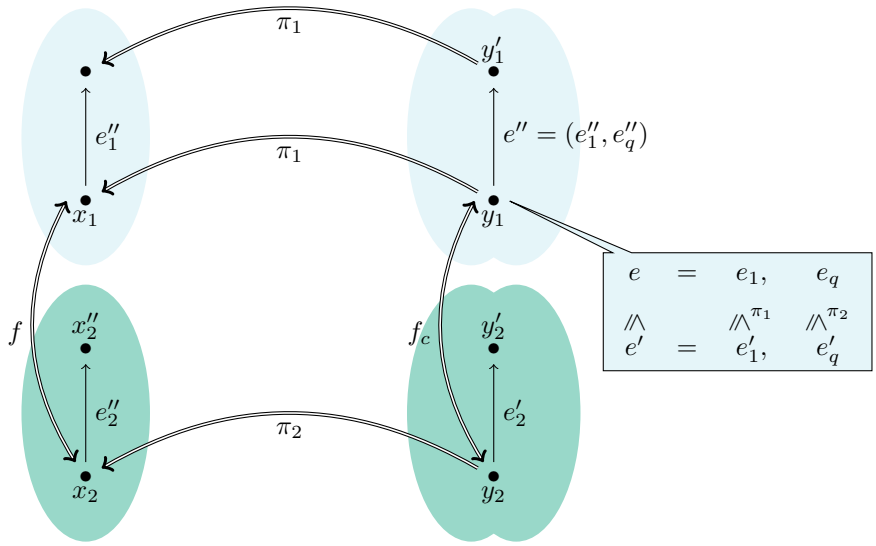
Proposition 8 (Hereditary history preserving bisimulation is a congruence)**.** *For all singly labelled P_1, P_2, $[\![P_1]\!] \sim [\![P_2]\!] \implies \forall C, [\![C[P_1]]\!] \sim [\![C[P_2]]\!]$.*

Proof. The proof amounts to carefully build a relation between $[\![C[P_1]]\!]$ and $[\![C[P_2]]\!]$ that reflects the known bisimulation between $[\![P_1]\!]$ and $[\![P_2]\!]$. Its uses that causality in a product is the result of the entanglement of the causality of its elements (Proposition 3).

Due to the restriction on the contexts we consider, we only have to prove that

$$\forall P_1, P_2, [\![P_1]\!] \sim [\![P_2]\!] \implies \forall Q, [\![P_1|Q]\!] \sim [\![P_2|Q]\!]$$

$[\![P_1]\!] = \langle E_1, C_1, \ell_1 \rangle \quad [\![P_1 \mid Q]\!] = \langle E_1', C_1', \ell_1' \rangle = ([\![P_1]\!] \times [\![Q]\!]) \upharpoonright E_1$



$[\![P_2]\!] = \langle E_2, C_2, \ell_2 \rangle \quad [\![P_2 \mid Q]\!] = \langle E_2', C_2', \ell_2' \rangle = ([\![P_2]\!] \times [\![Q]\!]) \upharpoonright E_2$

Figure 6: Configurations Structures by the end of the proof of Proposition 8

As $[\![P_1]\!] \sim [\![P_2]\!]$, there exists $\mathcal{R}$ a hereditary history preserving bisimulation (HHPB) between $[\![P_1]\!]$ and $[\![P_2]\!]$. Figure 6 introduces the variables names and types.

Define $\mathcal{R}_c \subseteq C_1' \times C_2' \times \mathcal{P}(E_1' \times E_2')$ as follows:

$$(y_1, y_2, f_c) \in \mathcal{R}_c \iff \begin{cases} (\pi_1(y_1), \pi_2(y_2), \pi_1 \circ f) \in \mathcal{R} \\ f_c(e) = (\pi_1 \circ f(e)), \pi_2(e)) \in y_2 \text{ for all } e \in y_1 \end{cases}$$

Informally (y_1, y_2, f_c) is in the relation $\mathcal{R}_c$ if there is (x_1, x_2, f) in $\mathcal{R}$ such that x_i is the first projection of y_i and such that f_c satisfies the property: for $(e_1, e_q) \in E_1'$, $f_c(e_1, e_q) = (f(e_1), e_q)$ and $(f(e_1), e_q) \in E_2'$.

