Explicit Identifiers and Contexts in Reversible Concurrent Calculus*

Clément Aubert[1][0000−0001−6346−3043] and Doriana Medić[2][0000−0002−7163−5375]

[1] School of Computer & Cyber Sciences, Augusta University, USA,
caubert@augusta.edu
[2] Focus Team/University of Bologna, Inria, Sophia Antipolis, France,
doriana.medic@gmail.com

Abstract. Existing formalisms for the algebraic specification and representation of networks of reversible agents suffer some shortcomings. Despite multiple attempts, reversible declensions of the Calculus of Communicating Systems (CCS) do not offer satisfactory adaptation of notions usual in "forward-only" process algebras, such as replication or context. Existing formalisms disallow the "hot-plugging" of processes during their execution in contexts with their own past. They also assume the existence of "eternally fresh" keys or identifiers that, if implemented poorly, could result in unnecessary bottlenecks and look-ups involving all the threads. In this paper, we begin investigating those issues, by first designing a process algebra endowed with a mechanism to generate identifiers without the need to consult with the other threads. We use this calculus to recast the possible representations of non-determinism in CCS, and as a by-product establish a simple and straightforward definition of concurrency. Our reversible calculus is then proven to satisfy expected properties. We also observe that none of the reversible bisimulations defined thus far are congruences under our notion of "reversible" contexts.

Keywords: Formal semantics· Process algebras and calculi· Context for reversible calculi

1 Introduction: Filling the Blanks in Reversible Process Algebras

Reversibility's Future is intertwined with the development of formal models for analyzing and certifying concurrent behaviors. Even if the development of quantum computers [30], CMOS adiabatic circuits [18] and computing biochemical systems promise unprecedented efficiency or "energy-free" computers, it would be a mistake to believe that when one of those technologies—each with their own connection to reversibility—reaches a mature stage, distribution of the computing capacities will become superfluous. On the opposite, the future probably resides in connecting together computers using different paradigms (i.e., "traditional", quantum, biological, etc.), and possibly themselves heterogeneous (for instance

* This work has been supported by French ANR project DCore ANR-18-CE25-0007.

using the "classical control of quantum data" motto [37]). In this coming situation, "traditional" model-checking techniques will face an even worst state explosion problem in presence of reversibility, that e.g. the usual "back-tracking" methods will likely fail to circumvent. Due to the notorious difficulty of connecting heterogeneous systems correctly and the "volatile" nature of reversible computers—that can erase all trace of their actions—, it seems absolutely necessary to design languages for the specification and verification of reversible distributed systems.

Process Algebras offer an ideal touch of abstraction while maintaining implementable specification and verification languages. In the family of process calculi, the Calculus of Communicating Systems (CCS) [35] plays a particular role, both as seminal work and as direct root of numerous systems (e.g. π-[42], Ambient [33], applied [1] and distributed [23] calculi). Reversible CCS (RCCS) [15] and CCS with keys (CCSK) [38] are two extensions to CCS providing a better understanding of the mechanisms underlying reversible concurrent computation—and they actually turned out to be the two faces of the same coin [27]. Most [3,14,32,34]—if not all—of the later systems developed to enhance the expressiveness with some respect (rollback operator, name-passing abilities, probabilistic features) stem from one approach or the other. However, those two systems, as well as their extensions, both share the same drawbacks, in terms of missing features and missing opportunities.

An Incomplete Picture is offered by RCCS and CCSK, as they miss "expected" features despite repetitive attempts. For instance, no satisfactory notion of context was ever defined: the discussed notions [5] do not allow the "hot-plugging" of a process with a past into a context with a past as well. As a consequence, defining congruence is impossible, forbidding the study of bisimilarities—though they are at the core of process algebras [41]. Also, recursion and replication are different [36], but only recursion have been investigated [22,25] or mentioned [15,16], and only for "memory-less" processes. Stated differently, the study of the duplication of systems with a past has been left aside.

Opportunities Have Been Missed as previous process algebras are *conservative extensions of restricted versions of CCS*, instead of considering "a fresh start". For instance, reversible calculi inherited the sum operator in its guarded version: while this restriction certainly makes sense when studying (weak) bisimulations for forward-only models, we believe it would be profitable to suspend this restriction and consider *all* sums, to establish their specificities and interests in the reversible frame. Also, both RCCS and CCSK have impractical mechanisms for keys or identifiers: aside from supposing "eternal freshness"—which requires to "ping" all threads when performing a transition, creating a potential bottle-neck—, they also require to inspect, in the worst case scenario, *all the memories of all the threads* before performing a backward transition.

Our Proposal for "yet" another language is guided by the desire to "complete the picture", but starts from scratch instead of trying to "correct" existing systems[3]. We start by defining an "identified calculus" that sidesteps the previous limitations

[3] Of course, due credit should be given for those previous calculi, that strongly inspired ours, and into which our system can be partially embedded, cf. Sect. 3.3.

of the key and memory mechanisms and considers multiple declensions of the sum: 1. the summation [35, p. 68], that we call "non-deterministic choice" and write $\between$, [44], 2. the guarded sum, $+$, and 3. the internal choice, $\sqcap$, inspired from the Communicating Sequential Processes (CSP) [24]—even if we are aware that this operator can be represented [2, p. 225] in forward systems, we would like to re-consider all the options in the reversible set-up, where "representation" can have a different meaning. Our formalism meets the usual criterion, and allows to sketch interesting definitions for contexts, that allows to prove that, even under a mild notion of context, the usual bisimulation for reversible calculi is not a congruence. As a by-product, we obtain a notion of concurrency, both for forward and forward-and-backward calculi, that rests solely on identifiers and can be checked locally.

Our Contribution tries to lay out a solid foundation to study reversible process algebras in all generality, and opens some questions that have been left out. Our detailed frame explicits aspects not often acknowledged, but does not yet answer questions such as "what is the right structural *congruence* for reversible calculi" [7]: while we can define a structural *relation* for our calculus, we would like to get a better take on what a congruence for reversible calculi is before committing. How our three sums differ and what benefits they could provide is also left for future work, possibly requiring a better understanding of non-determinism in the systems we model. Another direction for future work is to study new features stemming from reversibility, such as the capacity of distinguishing between multiple replications, based on how they replicate the memory mechanism allowing to reverse the computation.

All proofs and some ancillary definitions are in the extended version [8].

2 Forward-Only Identified Calculus With Multiple Sums

We enrich CCS's processes and labeled transition system (LTS) with identifiers needed to define reversible systems: indeed, in addition to the usual labels, the reversible LTS developed thus far all annotate the transition with an additional key or identifier that becomes part of the memory. This development can be carried out independently of the reversible aspect, and could be of independent interest. Our formal "identifier structures" allows to precisely define how such identifiers could be generated while guaranteeing eternal freshness of the identifiers used to annotate the transitions (Lemma 1) of our calculus that extends CCS conservatively (Lemma 2).

2.1 Preamble: Identifier Structures, Patterns, Seeds and Splitters

Definition 1 (Identifier structure and pattern). *An* identifier structure $\mathsf{IS} = (\mathsf{I}, \gamma, \oplus)$ *is s.t.*

- I *is an infinite set of* identifiers*, with a partition between infinite sets of* atomic identifiers I_a *and* paired identifiers I_p*, i.e.* $\mathsf{I}_a \cup \mathsf{I}_p = \mathsf{I}$*,* $\mathsf{I}_a \cap \mathsf{I}_p = \emptyset$*,*

– $\gamma : \mathbb{N} \to \mathsf{I}_a$ *is a bijection called a* generator*,*
– $\oplus : \mathsf{I}_a \times \mathsf{I}_a \to \mathsf{I}_p$ *is a bijection called a* pairing function*.*

Given an identifier structure IS*, an* identifier pattern ip *is a tuple* (c, s) *of integers called* current *and* step *such that* $s > 0$*. The* stream *of atomic identifiers generated by* (c, s) *is* $\mathsf{IS}(c, s) = \gamma(c), \gamma(c + s), \gamma(c + s + s), \gamma(c + s + s + s), \ldots$.