Let us show that $\mathcal{R}_c$ is a HHPB between $\langle E_1', C_1', \ell_1' \rangle$ and $\langle E_2', C_2', \ell_2' \rangle$.

- $(\emptyset, \emptyset, \emptyset) \in \mathcal{R}_c$;
- For $(y_1, y_2, f_c) \in \mathcal{R}$ we show that f_c is label and order preserving bijection. We have that f_c is defined as $f_c(e) = (\pi_1 \circ f(e)), \pi_2(e))$, for some f label and order preserving bijection such that $(\pi_1(y_1), \pi_2(y_2), \pi_1 \circ f) \in \mathcal{R}$.

  That f_c is a bijection follows from f being a bijection.

  Let $e \in y_1$ with $\pi_1(e) = e_1$, $\pi_2(e) = e_q$, then $f_c(e) = (f(e_1), e_q)$ for some f_c s.t. $(\pi(y_1), \pi_2(y_2), f) \in \mathcal{R}$. We have that $\ell_1'(e) = (\ell_1(e_1), \ell_Q(e_q))$ and

$$\ell_2'(f_c(e)) = \ell_2'(f(e_1), e_q) = \big(\ell_2(f(e_1)), \ell_Q(e_q)\big)$$

As f is label preserving we get $\ell'_2(f_c(e)) = (\ell_1(e_1), \ell_Q(e_q))$, hence $\ell'_1(e) = \ell'_2(f_c(e))$.

Let us now show that for $e, e' \in y_1$, if $e \rightarrow_{y_1} e'$ then $f_c(e) \leqslant_{y_2} f_c(e')$. We denote $\pi_1(e) = e_1$, $\pi_2(e) = e_q$ and $\pi_1(e') = e'_1$, $\pi_2(e') = e'_q$. Then from Proposition 3

$$e \rightarrow_{y_1} e' \implies e_1 \leqslant_{\pi_1(y_1)} e'_1 \text{ or } e_q \leqslant_{\pi_2(y_1)} e'_q$$

We consider the case where $e_1 \leqslant_{\pi_1(y_1)} e'_1$. As f is order preserving we have that $f(e_1) \leqslant_{\pi_1(y_2)} f(e'_1)$. Then $(f(e_1), e_q) \leqslant_{x_2} (f(e'_1), e'_q)$, as the projections are order reflecting.

- Let $(y_1, y_2, f_c) \in \mathcal{R}_c$ and $y_1 \xrightarrow{e''} y'_1$, $y'_1 = y_1 \cup \{e''\}$. We consider only the case when $\pi_1(e'') = e''_1 \neq \star$, $\pi_2(e'') = e''_q \neq \star$ as the rest is similar. From the definition of the projections $\pi_1(y_1)$, $\pi_1(y'_1) \in C'_1$ and as $\pi_1(e'') = e''_1 \neq \star$, we have that $\pi_1(y'_1) = \pi_1(y_1) \cup \{e''_1\}$. We reason similarly on $\pi_2(y_1)$ and get

$$\pi_1(y_1) \xrightarrow{e''_1} \pi_1(y'_1) \text{ and } \pi_2(y_1) \xrightarrow{e''_q} \pi_2(y'_1). \tag{7}$$

From Equation 7 and as $(\pi_1(y_1), \pi_2(y_2), f) \in \mathcal{R}$, by definition of $\mathcal{R}_c$, we have that

$$\exists x'_2 \text{ s.t. } \pi_1(y_2) \xrightarrow{e''_2} x'_2 = x_2 \cup \{e''_2\} \tag{8}$$

and

$$f' = f \cup \{e''_1 \leftrightarrow e''_2\} \tag{9}$$

such that $(x'_1, x'_2, f') \in \mathcal{R}$.