Example 1. Traditionally, a pairing function is a bijection between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N}$, and the canonical examples are Cantor's bijection and $(m, n) \mapsto 2^m(2n + 1) - 1$ [40,43]. Let p be any of those pairing function, and let $\mathrm{p}^-(m, n) = -(\mathrm{p}(m, n))$.

Then, $\mathsf{I}\mathbb{Z} = (\mathbb{Z}, \mathrm{id}_{\mathbb{N}}, \mathrm{p}^-)$ is an identifier structure, with $\mathsf{I}_a = \mathbb{N}$ and $\mathsf{I}_p = \mathbb{Z}^-$. The streams $\mathsf{I}\mathbb{Z}(0, 2)$ and $\mathsf{I}\mathbb{Z}(1, 2)$ are the series of even and odd numbers.

We now assume given an identifier structure IS and use $\mathsf{I}\mathbb{Z}$ in our examples.

Definition 2 (Compatible identifier patterns). *Two identifier patterns* ip_1 *and* ip_2 *are* compatible*,* $\mathsf{ip}_1 \perp \mathsf{ip}_2$*, if the identifiers in the streams* $\mathsf{IS}(\mathsf{ip}_1)$ *and* $\mathsf{IS}(\mathsf{ip}_2)$ *are all different.*

Definition 3 (Splitter). *A* splitter *is a function* $\cap$ *from identifier pattern to pairs of compatible identifier patterns, and we let* $\cap_1(\mathsf{ip})$ *(resp.* $\cap_2(\mathsf{ip})$*) be its first (resp. second) projection.*

We now assume that every identifier structure IS is endowed with a splitter.

Example 2. For $\mathsf{I}\mathbb{Z}$ the obvious splitter is $\cap(c, s) = ((c, 2 \times s), (c + s, 2 \times s))$. Note that $\cap(0, 1) = ((0, 2), (1, 2))$, and it is easy to check that the two streams $\mathsf{I}\mathbb{Z}(0, 2)$ and $\mathsf{I}\mathbb{Z}(1, 2)$ have no identifier in common. However, $(1, 7)$ and $(2, 13)$ are not compatible in $\mathsf{I}\mathbb{Z}$, as their streams both contain 15.

Definition 4 (Seed (splitter)). *A* seed s *is either an identifier pattern* ip*, or a pair of seeds* $(\mathsf{s}_1, \mathsf{s}_2)$ *such that all the identifier patterns occurring in* s_1 *and* s_2 *are pairwise compatible. Two seeds* s_1 *and* s_2 *are* compatible*,* $\mathsf{s}_1 \perp \mathsf{s}_2$*, if all the identifier patterns in* s_1 *and* s_2 *are compatible.*

We extend the splitter $\cap$ *and its projections* $\cap_j$ *(for* $j \in \{1, 2\}$*) to functions from seeds to seeds that we write* $[\cap]$ *and* $[\cap_j]$ *defined by*

$$[\cap](\mathsf{ip}) = \cap(\mathsf{ip}) \qquad [\cap_j](\mathsf{ip}) = \cap_j(\mathsf{ip})$$
$$[\cap](\mathsf{s}_1, \mathsf{s}_2) = ([\cap](\mathsf{s}_1), [\cap](\mathsf{s}_2)) \qquad [\cap_j](\mathsf{s}_1, \mathsf{s}_2) = ([\cap_j](\mathsf{s}_1), [\cap_j](\mathsf{s}_2))$$

Example 3. A seed over $\mathsf{I}\mathbb{Z}$ is $(\mathrm{id} \times \cap)(\cap(0, 1)) = ((0, 2), ((1, 4), (3, 4)))$.

2.2 Identified CCS and Unicity Property

We will now discuss and detail how a general version of (forward-only) CCS can be equipped with identifiers structures so that every transition will be labeled not only by a (co-)name, τ or υ[4], but also by an identifier that is guaranteed to be unique in the trace.

[4] We use this label to annotate the "internally non-deterministic" transitions introduced by the operator $\sqcap$. It can be identified with τ for simplicity if need be, and as τ, it does not have a complement.

Definition 5 (Names, co-names and labels). *Let* $\mathsf{N} = \{a, b, c, \dots\}$ *be a set of* names *and* $\overline{\mathsf{N}} = \{\overline{a}, \overline{b}, \overline{c}, \dots\}$ *its set of* co-names. *We define the set of labels* $\mathsf{L} = \mathsf{N} \cup \overline{\mathsf{N}} \cup \{\tau, \upsilon\}$, *and use* α *(resp.* μ, λ*) to range over* L *(resp.* $\mathsf{L}\backslash\{\tau\}$, $\mathsf{L}\backslash\{\tau, \upsilon\}$*). The* complement *of a name is given by a bijection* $\bar{\cdot} : \mathsf{N} \to \overline{\mathsf{N}}$, *whose inverse is also written* $\bar{\cdot}$.

Definition 6 (Operators).

$P, Q ::= \lambda.P$	(Prefix)	$P \Cup Q$	(Non-deterministic choice)
$P \mid Q$	(Parallel Composition)	$(\lambda_1.P_1) + (\lambda_2.P_2)$	(Guarded sum)
$P\backslash\lambda$	(Restriction)	$P \sqcap Q$	(Internal choice)

As usual, the inactive process 0 *is not written when preceded by a prefix, and we call* P *and* Q *the "threads" in a process* $P \mid Q$.

The labeled transition system (LTS) for this version of CCS, that we denote $\xrightarrow{\alpha}$, can be read from Fig. 1 by removing the seeds and the identifiers. Now, to define an identified declension of that calculus, we need to describe how each thread of a process can access its own identifier pattern to independently "pull" fresh identifiers when needed, without having to perform global look-ups. We start by defining how a seed can be "attached" to a CCS process.

Definition 7 (Identified process). *Given an identifier structure* IS, *an* identified process *is a CCS process* P *endowed with a seed* s *that we denote* $\mathsf{s} \circ P$.

We assume fixed a particular identifier structure $\mathsf{IS} = (\mathsf{I}, \gamma, \oplus, \cap)$, and now need to introduce how we "split" identifier patterns, to formalize when a process evolves from e.g. $\mathsf{ip} \circ a.(P \mid Q)$ that requires only one identifier pattern to $(\mathsf{ip}_1, \mathsf{ip}_2) \circ P \mid Q$, that requires two—because we want P and Q to be able to pull identifiers from respectively ip_1 and ip_2 without the need for an agreement. To make sure that our processes are always "well-identified" (Definition 10), i.e. with a matching number of threads and identifier patterns, we introduce an helper function.

Definition 8 (Splitter helper). *Given a process* P *and an identifier pattern* ip, *we define*

$$\cap^?(\mathsf{ip}, P) = \begin{cases} (\cap^?(\cap_1(\mathsf{ip}), P_1), \cap^?(\cap_2(\mathsf{ip}), P_2)) & \text{if } P = P_1 \mid P_2 \\ \mathsf{ip} \circ P & \text{otherwise} \end{cases}$$

and write e.g. $\cap^? \mathsf{ip} \circ a \mid b$ *for the "recomposition" of the pair* $\cap^?(\mathsf{ip}, a \mid b) = (\cap_1(\mathsf{ip}) \circ a, \cap_2(\mathsf{ip}) \circ b)$ *into the identified process* $(\cap_1(\mathsf{ip}), \cap_2(\mathsf{ip})) \circ a \mid b$.

Note that in the definition below, only the rules act., $+$ and $\sqcap$ can "uncover" threads, and hence are the only place where $\cap^?$ is invoked.