Let us show that $\exists y'_2 \in (\llbracket P_2 \rrbracket \times \llbracket P_Q \rrbracket)$ with $y'_2 = y_2 \cup \{e'_2\}$ and $\pi_1(e'_2) = e''_2$, $\pi_2(e'_2) = e''_q$. From Equation 7 and Equation 8 we have that the projections are defines with $\pi_1(y'_2) = x'_2$, $\pi_2(y'_2) = \pi_2(y'_1)$. The axioms of finiteness and coincidence freeness on y'_2 follows from y_2 being a configuration in $(\llbracket P_2 \rrbracket \times \llbracket P_Q \rrbracket)$.

Let us show that $y'_2 \notin X_2$. We have that $y'_1 \notin X_1$. As $\ell(e''_1)$ and $\ell(e''_q)$ are compatible, then so are $\ell(e''_2)$ and $\ell(e''_q)$, hence $y_2 \cup \{(e''_2, e''_q)\} \notin X_2$.

Remains to show $(y'_1, y'_2, f'_c) \in \mathcal{R}$, where $f'_c = f_c \cup \{e''_1 \leftrightarrow e''_2\}$. We have that $(\pi_1(y'_1), \pi_1(y'_2), f') \in \mathcal{R}_c$ and from Equation 9 that $\pi_1 \circ f'_c = f'$. □

Theorem 2. *For all singly labelled P_1 and P_2, $\llbracket P_1 \rrbracket \sim \llbracket P_2 \rrbracket \iff \llbracket P_1 \rrbracket \sim^\tau \llbracket P_2 \rrbracket$.*

Proof. The left-to-right direction follows from the definition of $\sim$ (Definition 23) and from Proposition 8.

We prove the other direction by contraposition: let us suppose that $\llbracket P_1 \rrbracket \sim^\tau \llbracket P_2 \rrbracket$ and $\llbracket P_1 \rrbracket \not\sim \llbracket P_2 \rrbracket$, we will find a contradiction. Figure 7 presents the general shape of the configurations at the end of the proof.

As $\llbracket P_1 \rrbracket \not\sim \llbracket P_2 \rrbracket$, by Lemma 8, there exists $x_1 \in \llbracket P_1 \rrbracket$ such that $\forall x_2 \in \llbracket P_2 \rrbracket$, $(x_1, x_2, f) \notin F_n \cap B_n$ holds. Let us consider the largest such x_1. Note that

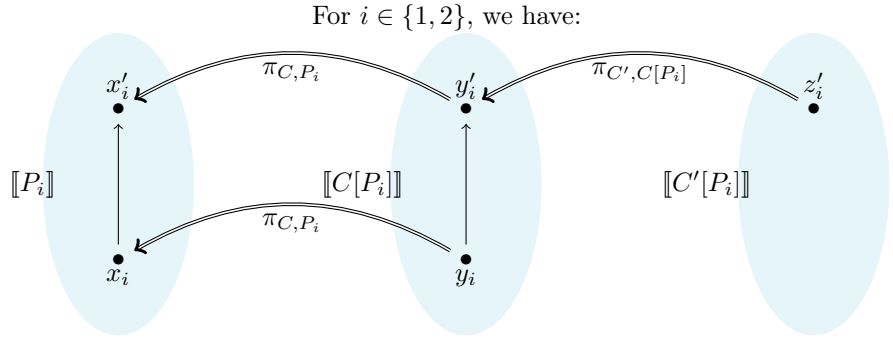


Figure 7: Configurations Structures by the end of the proof of Theorem 2

we consider only x_2 such that $\mathrm{Card}(x_1) = \mathrm{Card}(x_2) = n$, and that we use the projections $\pi_{C,P}$ (Definition 27) to separate the events of the process P from the events of the context C.