Definition 9 (ILTS). *We let the* identified labeled transition system *between identified processes be the union of all the relations* $\xrightarrow{i:\alpha}$ *for* $i \in \mathsf{I}$ *and* $\alpha \in \mathsf{L}$ *of Fig. 1. Structural relation is as usual [8] but will not be used.*

Action and Restriction

$$\frac{}{(c,s)\circ\lambda.P \xrightarrow{\gamma(c):\lambda} \cap^?(c+s,s)\circ P}\ \text{act.} \qquad a\notin\{\alpha,\bar{\alpha}\}\ \frac{\mathsf{s}\circ P \xrightarrow{i:\alpha} \mathsf{s}'\circ P'}{\mathsf{s}\circ P\backslash a \xrightarrow{i:\alpha} \mathsf{s}'\circ P'\backslash a}\ \text{res.}$$

Parallel Group

$$\mathsf{s}_1 \perp \mathsf{s}_2\ \frac{\mathsf{s}_1\circ P \xrightarrow{i_1:\lambda} \mathsf{s}_1'\circ P' \qquad \mathsf{s}_2\circ Q \xrightarrow{i_2:\overline{\lambda}} \mathsf{s}_2'\circ Q'}{(\mathsf{s}_1,\mathsf{s}_2)\circ P \mid Q \xrightarrow{i_1\oplus i_2:\tau} (\mathsf{s}_1',\mathsf{s}_2')\circ P' \mid Q'}\ \text{syn.}$$

$$\mathsf{s}_1 \perp \mathsf{s}_2\ \frac{\mathsf{s}_1\circ P \xrightarrow{i:\alpha} \mathsf{s}_1'\circ P'}{(\mathsf{s}_1,\mathsf{s}_2)\circ P \mid Q \xrightarrow{i:\alpha} (\mathsf{s}_1',\mathsf{s}_2)\circ P' \mid Q}\ |_{\text{L}}$$

$$\mathsf{s}_1 \perp \mathsf{s}_2\ \frac{\mathsf{s}_2\circ Q \xrightarrow{i:\alpha} \mathsf{s}_2'\circ Q'}{(\mathsf{s}_1,\mathsf{s}_2)\circ P \mid Q \xrightarrow{i:\alpha} (\mathsf{s}_1,\mathsf{s}_2')\circ P \mid Q'}\ |_{\text{R}}$$

Sum Group

$$\frac{\mathsf{s}\circ P \xrightarrow{i:\alpha} \mathsf{s}'\circ P'}{\mathsf{s}\circ P \mathbin{\textcircled{\scriptsize V}} Q \xrightarrow{i:\alpha} \mathsf{s}'\circ P'}\ \textcircled{\scriptsize V}_{\text{L}} \qquad \frac{\mathsf{s}\circ Q \xrightarrow{i:\alpha} \mathsf{s}'\circ Q'}{\mathsf{s}\circ P \mathbin{\textcircled{\scriptsize V}} Q \xrightarrow{i:\alpha} \mathsf{s}'\circ Q'}\ \textcircled{\scriptsize V}_{\text{R}}$$

$$\frac{}{(c,s)\circ(\lambda_1.P_1)+(\lambda_2.P_2) \xrightarrow{\gamma(c):\lambda_1} \cap^?(c+s,s)\circ P_1}\ +_{\text{L}}$$

$$\frac{}{(c,s)\circ(\lambda_1.P_1)+(\lambda_2.P_2) \xrightarrow{\gamma(c):\lambda_2} \cap^?(c+s,s)\circ P_2}\ +_{\text{R}}$$

$$\frac{\dfrac{}{(c,s)\circ P\sqcap Q \xrightarrow{\gamma(c):\upsilon} \cap^?(c+s,s)\circ P}\ \sqcap_{\text{L}}}{(c,s)\circ P\sqcap Q \xrightarrow{\gamma(c):\upsilon} \cap^?(c+s,s)\circ Q}\ \sqcap_{\text{R}}$$

Fig. 1. Rules of the identified labeled transition system (ILTS)

Example 4. The result of $\cap^?(0,1)\circ(a\mid(b\mid(c+d)))$ is $((0,2),((1,4),(3,4)))\circ(a\mid(b\mid(c+d))$, and a (resp. b, $c+d$) would get its next transition identified with 0 (resp. 1, 3).

Definition 10 (Well-identified process). *An identified process* $\mathsf{s}\circ P$ *is* well-identified *iff* $\mathsf{s}=(\mathsf{s}_1,\mathsf{s}_2)$, $P=P_1\mid P_2$ *and* $\mathsf{s}_1\circ P_1$ *and* $\mathsf{s}_2\circ P_2$ *are both well-identified, or* P *is not of the form* $P_1\mid P_2$ *and* s *is an identifier pattern.*

We now always assume that identified processes are well-identified.

Definition 11 (Traces). *In a transition* $t : \mathsf{s} \circ P \xrightarrow{i:\alpha} \mathsf{s}' \circ P'$*, process* $\mathsf{s} \circ P$ *is the* source*, and* $\mathsf{s}' \circ P'$ *is the* target *of transition* t*. Two transitions are* coinitial *(resp.* cofinal*) if they have the same source (resp. target). Transitions* t_1 *and* t_2 *are* composable*,* $t_1; t_2$*, if the target of* t_1 *is the source of* t_2*. A sequence of pairwise composable transitions is called a* trace*, written* $t_1; \cdots ; t_n$*.*

Lemma 1 (Unicity). *The trace of an identified process contains any identifier at most once, and if a transition has identifier* $i_1 \oplus i_2 \in \mathsf{I}_p$*, then neither* i_1 *nor* i_2 *occur in the trace.*

Lemma 2. *For all CCS process* P*,* $\exists \mathsf{s}$ *s.t.* $P \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} P' \Leftrightarrow (\mathsf{s} \circ P \xrightarrow{i_1:\alpha_1} \cdots \xrightarrow{i_n:\alpha_n} \mathsf{s}' \circ P')$*.*

Definition 12 (Concurrency and compatible identifiers). *Two coinitial transitions* $\mathsf{s} \circ P \xrightarrow{i_1:\alpha_1} \mathsf{s}_1 \circ P_1$ *and* $\mathsf{s} \circ P \xrightarrow{i_2:\alpha_2} \mathsf{s}_2 \circ P_2$ *are* concurrent *iff* i_1 *and* i_2 *are* compatible*,* $i_1 \perp i_2$*, i.e. iff*

$$\begin{cases} i_1 \neq i_2 & \text{if } i_1,\ i_2 \in \mathsf{I}_a \\ \text{there is no } i \in \mathsf{I}_a \text{ s.t. } i_1 \oplus i = i_2 & \text{if } i_1 \in \mathsf{I}_a, i_2 \in \mathsf{I}_p \\ \text{there is no } i \in \mathsf{I}_a \text{ s.t. } i \oplus i_2 = i_1 & \text{if } i_1 \in \mathsf{I}_p, i_2 \in \mathsf{I}_a \\ \text{for } i_1^1, i_1^2, i_2^1 \text{ and } i_2^2 \text{ s.t. } i_1 = i_1^1 \oplus i_1^2 \text{ and } i_2 = i_2^1 \oplus i_2^2, & \\ \qquad i_1^j \neq i_2^k \text{ for } j,k \in \{1,2\} & \text{if } i_1, i_2 \in \mathsf{I}_p \end{cases}$$

Example 5. The identified process $\mathsf{s} \circ P = ((0,2),(1,2)) \circ a + b \mid \overline{a}.c$ has four possible transitions:

$$t_1 : \mathsf{s} \circ P \xrightarrow{0:a} ((2,2),(1,2)) \circ 0 \mid \overline{a}.c \quad t_3 : \mathsf{s} \circ P \xrightarrow{1:\overline{a}} ((0,2),(3,2)) \circ a + b \mid c$$
$$t_2 : \mathsf{s} \circ P \xrightarrow{0:b} ((2,2),(1,2)) \circ 0 \mid \overline{a}.c \quad t_4 : \mathsf{s} \circ P \xrightarrow{0 \oplus 1:\tau} ((2,2),(3,2)) \circ 0 \mid c$$

Among them, only t_1 and t_3, and t_2 and t_3 are concurrent: transitions are concurrent when they do not use overlapping identifiers, not even as part of synchronizations.

Hence, concurrency becomes an "easily observable" feature that does not require inspection of the term, of its future transitions—as for "the diamond property" [29]—or of an intermediate relation on proof terms [11, p. 415]. We believe this contribution to be of independent interest, and it will help significantly the precision and efficiency of our forward-and-backward calculus in multiple respect.

3 Reversible and Identified CCS

A reversible calculus is always defined by a forward calculus and a backward calculus. Here, we define the forward part as an extension of the identified calculus of Definition 9, without copying the information about the seeds for conciseness,

but using the identifiers they provide. The backward calculus will require to make the seed explicit again, and we made the choice of having backward transitions re-use the identifier from their corresponding forward transition, and to restore the seed in its previous state. Expected properties are detailed in Sect. 3.2.

3.1 Defining the Identified Reversible CCS

Definition 13 (Memories and reversible processes). *Let $o \in \{\textcircled{0}, +, \sqcap\}$, $d \in \{\mathrm{L}, \mathrm{R}\}$, we define* memory events*,* memories *and* identified reversible processes *as follows, for $n \geqslant 0$:*

$$
\begin{aligned}
e &::= \langle i, \mu, ((o_1, P_1, d_1), \dots (o_n, P_n, d_n))\rangle && \text{(Memory event)}\\
m_s &::= e.m_s \mid \emptyset && \textit{(Memory stack)}\\
m_p &::= [m, m] && \textit{(Memory pair)}\\
m &::= m_s \mid m_p && \text{(Memory)}\\
R, S &::= \mathsf{s} \circ m \rhd P && \text{(Identified reversible processes)}
\end{aligned}
$$

In a memory event, if $n = 0$, then we will simply write $\_$. We generally do not write the trailing empty memories in memory stacks, e.g. we will write e instead of $e.\emptyset$.

Stated differently, our memory are represented as a stack or tuples of stacks, on which we define the following two operations.

Definition 14 (Operations on memories). *The* identifier substitution *in a memory event is written $e[i \leftarrow j]$ and is defined as substitutions usually are. The* identified insertion *is defined by*

$$
\langle i, \mu, ((o_1, P_1, d_1), \dots (o_n, P_n, d_n))\rangle \dashv\!\vdash_j (o, P, d) = \begin{cases} \langle i, \mu, ((o_1, P_1, d_1), \dots (o_n, P_n, d_n), (o, P, d))\rangle & \textit{if } i = j \\ \langle i, \mu, ((o_1, P_1, d_1), \dots (o_n, P_n, d_n))\rangle & \textit{otherwise} \end{cases}
$$

The operations are easily extended to memories by simply propagating them to all memory events.

When defining the forward LTS below, we omit the identifier patterns to help with readability, but the reader should assume that those rules are "on top" of the rules in Fig. 1. The rules for the backward LTS, in Fig. 3, includes both the seeds and memories, and is the exact symmetric of the forward identified LTS with memory, up to the condition in the parallel group that we discuss later. A bit similarly to the splitter helper (Definition 8), we need an operation that duplicates a memory if needed, that we define on processes with memory but without seeds for clarity.

Definition 15 (Memory duplication). *Given a process P and a memory m, we define*

$$
\delta^?(m, P) = \begin{cases} (\delta^?(m, P_1), \delta^?(m, P_2)) & \textit{if } P = P_1 \mid P_2 \\ m \rhd P & \textit{otherwise} \end{cases}
$$

and write e.g. $\delta^?(m) \rhd a \mid b$ *for the "recomposition" of the pair of identified processes* $\delta^?(m, a \mid b) = (\delta^?(m,a), \delta^?(m,b)) = (m \rhd a, m \rhd b)$ *into the process* $[m,m] \rhd a \mid b$.

Definition 16 (IRLTS). *We let the* identified reversible labeled transition system *between identified reversible processes be the union of all the relations* $\xrightarrow{i:\alpha}$ *and* $\overset{i:\alpha}{\rightsquigarrow}$ *for* $i \in \mathsf{I}$ *and* $\alpha \in \mathsf{L}$ *of Figures 2 and 3, and let* $\twoheadrightarrow = \rightarrow \cup \rightsquigarrow$. *Structural relation is as usual [8] but will not be used.*

Action and Restriction

$$\frac{}{m \rhd \lambda.P \xrightarrow{i:\lambda} \delta^?(\langle i, \lambda, \_\rangle.m) \rhd P}\ \text{act.}$$

$$a \notin \{\alpha, \bar{\alpha}\}\ \frac{m \rhd P \xrightarrow{i:\alpha} m' \rhd P'}{m \rhd P\backslash a \xrightarrow{i:\alpha} m' \rhd P'\backslash a}\ \text{res.}$$

Parallel Group

$$\frac{m_1 \rhd P \xrightarrow{i_1:\lambda} m_1' \rhd P' \qquad m_2 \rhd Q \xrightarrow{i_2:\overline{\lambda}} m_2' \rhd Q'}{[m_1, m_2] \rhd P \mid Q \xrightarrow{i_1 \oplus i_2:\tau} [m_1'[i_1 \leftarrow i_1 \oplus i_2], m_2'[i_2 \leftarrow i_2 \oplus i_1]] \rhd P' \mid Q'}\ \text{syn.}$$

$$\frac{m_1 \rhd P \xrightarrow{i:\alpha} m_1' \rhd P'}{[m_1, m_2] \rhd P \mid Q \xrightarrow{i:\alpha} [m_1', m_2] \rhd P' \mid Q}\ |_{\mathrm{L}}$$

Sum Group

$$\frac{m \rhd P \xrightarrow{i:\alpha} m' \rhd P'}{m \rhd (P \leqslant Q) \xrightarrow{i:\alpha} m' +\!\!+_i (\leqslant, Q, \mathrm{R}) \circ P'}\ \leqslant_{\mathrm{L}}$$

$$\frac{}{m \rhd ((\lambda_1.P_1) + (\lambda_2.P_2)) \xrightarrow{i:\lambda_1} \delta^?(\langle i, \lambda_1, (+, \lambda_2.P_2, \mathrm{R})\rangle.m) \rhd P_1}\ +_{\mathrm{L}}$$

$$\frac{}{m \rhd (P \sqcap Q) \xrightarrow{i:\upsilon} \delta^?(\langle i, \upsilon, (\sqcap, Q, \mathrm{R})\rangle.m) \rhd P}\ \sqcap_{\mathrm{L}}$$

The rules $|_{\mathrm{R}}$, $\leqslant_{\mathrm{R}}$, $+_{\mathrm{R}}$ and $\sqcap_{\mathrm{R}}$ can easily be inferred.

Fig. 2. Forward rules of the identified reversible labeled transition system (IRLTS)

In its first version, RCCS was using the whole memory as an identifier [15], but then it moved to use specific identifiers [4,31], closer in inspiration to CCSK's

keys [38]. This strategy, however, forces the act. rules (forward and backward) to check that the identifier picked (or present in the memory event that is being reversed) is not occurring in the memory, while our system can simply pick identifiers from the seed without having to inspect the memory, and can go backward simply by looking if the memory event has identifier in I_a—something enforced by requiring the identifier to be of the form $\gamma^{-1}(c)$. Furthermore, memory events and annotated prefixes, as used in RCCS and CCSK, do not carry information on whenever they synchronized with other threads: retrieving this information require to inspect all the memories, or keys, of all the other threads, while our system simply observes if the identifier is in I_p, hence enforcing a "locality" property. However, when backtracking, the memories of the threads need to be checked for "compatibility", otherwise i.e. $((1,2),(2,2))\circ[\langle 0,a,\_\rangle,\langle 0,a,\_\rangle]\rhd P \mid Q$ could backtrack to $((1,2),(0,2))\circ[\langle 0,a,\_\rangle,\emptyset]\rhd P \mid a.Q$ and then be stuck instead of $(0,1)\circ\emptyset\rhd a.(P\mid Q)$.