For any x_1 we define $C[\cdot] \coloneqq \prod_{e_i \in x_i} (\overline{\ell(e_i)} + c_{e_i}) \mid [\cdot]$ where $c_{e_i} \notin \mathrm{nm}(P_1) \cup \mathrm{nm}(P_2)$, such that the following holds

- $\exists y_1 \in [\![C[P_1]]\!]$ such that y_1 is closed, $\pi_{C,P_1}(y_1) = x_1$ and $y_1 \not\downarrow_{c_{e_i}}$ for all $e_i \in x_1$;
- We supposed that $[\![P_1]\!] \sim^\tau [\![P_2]\!]$, so $[\![C[P_1]]\!] \dot{\sim}^\tau [\![C[P_2]]\!]$. Hence $\exists\, \mathcal{R}$ a back-and-forth barbed bisimulation and $\exists y_2 \in [\![C[P_2]]\!]$ such that $(y_1, y_2) \in \mathcal{R}$ and $y_2 \not\downarrow_{c_{e_i}}$ for all $e_i \in x_1$.

We proceed as follows:

- we show that there exists f a label and order preserving bijection between x_1 and $\pi_{C,P_1}(y_2)$;
- then we show that $(x_1, \pi_{C,P_1}(y_2), f) \in F_n$ for f defined above;
- similarly we show that $(x_1, \pi_{C,P_1}(y_2), f) \in B_n$.

We denote $\pi_{C,P_1}(y_2)$ with x_2. We have by induction on the trace $\emptyset \longrightarrow^\star y_1$ that if y_1 is closed then y_2 is closed as well. Moreover we define a bijection $g : y_1 \to y_2$ that is order and label preserving. It follows again from an induction on the trace $\emptyset \longrightarrow^\star y_1$ and from $y_2 \not\downarrow_{c_{e_i}}$ for all $e_i \in x_1$.

We have that $\forall e_1, e'_1 \in x_1$, and $e_2 \in x_2$,

$$e_2 \in x_2 \iff e_1 \in x_2 \text{ and } \ell(e_1) = \ell(e_2) \tag{10}$$

$$e_1 <_{x_1} e'_1 \implies \pi^{-1}_{C,P_1}(e_1) <_{y_1} \pi^{-1}_{C,P_1}(e'_1) \tag{11}$$

$$\implies g(\pi^{-1}_{C,P_1}(e_1)) <_{y_2} g(\pi^{-1}_{C,P_1}(e'_1)) \tag{12}$$

Remark that (10) follows from $y_2 \not\downarrow_{c_{e_i}}$ and from the fact that if y_1 is closed we can show by contradiction that y_2 is closed as well. Secondly, (11) follows from the morphisms reflecting causality. Lastly, (12) follows from g being an order preserving bijection between y_1 and y_2.

For every events in $e_1'', e_2'' \in y_2$ such that $e_1'' \rightarrow_{y_2} e_2''$ from Proposition 3, either $\pi_{C,P_2}(e_1'') \leqslant_{\pi_{C,P_2}(y_2)} \pi_{C,P_2}(e_2'')$ or the projection of the two events are causal dependent in the context. However, the context does not induce any causality between the events. As $\pi_{C,P_2}(e_1'') \leqslant_{\pi_{C,P_2}(y_2)} \pi_{C,P_2}(e_2'')$, we have that there exists f a label and order preserving bijection between x_1 and $\pi_{C,P_1}(y_2)$.

Let us now prove that $(x_1, x_2, f) \in F_{n+1}$. There are two cases:

$$\nexists x_1', x_1 \xrightarrow{e_1} x_1', \exists x_2', x_2 \xrightarrow{e_2} x_2' \tag{13}$$

$$\exists x_1', x_1 \xrightarrow{e_1} x_1', \forall x_2', x_2 \xrightarrow{e_2} x_2' \text{ and } (x_1', x_2', f') \notin F_k \tag{14}$$

The implication (13) is easier: if $\exists x_2', x_2 \xrightarrow{e_2} x_2'$, then, as a context cannot remove transitions from the original process, $\exists y_2', y_2 \xrightarrow{(e_2,\star)} y_2'$. As $[\![C[P_2]]\!] \dot{\sim}^{\tau} [\![C[P_1]]\!]$, $\exists y_1', y_1 \xrightarrow{(e_1,\star)} y_1'$, and a similar argument on the context shows that $\exists x_1', x_1 \xrightarrow{e_1} x_1'$. Hence a contradiction.