3.2 Properties: From Concurrency to Causal Consistency and Unicity

We now prove that our calculus satisfies typical properties for reversible process calculi [13,15,26,38]. Notice that showing that the forward-only part of our calculus is a conservative extension of CCS is done by extending Lemma 2 to accommodate memories and it is immediate. We give a notion of concurrency, and prove that our calculus enjoys the required axioms to obtain causal consistency "for free" [28]. All our properties, as commonly done, are limited to the reachable processes.

Definition 17 (Initial, reachable and origin process). *A process* $\mathsf{s}\circ m\rhd P$ *is* initial *if* $\mathsf{s}\circ P$ *is well-identified and if* $m=\emptyset$ *if* P *is not of the form* $P_1\mid P_2$*, or if* $m=[m_1,m_2]$*,* $P=P_1\mid P_2$ *and* $[\cap_j](\mathsf{s})\circ m_j\rhd P_j$ *for* $j\in\{1,2\}$ *are initial. A process* R *is* reachable *if it can be derived from an initial process, its* origin*, written* O_R*, by applying the rules in Figures 2 and 3.*

Concurrency To define concurrency in the forward *and backward* identified LTS is easy when both transitions have the same direction: forward transitions will adopt the definition of the identified calculus, and backward transitions will always be concurrent. More care is required when transitions have opposite directions, but the seed provides a good mechanism to define concurrency easily. In a nutshell, the forward transition will be in conflict with the backward transition when the forward identifier was obtained using the identifier pattern(s) that have been used to generate the backward identifier, something we call "being downstream". Identifying the identifier pattern(s) that have been used to generate an identifier in the memory is actually immediate:

Definition 18. *Given a backward transition* $t:\mathsf{s}\circ m\rhd P\overset{i:\alpha}{\rightsquigarrow}\mathsf{s}'\circ m'\rhd P'$*, we write* ip_t *(resp.* ip_t^1*,* ip_t^2*) for the unique identifier pattern(s) in* s' *such that* $i\in\mathsf{I}_a$ *(resp.* i_1 *and* i_2 *s.t.* $i_1\oplus i_2=i\in\mathsf{I}_p$*) is the first identifier in the stream generated by* ip_t *(resp. are the first identifiers in the streams generated by* ip_t^1 *and* ip_t^2*).*

Action and Restriction

$$\frac{}{\cap^?(\gamma^{-1}(i)+s,s)\circ\delta^?(\langle i,\lambda,\_\rangle.m)\rhd P \overset{i:\lambda}{\rightsquigarrow} (\gamma^{-1}(i),s)\circ m\rhd \lambda.P}\ \text{act.}$$

$$a\notin\{\alpha,\bar{\alpha}\}\ \frac{\mathsf{s}\circ m\rhd P\overset{i:\alpha}{\rightsquigarrow}\mathsf{s}'\circ m'\rhd P'}{\mathsf{s}\circ m\rhd P\backslash a\overset{i:\alpha}{\rightsquigarrow}\mathsf{s}'\circ m'\rhd P'\backslash a}\ \text{res.}$$

Parallel Group

The rule syn. (resp. $|_{\mathrm{L}}$) can be applied only if $\mathsf{s}_1\perp\mathsf{s}_2$ and $i_1\notin m_2'$, $i_2\notin m_1'$ (resp. $i\notin m_2$).

$$\frac{\begin{array}{c}\mathsf{s}_1\circ m_1[i_1\oplus i_2\leftarrow i_1]\rhd P\overset{i_1:\lambda}{\rightsquigarrow}\mathsf{s}_1'\circ m_1'\rhd P'\\ \mathsf{s}_2\circ m_2[i_2\oplus i_1\leftarrow i_2]\rhd Q\overset{i_2:\overline{\lambda}}{\rightsquigarrow}\mathsf{s}_2'\circ m_2'\rhd Q'\end{array}}{(\mathsf{s}_1,\mathsf{s}_2)\circ[m_1,m_2]\rhd P\mid Q\overset{i_1\oplus i_2:\tau}{\rightsquigarrow}(\mathsf{s}_1',\mathsf{s}_2')\circ[m_1',m_2']\rhd P'\mid Q'}\ \text{syn.}$$

$$\frac{\mathsf{s}_1\circ m_1\rhd P\overset{i:\alpha}{\rightsquigarrow}\mathsf{s}_1'\circ m_1'\rhd P'}{(\mathsf{s}_1,\mathsf{s}_2)\circ[m_1,m_2]\rhd P\mid Q\overset{i:\alpha}{\rightsquigarrow}(\mathsf{s}_1',\mathsf{s}_2)\circ[m_1',m_2]\rhd P'\mid Q}\ |_{\mathrm{L}}$$

Sum Group

$$\frac{\mathsf{s}\circ m\rhd P\overset{i:\alpha}{\rightsquigarrow}\mathsf{s}'\circ m'\rhd P'}{\mathsf{s}\circ m\mathbin{+\!\!+_i}(\mathbin{\mathbb{O}},Q,\mathrm{R})\rhd P\overset{i:\alpha}{\rightsquigarrow}\mathsf{s}'\circ m'\rhd(P'\mathbin{\mathbb{O}}Q)}\ \mathbb{O}_{\mathrm{L}}$$

$$\frac{}{\begin{array}{l}\cap^?(\gamma^{-1}(i)+s,s)\circ\delta^?(\langle i,\lambda_1,(+,\lambda_2.P_2,\mathrm{R})\rangle.m)\rhd P_1\overset{i:\lambda_1}{\rightsquigarrow}\\ \qquad(\gamma^{-1}(i),s)\circ m\rhd((\lambda_1.P_1)+(\lambda_2.P_2))\end{array}}\ +_{\mathrm{L}}$$

$$\frac{}{\cap^?(\gamma^{-1}(i)+s,s)\circ\delta^?(\langle i,v,(\sqcap,Q,\mathrm{R})\rangle.m)\rhd P\overset{i:v}{\rightsquigarrow}(\gamma^{-1}(i),s)\circ m\rhd(P\sqcap Q)}\ \sqcap_{\mathrm{L}}$$

The rules $|_{\mathrm{R}}$, $\mathbb{O}_{\mathrm{R}}$, $+_{\mathrm{R}}$ and $\sqcap_{\mathrm{R}}$ can easily be inferred.

Fig. 3. Backward rules of the identified reversible labeled transition system (IRLTS)

Definition 19 (Downstream). *An identifier i is* downstream *of an identifier pattern (c,s) if*

$$\begin{cases} i\in\mathsf{IS}(c,s) & \text{if } i\in\mathsf{I}_a\\ \text{there exists } j,k\in\mathsf{I}_a \text{ s.t. } j\oplus k=i \text{ and } j \text{ or } k \text{ is downstream of } (c,s) & \text{if } i\in\mathsf{I}_p\end{cases}$$

Definition 20 (Concurrency). *Two different coinitial transitions* $t_1 : \mathsf{s} \circ m \rhd P \xrightarrow{i_1:\alpha_1}\!\!\!\!\rightarrow \mathsf{s}_1 \circ m_1 \rhd P_1$ *and* $t_2 : \mathsf{s} \circ m \rhd P \xrightarrow{i_2:\alpha_2}\!\!\!\!\rightarrow \mathsf{s}_2 \circ m_2 \rhd P_2$ *are* concurrent *iff*

- t_1 *and* t_2 *are forward transitions and* $i_1 \perp i_2$ *(Definition 12);*
- t_1 *is a forward and* t_2 *is a backward transition and* i_1 *(or* i_1^1 *and* i_1^2 *if* $i_1 = i_1^1 \oplus i_1^2$*) is not downstream of* ip_{t_2} *(or* $\mathsf{ip}^1_{t_2}$ *nor* $\mathsf{ip}^2_{t_2}$*);*
- t_1 *and* t_2 *are backward transitions.*

Example 6. Re-using the process from Example 5 and adding the memories, after having performed t_1 and t_3, we obtain the process $\mathsf{s} \circ [m_1, m_2] \rhd 0 \mid c$, where $\mathsf{s} = ((2,2),(3,2))$, $m_1 = \langle 0, a, (+, b, \mathrm{R})\rangle$ and $m_2 = \langle 1, \overline{a}, \_\rangle$, that has three possible transitions:

$$t_1 : \mathsf{s} \circ [m_1, m_2] \rhd 0 \mid c \xrightarrow{3:c} ((2,2),(5,2)) \circ [m_1, \langle 3, c, \_\rangle.m_2] \rhd 0 \mid 0$$
$$t_2 : \mathsf{s} \circ [m_1, m_2] \rhd 0 \mid c \overset{1:\overline{a}}{\rightsquigarrow} ((2,2),(1,2)) \circ [m_1, \emptyset] \rhd 0 \mid \overline{a}.c$$
$$t_3 : \mathsf{s} \circ [m_1, m_2] \rhd 0 \mid c \overset{0:a}{\rightsquigarrow} ((0,2),(3,2)) \circ [\emptyset, m_2] \rhd a + b \mid c$$

Among them, t_2 and t_3 are concurrent, as they are both backward, as well as t_1 and t_3, as 3 was not generated by $\mathsf{ip}_{t_3} = (0,2)$. However, as 3 is downstream of $\mathsf{ip}_{t_2} = (1,2)$, t_1 and t_2 are *not* concurrent.

Causal Consistency We now prove that our framework enjoys causal consistency, a property stating that an action can be reversed only provided all its consequences have been undone. Causal consistency holds for a calculus which satisfies four basic axioms [28]: *Loop Lemma*—"any reduction can be undone"—, *Square Property*—"concurrent transitions can be executed in any order"—, *Concurrency (independence) of the backward transitions*—"coinitial backward transitions are concurrent"— and *Well-foundedness*—"each process has a finite past". Additionally, it is assumed that the semantics is equipped with the independence relation, in our case concurrency relation.

Lemma 3 (Axioms). *For every reachable processes* R, R'*, IRLTS satisfies the following axioms:*

Loop Lemma: *for every forward transition* $t : R \xrightarrow{i:\alpha} R'$ *there exists a backward transition* $t^\bullet : R' \overset{i:\alpha}{\rightsquigarrow} R$ *and vice versa.*

Square Property: *if* $t_1 : R \xrightarrow{i_1:\alpha_1}\!\!\!\!\rightarrow R_1$ *and* $t_2 : R \xrightarrow{i_2:\alpha_2}\!\!\!\!\rightarrow R_2$ *are two coinitial concurrent transitions, there exist two cofinal transitions* $t_2' : R_1 \xrightarrow{i_2:\alpha_2}\!\!\!\!\rightarrow R_3$ *and* $t_1' : R_2 \xrightarrow{i_1:\alpha_1}\!\!\!\!\rightarrow R_3$*.*

Backward transitions are concurrent: *any two coinitial backward transitions* $t_1 : R \overset{i_1:\alpha_1}{\rightsquigarrow} R_1$ *and* $t_2 : R \overset{i_2:\alpha_2}{\rightsquigarrow} R_2$ *where* $t_1 \neq t_2$ *are concurrent.*

Well-foundedness: *there is no infinite backward computation.*

We now define the "causal equivalence" [15] relation on traces allowing to swap concurrent transitions and to delete transitions triggered in both directions. The causal equivalence relation is defined for the LTSI which satisfies the Square Property and re-use the notations from above.

Definition 21 (Causal equivalence). Causal equivalence, $\sim$, *is the least equivalence relation on traces closed under composition satisfying* $t_1; t_2' \sim t_2; t_1'$ *and* $t; t^\bullet \sim \epsilon$— ϵ *being the empty trace.*

Now, given the notion of causal equivalence, using an axiomatic approach [28] and that our reversible semantics satisfies necessary axioms, we obtain that our framework satisfies causal consistency, given bellow.

Theorem 1 (Causal consistency). *In IRLTS, two traces are coinitial and cofinal iff they are causally equivalent.*

Finally, we give the equivalent to the "unicity lemma" (Lemma 2) for IRLTS: note that since the same transition can occur multiple times, and as backward and forward transitions may share the same identifiers, we can have the exact same guarantee that any transition uses identifiers only once only up to causal consistency.

Lemma 4 (Unicity for IRLTS). *For a given trace* d*, there exist a trace* d'*, such that* $d' \sim d$ *and* d' *contains any identifier at most once, and if a transition in* d' *has identifier* $i_1 \oplus i_2 \in \mathsf{I}_p$*, then neither* i_1 *nor* i_2 *occur in* d'*.*

3.3 Links to RCCS and CCSK: Translations and Comparisons

It is possible to work out an encoding of our IRLTS terms into RCCS and CCSK terms [8]. Our calculus is more general, since it allows multiple sums, and more precise, since the identifier mechanism is explicit, but has some drawbacks with respect to those calculi as well.

While RCCS "maximally distributes" the memories to all the threads, our calculus for the time being forces all the memories to be stored in one shared place. Poor implementations of this mechanism could result in important bottlenecks, as memories need to be centralized: however, we believe that an asynchronous handling of the memory accesses could allow to bypass this limitation in our calculus, but reserve this question for future work. With respect to CCSK, our memory events are potentially duplicated every time the $\delta^?$ operator is applied, resulting in a space waste, while CCSK never duplicates any memory event. Furthermore, the stability of CCSK's terms through execution—as the number of threads does not change during the computation—could be another advantage.

We believe the encoding we present to be fairly straightforward, and that it will open up the possibility of switching from one calculus to another based on the needs to distribute the memories or to reduce the memory footprint.

4 Contexts, and How We Do Not Have Congruences Yet

We remind the reader of the definition of contexts $C[\cdot]$ on CCS terms P, before introducing contexts $C^{\mathsf{I}}[\cdot]$ (resp. $M[\cdot]$, $C^{\mathsf{R}}[\cdot]$) on identified terms I (resp. on memories M, on identified reversible terms R).

Definition 22 (Term Context). *A context $C[\cdot] : \mathsf{P} \to \mathsf{P}$ is inductively defined using all process operators and a fresh symbol $\cdot$ (the* slot*) as follows (omitting the symmetric contexts):*

$$C[\cdot] \coloneqq \lambda.C[\cdot] \mid P \mid C[\cdot] \mid C[\cdot]\backslash\lambda \mid \lambda_1.P + \lambda_2.C[\cdot] \mid P \between C[\cdot] \mid P \sqcap C[\cdot] \mid \cdot$$

When placing an identified term into a context, we want to make sure that a well-identified process remains well-identified, something that can be easily achieved by noting that for all process P and seed s, $(\cup^? \cap^? \mathsf{s}) \circ P$ is always well-identified, for the following definition of $\cup^?$:

Definition 23 (Unifier). *Given a process P and a seed s, we define*

$$\cup^?(\mathsf{ip}, P) = \mathsf{ip} \circ P$$
$$\cup^?((\mathsf{s}_1, \mathsf{s}_2), P) = \begin{cases} (\cup^?(\cap_1(\mathsf{s}_1), P)) & \text{if } \mathsf{s}_1 \text{ is not of the form } \mathsf{ip}_1 \\ (\cap_1(\mathsf{s}_1), P) & \text{otherwise} \end{cases}$$

Definition 24 (Identified Context). *An identified context $C^{\mathsf{I}}[\cdot] : \mathsf{I} \to \mathsf{I}$ is defined using term contexts as $C^{\mathsf{I}}[\cdot] = (\cup^? \cap^? \cdot) \circ C[\cdot]$.*

Example 7. A term $(0,1) \circ a + b$, in the identified context $(\cup^? \cap^? \cdot) \circ \cdot \mid \overline{a}$, gives the term $((0,2),(1,2)) \circ a + b \mid \overline{a}$ from Example 5. The term $((0,2),(1,2)) \circ a \mid b$ placed in the same context would give $((0,4),(1,4)),(2,4)) \circ (a \mid b) \mid \overline{a}$.

To study *memory contexts*, we write M for the set of all memories.