To prove (14) requires more work. First, let $C'[\cdot] \coloneqq C[\cdot] \mid (\overline{\ell(e_1)} + c_{e_1})$. By induction hypothesis, there exists $z_1' \in [\![C'[P_1]]\!]$ such that z_1' is closed, $\pi_{C',C[P_1]}(z_1') = y_1'$ and $z_1' \not\downarrow_{c_{e_i}}$ and $z_1' \not\downarrow_{c_{e_1}}$ for all $e_i \in x_1$.

By hypothesis, $[\![P_1]\!] \sim^{\tau} [\![P_2]\!]$, hence there exists $\mathcal{R}'$ a back-and-forth barbed bisimulation between $[\![C'[P_1]]\!]$ and $[\![C'[P_2]]\!]$. It implies that $\exists z_2'$ such that $z_2 \in [\![C'[P_2]]\!]$ and $z_2' \not\downarrow_{c_{e_i}}$ and $z_2' \not\downarrow_{c_{e_1}}$ for all $e_i \in x_1$.

Using a similar argument to above we have that z_2' is closed and that there exists a bijection h between z_1' and z_2'.

Let us denote the projection $\pi_{C',P_2}(z_2')$ as x_2'. We infer using the fact that z_2' is closed and that $z_2' \not\downarrow_{c_{e_1}}$ that $\exists e_2' \in x_2'$ such that $\ell(e_2') = \ell(e_1)$.

As there exists a label and order preserving bijection h' between z_1' and z_2', and as we forbid auto concurrency and ambiguous non-deterministic sum (Remark 4), we conclude that $x_2' \setminus \{e_2'\} = x_2$, for $\pi_{C',P_2}(z_2') = x_2'$.

Then we have $\pi_{C',P_1}(z_1') = x_1', \pi_{C',P_2}(z_2') = x_2'$ and $f \cup \{e_1' \leftrightarrow e_2'\}$ a bijection between the two. As we supposed that x_1 is the largest configuration for which the HHPB breaks we get that $\exists x_2''$ such that $(x_1', x_2'', f'') \in F_{n+1}$. But such an x_2'' is unique since P_2 is singly labelled. Thus we conclude that $(x_1', x_2', f \cup \{e_1' \leftrightarrow e_2'\}) \in F_{n+1}$.

The proof that $(x_1, x_2, f) \in B_n$ goes along the line of (and uses) the proof that $(x_1, x_2, f) \in F_n$. □

Remark 7 (On Theorem 2)**.** Note that Theorem 2 is a result on RCCS processes that have *an empty memory*. It is a consequence of HHPB and the back-and-forth barbed congruence on configuration structure (Definition 28) being defined on configurations structures, and not on the tuples of configurations structures and configurations. However, we need the reversible setting to simulate the back-and-forth behaviour that we acquire when moving to configurations structures. The

result above then should be read as: *reversible process with* an empty memory *are barbed congurent if and only if their encodings in configurations structures are in a HHPB relation.*

To make the result more general and include any reversible process we need to reformulate it as follows.

Conjecture 1. *If $R \sim^{\tau} S$ such that $[\![R]\!] = (\mathcal{C}_R, x_R)$ and $[\![S]\!] = (\mathcal{C}_S, x_S)$ then there exists $\mathcal{R}$ a HHPB between $\mathcal{C}_R$ and $\mathcal{C}_S$ with $(x_R, x_S, f) \in \mathcal{R}$, for some f.*

We leave this as future work.

Conclusions and future work

We showed that, for a restricted class of RCCS processes (without recursion, auto-concurrency nor auto-conflict (Remark 4)) hereditary history preserving bisimilarity has a contextual characterisation in CCS. We used the barbed congruence defined on RCCS as the congruence of reference, adapted it to configurations structures and then showed a correspondence with HHPB. As a proof tool, we defined two inductively relations that approximate HHPB. Consequently we have that adding reversibility into the syntax helps in retrieving some of the discriminating power of configurations structures.

Note that one could prove the main result of the paper by showing that the bisimulation defined on the LTS of RCCS and the barbed congruence (Definition 13) equate the same terms. We chose to use configurations structures instead, as we plan to investigate other equivalences on reversible process algebra and their interpretations in configurations structures give interesting insights.

Weak equivalences. This work follows notable efforts [7, 18] to understand equivalences for reversible processes. There are numerous interesting continuations. A first one is to move to weak equivalences, which ignores silent moves τ and focus on the observable part of a process. This is arguably a more interesting relation than the strong one, in which processes have to mimick *exactly* each other's silent moves. Even if such a relation on configurations structures exists [17, 34] one still has to show that this is indeed the relation we expect.

In configuration structures, the adjective *weak* has sometimes [14, 28] a different meaning: it stands for the ability to change the label and order preserving bijection as the relation grows, to modify choices that were made before this step. It would be interesting to understand what "weak" relations in this sense represent for reversible processes.

Insensitiveness to the direction of the transitions and irreversibility. The relations defined so far simulate forward (resp. backward) transitions only with forward (resp. backward) transitions, and only consider *forward* barb. Ignoring the direction of the transitions could introduce some fruitful liberality in the way processes simulate each other. Depending on the answer, $a + \tau.b$ and $a + b$ would be weakly bisimilar or not. A weak bisimulation that ignores the direction of transitions [18] already exists, but it equates a reversible process with all its

derivatives. Irreversible moves could play an important role in such equivalences and would help to understand what are the meaningful equivalences in the setting of transactions [21].

Reversibility is commonly used in transactional systems, i.e. participative computations where a commitment phase is reached whenever a consensus occurs. This has two effects: it forbids the further exploration of the solution space, and prevents all the participants to complete if a participant cancels the transaction [6]. Commitment is modelled as an *irreversible* action: such a feature is present in RCCS [5], but absent from our work. It could probably be implemented by adding a mechanism to "update" the origin of a term, and by "cutting" the configuration structure after an irreversible transition (in the spirit of the LTS of Definition 21). However, it remains to prove that those two actions would be equivalent.

Removing the limitations. Context—which plays a key role—raise questions on the memory handling of RCCS : what about a context that could fix the memory of an incoherent process?

Maybe of less interest but important for the generality of these results, one should include infinite processes as well. This needs a rework of the relations in Definition 31 used to approximate the HHPB. In configurations structures however one usually handles the recursive case by unfolding the process up to a finite level.

One way to retrieve the class of processes with auto-conflict and auto-concurrence could be to define bisimulations that take into account tagged labels. At the price of a verbose syntax, one could imagine being able to discriminate between configurations reached after firing events with the same labels, thus allowing to define configurations structures for arbitrary RCCS terms. Are relations taking into account those "localities" [35], which uniquely determine occurrences of a label, more discriminating the traditional bisimulations?

Lastly, we conjecture that HHPB is equivalent to a congruence relation on terms that do not exhibit auto-conflict. More precisely, we could imagine that congruent processes have isomorphic event structures, and that configurations structures are isomorphic if and only if they are in HHPB relation.

Acknowledgement

We would like to warmly thank D. Varacca and J. Krivine for the useful discussions as well as the referees of an earlier version [1] for their helpful remarks.

References

[1] C. Aubert, I. Cristescu, Reversible barbed congruence on configuration structures, in: S. Knight, A. Lluch Lafuente, I. Lanese, H. T. Vieira (Eds.), ICE 2015, Vol. 189 of Electronic Proceedings in Theoretical Computer Science, 2015, pp. 68–95. doi:10.4204/EPTCS.189.7.