Definition 25 (Memory Context). *A memory context $M[\cdot] : \mathsf{M} \to \mathsf{M}$ is inductively defined using the operators and operations of Definitions 13, 14 and 15, an "append" operation and a fresh symbol $\cdot$ (the* slot*) as follows:*

$$M[\cdot] \coloneqq [M[\cdot], m] \mid [m, M[\cdot]] \mid e.M[\cdot] \mid M[\cdot].e \mid \delta^? M[\cdot] \mid M[\cdot][j \leftarrow k] \mid M[\cdot] +\!\!+_j (o, P, d) \mid \cdot$$

Where $e.m = [e.m_1, e.m_2]$ and $m.e = [m_1.e, m_2.e]$ if $m = [m_1, m_2]$, and $m.e = m'.e.\emptyset$ if $m = m'.\emptyset$.

Definition 26 (Reversible Context). *A reversible context $C^{\mathsf{R}}[\cdot] : \mathsf{R} \to \mathsf{R}$ is defined using term and memory contexts as $C^{\mathsf{R}}[\cdot] = (\cup^? \cap^? \cdot) \circ M[\cdot] \rhd C[\cdot]$. It is* memory neutral *if $M[\cdot]$ is built using only $\cdot$, $[\emptyset, M[\cdot]]$ and $[M[\cdot], \emptyset]$.*

Of course, a reversible context can change the past of a reversible process R, and hence the initial process O_R to which it corresponds (Definition 17).

Example 8. Let $C^{\mathsf{R}}[\cdot]_1 = [\emptyset, \cdot] \rhd P \mid C[\cdot]$ and $C^{\mathsf{R}}[\cdot]_2 = \delta^?[\cdot] \rhd P \mid C[\cdot]$. Letting $R = (1,1) \circ \langle 0, a, \_\rangle \rhd b$, we obtain $C^{\mathsf{R}}[R]_1 = ((1,2),(2,2)) \circ [\emptyset, \langle 0, a, \_\rangle] \rhd P \mid b$ and $C^{\mathsf{R}}[R]_2 = ((1,2),(2,2)) \circ [\langle 0, a, \_\rangle, \langle 0, a, \_\rangle] \rhd P \mid b$, and we have

$$C^{\mathsf{R}}[R]_1 \overset{0:a}{\rightsquigarrow} ((1,2),(0,2)) \circ [\emptyset, \emptyset] \rhd P \mid a.b \qquad C^{\mathsf{R}}[R]_2 \overset{0:a}{\rightsquigarrow} (0,1) \circ \emptyset \rhd a.(P \mid b)$$

Note that not all of the reversible contexts, when instantiated with a reversible term, will give accessible terms. Typically, the context $[\emptyset, \cdot] \rhd \cdot$ will be "broken" since the memory pair created will never coincide with the structure of the term and its memory inserted in those slots. However, even restricted to contexts producing accessible terms, reversible contexts are strictly more expressive that term contexts. To make this more precise in Lemma 5, we use two bisimulations close in spirit to Forward-reverse bisimulation [39] and back-and-forth bisimulation [10], but that leave some flexibility regarding identifiers and corresponds to Hereditary-History Preserving Bisimulations [6]. Those bisimulations—B&F and SB&F [6,8]—are *not* congruences, not even under "memory neutral" contexts.

Lemma 5. *For all non-initial reversible process R, there exists reversible contexts $C^{\mathsf{R}}[\cdot]$ such $O_{C^{\mathsf{R}}[R]}$ is reachable and for all term context $C[\cdot]$, $C[O_R]$ and $O_{C^{\mathsf{R}}[R]}$ are not* B&F.

Theorem 2. B&F *and* SB&F *are not congruences, not even under memory neutral contexts.*

Proof. The processes $R_1 = (1,1) \circ \langle 0, a, \_ \rangle \rhd b + b$ and $R_2 = (1,1) \circ \langle 0, a, (+, a.b, \mathrm{R}) \rangle \rhd b$ are B&F, but letting $C^{\mathsf{R}}[\cdot] = \cdot \rhd \cdot + c$, $C^{\mathsf{R}}[R_1]$ and $C^{\mathsf{R}}[R_2]$ are not. Indeed, it is easy to check that R_1 and R_2, as well as $O_{R_1} = (0,1) \circ \emptyset \rhd a.(b+b)$ and $O_{R_2} = (0,1) \circ \emptyset \rhd (a.b) + (a.b)$, are B&F, but $O_{C^{\mathsf{R}}[R_1]} = (0,1) \circ \emptyset \rhd a.((b+b)+c)$ and $O_{C^{\mathsf{R}}[R_2]} = (0,1) \circ \emptyset \rhd (a.(b+c)) + (a.b)$ are not B&F, and hence $C^{\mathsf{R}}[R_1]$ and $C^{\mathsf{R}}[R_2]$ cannot be either. The same example works for SB&F.

We believe similar reasoning and example can help realizing that *none of the bisimulations introduced for reversible calculi are congruences* under our definition of reversible context. Some congruences for reversible calculi have been studied [5], but they allowed the context to be applied only to the origins of the reversible terms: whenever interesting congruences allowing contexts to be applied to non-initial terms exist is still an open problem, in our opinion, but we believe our formal frame will allow to study it more precisely.

5 Conclusion

We like to think of our contribution as a first sketch enabling researchers to tackle much more ambitious problems. It is our hope that our identified calculus can at the same time help sidestepping some of the implementation issues for reversible protocols [12], and can be re-used for RCCS or CCSK as a convenient base, or plug-in, to obtain distributed and reliable keys or identifiers. We also hope that the probabilistic choice [17]—whose representation requires to either develop an auxiliary relation [17, p. 67], to make the transition system become probabilistic as well [9], or to use Segala automata [44]—will be within the realm of reversible protocols, as its implications and applications could be numerous. The interleaving of the sums—for instance in the mixed choice [21], that offers both probabilistic choice and nondeterministic choice—could then possibly be

unlocked and provides opportunities to model and study more complex behavior without leaving the reversible frame.

It is known that CCS is not "universally expressive" [19,20], and we would like to assess how universal the protocol detailed in this paper is. To that aim, careful study of reversible and heterogeneous computing devices will be required, that in turns could shed a new light on some of the questions we left unanswered. Typically, this could lead to the development of "location-aware" calculi, where the distribution of seeds and memory is made explicit, or to make progress in the definition of "the right" structural congruence [7]. Last but not least, interesting declensions of contexts were left out in this study, taking for instance a reversible context $\cdot \rhd P$ that "throws away" the term under study but "steals" its memory.

Acknowledgments The authors wish to express their gratitude to Ioana Cristescu for asking some of the questions we tried to answer in this paper, to Assya Sellak for suggesting to use (something close to) caktus stacks to represent our memories, and to the reviewers for their interesting observations.

References

1. Abadi, M., Blanchet, B., Fournet, C.: The applied pi calculus: Mobile values, new names, and secure communication. J. ACM **65**(1), 1:1–1:41 (2018). https://doi.org/10.1145/3127586
2. Amadio, R.M.: Operational methods in semantics. Lecture notes, Université Denis Diderot Paris 7 (Dec 2016), https://hal.archives-ouvertes.fr/cel-01422101
3. Arpit, Kumar, D.: Calculus of concurrent probabilistic reversible processes. In: ICCCT. p. 34–40. ICCCT-2017, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3154979.3155004
4. Aubert, C., Cristescu, I.: Reversible barbed congruence on configuration structures. In: ICE 2015. EPTCS, vol. 189, pp. 68–95 (2015). https://doi.org/10.4204/EPTCS.189.7
5. Aubert, C., Cristescu, I.: Contextual equivalences in configuration structures and reversibility. J. Log. Algebr. Methods Program. **86**(1), 77–106 (2017). https://doi.org/10.1016/j.jlamp.2016.08.004
6. Aubert, C., Cristescu, I.: How reversibility can solve traditional questions: The example of hereditary history-preserving bisimulation. In: CONCUR. LIPIcs, vol. 2017, pp. 13:1–13:24. Schloss Dagstuhl (2020). https://doi.org/10.4230/LIPIcs.CONCUR.2020.13
7. Aubert, C., Cristescu, I.: Structural equivalences for reversible calculi of communicating systems (oral communication). Research report, Augusta University (2020), https://hal.archives-ouvertes.fr/hal-02571597, communication at ICE 2020
8. Aubert, C., Medić, D.: Enabling Replications and Contexts in Reversible Concurrent Calculus (Extended Version) (May 2021), https://hal.archives-ouvertes.fr/hal-03183053
9. Baier, C., Kwiatkowska, M.Z.: Domain equations for probabilistic processes. MSCS **10**(6), 665–717 (2000). https://doi.org/10.1017/S0960129599002984
10. Bednarczyk, M.A.: 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), http://www.ipipan.gda.pl/~marek/papers/historie.ps.gz