[2] I. Cristescu, Operational and denotational semantics for the reversible π-calculus, Ph.D. thesis, Université Paris Diderot – Paris 7–Sorbonne Paris Cité (2015).
URL http://www.pps.univ-paris-diderot.fr/~ioana/these.pdf

[3] L. Bougé, On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes, Acta Informatica 25 (2) (1988) 179–201. doi:10.1007/BF00263584.

[4] C. A. R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.

[5] V. Danos, J. Krivine, Reversible communicating systems, in: P. Gardner, N. Yoshida (Eds.), CONCUR, Vol. 3170 of Lecture Notes in Computer Science, Springer, 2004, pp. 292–307. doi:10.1007/978-3-540-28644-8_19.

[6] V. Danos, J.-L. Krivine, F. Tarissan, Self-assembling trees, Electronic Notes in Theoretical Computer Science 175 (1) (2007) 19–32. doi:10.1016/j.entcs.2006.11.017.

[7] I. Phillips, I. Ulidowski, Reversibility and models for concurrency, Electronic Notes in Theoretical Computer Science 192 (1) (2007) 93–108. doi:10.1016/j.entcs.2007.08.018.

[8] M. Nielsen, G. D. Plotkin, G. Winskel, Petri nets, event structures and domains, in: G. Kahn (Ed.), Semantics of Concurrent Computation, Proceedings of the International Symposium, Evian, France, July 2-4, 1979, Vol. 70 of Lecture Notes in Computer Science, Springer, 1979, pp. 266–284. doi:10.1007/BFb0022474.

[9] R. J. van Glabbeek, G. D. Plotkin, Configuration structures, event structures and petri nets, Theoretical Computer Science 410 (41) (2009) 4111–4159. doi:10.1016/j.tcs.2009.06.014.

[10] G. Winskel, Event structures, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.), Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986, Vol. 255 of Lecture Notes in Computer Science, Springer, 1986, pp. 325–392. doi:10.1007/3-540-17906-2_31.

[11] K. Honda, N. Yoshida, On reduction-based process semantics, Theoretical Computer Science 151 (2) (1995) 437–486. doi:10.1016/0304-3975(95)00074-7.

[12] R. De Nicola, U. Montanari, F. W. Vaandrager, Back and forth bisimulations, in: J. C. M. Baeten, J. W. Klop (Eds.), CONCUR '90, Vol. 458 of Lecture Notes in Computer Science, Springer, 1990, pp. 152–165. doi:10.1007/BFb0039058.

[13] M. Nielsen, G. D. Plotkin, G. Winskel, Petri nets, event structures and domains, part I, Theoretical Computer Science 13 (1981) 85–108. doi:10.1016/0304-3975(81)90112-2.

[14] I. Phillips, I. Ulidowski, A hierarchy of reverse bisimulations on stable configuration structures, Mathematical Structures in Computer Science 22 (2) (2012) 333–372. doi:10.1017/S0960129511000429.

[15] M. A. Bednarczyk, Hereditary history preserving bisimulations or what is the power of the future perfect in program logics, Tech. rep., Instytut Podstaw Informatyki PAN filia w Gdańsku (1991).
URL http://www.ipipan.gda.pl/~marek/papers/historie.ps.gz

[16] P. Baldan, S. Crafa, A logic for true concurrency, Journal of the ACM 61 (4) (2014) 24. doi:10.1145/2629638.

[17] W. Vogler, Bisimulation and action refinement, Theoretical Computer Science 114 (1) (1993) 173–200. doi:10.1016/0304-3975(93)90157-0.

[18] I. Lanese, C. A. Mezzina, J. Stefani, Reversing higher-order pi, in: P. Gastin, F. Laroussinie (Eds.), CONCUR, Vol. 6269 of Lecture Notes in Computer Science, Springer, 2010, pp. 478–493. doi:10.1007/978-3-642-15375-4_33.