11. Boudol, G., Castellani, I.: Permutation of transitions: An event structure semantics for CCS and SCCS. In: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30 - June 3, 1988, Proceedings. LNCS, vol. 354, pp. 411–427. Springer (1988). https://doi.org/10.1007/BFb0013028
12. Cox, G.: SimCCSK: simulation of the reversible process calculi CCSK. Master's thesis, University of Leicester (4 2010), https://leicester.figshare.com/articles/thesis/SimCCSK_simulation_of_the_reversible_process_calculi_CCSK/10091681
13. Cristescu, I., Krivine, J., Varacca, D.: A compositional semantics for the reversible p-calculus. In: LICS. pp. 388–397. IEEE Computer Society (2013). https://doi.org/10.1109/LICS.2013.45
14. Cristescu, I., Krivine, J., Varacca, D.: Rigid families for CCS and the π-calculus. In: Theoretical Aspects of Computing - ICTAC 2015. LNCS, vol. 9399, pp. 223–240. Springer (2015). https://doi.org/10.1007/978-3-319-25150-9_14
15. Danos, V., Krivine, J.: Reversible communicating systems. In: CONCUR. LNCS, vol. 3170, pp. 292–307. Springer (2004). https://doi.org/10.1007/978-3-540-28644-8_19
16. Danos, V., Krivine, J.: Transactions in RCCS. In: CONCUR. LNCS, vol. 3653, pp. 398–412. Springer (2005). https://doi.org/10.1007/11539452_31
17. Fischer, N., van Glabbeek, R.J.: Axiomatising infinitary probabilistic weak bisimilarity of finite-state behaviours. J. Log. Algebr. Methods Program. **102**, 64–102 (2019). https://doi.org/10.1016/j.jlamp.2018.09.006
18. Frank, M.P., Brocato, R.W., Tierney, B.D., Missert, N.A., Hsia, A.H.: Reversible computing with fast, fully static, fully adiabatic CMOS. In: ICRC, Atlanta, GA, USA, December 1-3, 2020. pp. 1–8. IEEE (2020). https://doi.org/10.1109/ICRC2020.2020.00014
19. van Glabbeek, R.J.: On specifying timeouts. Electron. Notes Theor. Comput. Sci. **162**, 173–175 (2006). https://doi.org/10.1016/j.entcs.2005.12.083
20. van Glabbeek, R.J., Höfner, P.: CCS: it's not fair! - fair schedulers cannot be implemented in ccs-like languages even under progress and certain fairness assumptions. Acta Inform. **52**(2-3), 175–205 (2015). https://doi.org/10.1007/s00236-015-0221-6
21. Goubault-Larrecq, J.: Isomorphism theorems between models of mixed choice. MSCS **27**(6), 1032–1067 (2017). https://doi.org/10.1017/S0960129515000547
22. Graversen, E., Phillips, I., Yoshida, N.: Event structure semantics of (controlled) reversible CCS. In: RC 2018, Leicester, UK, September 12-14, 2018, Proceedings. LNCS, vol. 11106, pp. 102–122. Springer (2018). https://doi.org/10.1007/978-3-319-99498-7_7
23. Hennessy, M.: A distributed Pi-calculus. CUP (2007). https://doi.org/10.1017/CBO9780511611063
24. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
25. Krivine, J.: Algèbres de Processus Réversible - Programmation Concurrente Déclarative. Ph.D. thesis, Université Paris 6 & INRIA Rocquencourt (2006), https://tel.archives-ouvertes.fr/tel-00519528
26. Lanese, I., Lienhardt, M., Mezzina, C.A., Schmitt, A., Stefani, J.: Concurrent flexible reversibility. In: ESOP. LNCS, vol. 7792, pp. 370–390. Springer (2013). https://doi.org/10.1007/978-3-642-37036-6_21
27. Lanese, I., Medić, D., Mezzina, C.A.: Static versus dynamic reversibility in CCS. Acta Inform. (Nov 2019). https://doi.org/10.1007/s00236-019-00346-6

28. Lanese, I., Phillips, I.C.C., Ulidowski, I.: An axiomatic approach to reversible computation. In: FOSSACS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings. LNCS, vol. 12077, pp. 442–461. Springer (2020). https://doi.org/10.1007/978-3-030-45231-5_23
29. Lévy, J.J.: Réductions correctes et optimales dans le lambda-calcul. Ph.D. thesis, Paris 7 (Jan 1978), http://pauillac.inria.fr/~levy/pubs/78phd.pdf
30. Matthews, D.: How to get started in quantum computing. Nature **591**(7848), 166–167 (Mar 2021). https://doi.org/10.1038/d41586-021-00533-x
31. Medić, D., Mezzina, C.A.: Static VS dynamic reversibility in CCS. In: RC 2016. LNCS, vol. 9720, pp. 36–51. Springer (2016). https://doi.org/10.1007/978-3-319-40578-0_3
32. Medić, D., Mezzina, C.A., Phillips, I., Yoshida, N.: A parametric framework for reversible π-calculi. Inf. Comput. **275**, 104644 (2020). https://doi.org/10.1016/j.ic.2020.104644
33. Merro, M., Zappa Nardelli, F.: Behavioral theory for mobile ambients. J. ACM **52**(6), 961–1023 (2005). https://doi.org/10.1145/1101821.1101825
34. Mezzina, C.A., Koutavas, V.: A safety and liveness theory for total reversibility. In: TASE 2017, Sophia Antipolis, France, September 13-15. pp. 1–8. IEEE (2017). https://doi.org/10.1109/TASE.2017.8285635
35. Milner, R.: A Calculus of Communicating Systems. LNCS, Springer-Verlag (1980). https://doi.org/10.1007/3-540-10235-3
36. Palamidessi, C., Valencia, F.D.: Recursion vs replication in process calculi: Expressiveness. Bull. EATCS **87**, 105–125 (2005), http://eatcs.org/images/bulletin/beatcs87.pdf
37. Perdrix, S., Jorrand, P.: Classically-controlled quantum computation. Electron. Notes Theor. Comput. Sci. **135**(3), 119–128 (2006). https://doi.org/10.1016/j.entcs.2005.09.026
38. Phillips, I., Ulidowski, I.: Reversing algebraic process calculi. In: FoSSaCS. LNCS, vol. 3921, pp. 246–260. Springer (2006). https://doi.org/10.1007/11690634_17
39. Phillips, I., Ulidowski, I.: Reversibility and models for concurrency. Electron. Notes Theor. Comput. Sci. **192**(1), 93–108 (2007). https://doi.org/10.1016/j.entcs.2007.08.018
40. Rosenberg, A.L.: Efficient pairing functions - and why you should care. Int. J. Found. Comput. Sci. **14**(1), 3–17 (2003). https://doi.org/10.1142/S012905410300156X
41. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. CUP (2011)
42. Sangiorgi, D., Walker, D.: The Pi-calculus. CUP (2001)
43. Szudzik, M.P.: The rosenberg-strong pairing function. CoRR **abs/1706.04129** (2017)
44. de Visme, M.: Event structures for mixed choice. In: CONCUR. LIPIcs, vol. 140, pp. 11:1–11:16. Schloss Dagstuhl (2019). https://doi.org/10.4230/LIPIcs.CONCUR.2019.11, http://www.dagstuhl.de/dagpub/978-3-95977-121-4