[19] R. Milner, Communication and Concurrency, PHI Series in computer science, Prentice-Hall, 1989.

[20] R. M. Amadio, Operational methods in concurrency, draft of lecture notes, Université Paris Diderot (2014).
URL http://www.pps.univ-paris-diderot.fr/~amadio/Ens/concurrency.pdf

[21] V. Danos, J. Krivine, Transactions in rccs, in: M. Abadi, L. de Alfaro (Eds.), CONCUR, Vol. 3653 of Lecture Notes in Computer Science, Springer, 2005, pp. 398–412. doi:10.1007/11539452_31.

[22] J. Krivine, Algèbres de processus réversible - programmation concurrente déclarative, Ph.D. thesis, Université Paris 6 & INRIA Rocquencourt (2006).
URL http://www.pps.univ-paris-diderot.fr/~jkrivine/homepage/Research_files/phd.pdf

[23] B. Accattoli, Evaluating functions as processes, in: R. Echahed, D. Plump (Eds.), TERMGRAPH 2013, Vol. 110 of EPTCS, 2013, pp. 41–55. doi:10.4204/EPTCS.110.6.

[24] R. Milner, D. Sangiorgi, Barbed bisimulation, in: W. Kuich (Ed.), ICALP, Vol. 623 of Lecture Notes in Computer Science, Springer, 1992, pp. 685–695. doi:10.1007/3-540-55719-9_114.

[25] J.-M. Madiot, Higher-order languages: dualities and bisimulation enhancements, Ph.D. thesis, École Normale Supérieure de Lyon, Università di Bologna (2015).
URL https://hal.archives-ouvertes.fr/tel-01141067

[26] G. Winskel, Event structure semantics for CCS and related languages, in: M. Nielsen, E. M. Schmidt (Eds.), ICALP, Vol. 140 of Lecture Notes in Computer Science, Springer, 1982, pp. 561–576. doi:10.1007/BFb0012800.

[27] G. Winskel, M. Nielsen, Models for concurrency, in: S. Abramsky, D. M. Gabbay, T. S. E. Maibaum (Eds.), Semantic Modelling, Vol. 4 of Handbook of Logic in Computer Science, Oxford University Press, 1995, pp. 1–148.

[28] R. J. van Glabbeek, U. Goltz, Equivalence notions for concurrent systems and refinement of actions (extended abstract), in: A. Kreczmar, G. Mirkowska (Eds.), MFCS, Vol. 379 of Lecture Notes in Computer Science, Springer, 1989, pp. 237–248. doi:10.1007/3-540-51486-4_71.

[29] A. Joyal, M. Nielsen, G. Winskel, Bisimulation from open maps, Information and Computation 127 (2) (1996) 164–185. doi:10.1006/inco.1996.0057.

[30] I. Phillips, I. Ulidowski, Reversing algebraic process calculi, The Journal of Logic and Algebraic Programming 73 (1-2) (2007) 70–96. doi:10.1016/j.jlap.2006.11.002.

[31] R. J. van Glabbeek, History preserving process graphs, draft, Stanford University (1996).

[32] R. J. van Glabbeek, U. Goltz, Refinement of actions and equivalence notions for concurrent systems, Acta Informatica 37 (4/5) (2001) 229–327. doi:10.1007/s002360000041.

[33] G. Boudol, I. Castellani, A non-interleaving semantics for CCS based on proved transitions, Research Report RR-0919, INRIA (1988).
URL https://hal.inria.fr/inria-00075636

[34] M. P. Fiore, G. L. Cattani, G. Winskel, Weak bisimulation and open maps, in: LICS, IEEE Computer Society, 1999, pp. 67–76. doi:10.1109/LICS.1999.782590.

[35] G. Boudol, I. Castellani, A non-interleaving semantics for CCS based on proved transitions, Fundamenta Informaticae 11 (1988) 433–452, see also [33].