A Modular Formalization of Reversibility for Concurrent Models and Languages

Alexis Bernadet, Ivan Lanese

A Modular Formalization of Reversibility for Concurrent Models and Languages*

Alexis Bernadet
Dalhousie University, Canada
bernadet@lix.polytechnique.fr

Ivan Lanese
Focus Team, University of Bologna/INRIA, Italy
ivan.lanese@gmail.com

Causal consistency is the reference notion of reversibility for concurrency. We introduce a modular framework for defining causal-consistent reversible extensions of concurrent models and languages. We show how our framework can be used to define reversible extensions of formalisms as different as CCS and concurrent X-machines. The generality of the approach allows for the reuse of theories and techniques in different settings.

1 Introduction

Reversibility in computer science refers to the possibility of executing a program both in the standard forward direction, and backward, going back to past states. Reversibility appears in many settings, from the undo button present in most text editors, to algorithms for rollback-recovery [3]. Reversibility is also used in state-space exploration, as in Prolog, or for debugging [1]. Reversibility emerges naturally when modeling biological systems [6], where many phenomena are naturally reversible, and in quantum computing [2], since most quantum operations are reversible. Finally, reversibility can be used to build circuits which are more energy efficient than non-reversible ones [12].

Reversibility for concurrent systems has been tackled first in [8], mainly with biological motivations. The standard definition of reversibility in a sequential setting, recursively undo the last action, is not applicable in concurrent settings, where there are many actions executing at the same time. Indeed, a main contribution of [8] has been the definition of *causal-consistent* reversibility: any action can be undone provided that all the actions depending on it (if any) have already been undone. This definition can be applied to concurrent systems, and it is now a reference notion in the field (non causal reversibility is also studied, e.g., in [20]). See [15] for a survey on causal-consistent reversibility.

Following [8], causal-consistent reversible extensions of many concurrent models and languages have been defined, using different techniques [13, 7, 16, 19, 11, 14]. Nevertheless, the problem of finding a general procedure that given a formalism defines its causal-consistent reversible extension is still open: we tackle it here (we compare in Section 6 with other approaches to the same problem). In more details, we present a modular approach, where the problem is decomposed in three main steps. The first step defines the information to be saved to enable reversibility (in a sequential setting). The second step concerns the choice of the concurrency model used. The last step automatically builds a causal-consistent reversible extension of the given formalism with the chosen concurrency model.

Our approach is not aimed at providing efficient (in terms of amount of history information stored, or in terms of time needed to recover a past state) reversible causal-consistent semantics, but at providing guidelines to develop reversible causal-consistent semantics which are correct by construction. Indeed,

*This work was partially supported by the Italian MIUR project PRIN 2010-2011 CINA, the French ANR project REVER n. ANR 11 INSE 007 and the European COST Action IC1405.

Submitted to:
ICE 2016

the relevant properties expected from a causal-consistent reversible semantics will hold naturally because of our construction. Also, it clarifies the design space of causal-consistent reversibility, by clearly separating the sequential part (step 1) from the part related to concurrency (step 2).

Hence, our approach can be used:

- in models and languages where one or more causal-consistent reversible semantics already exist (such as CCS, see Section 4), to provide a reference model correct by contruction, to compare against the existing ones and to classify them according to the choices needed in steps 1 and 2 to match them;
- in models and languages where no causal-consistent reversible semantics currently exists (such as X-machines, see Section 5), to provide an idea on how such a semantics should look like, and which are the challenges to enable causal-consistent reversibility in the given setting.

Section 2 gives an informal overview of our approach. Section 3 gives a formal presentation of the construction of a reversible LTS extending a given one and proves that the resulting LTS satisfies the properties expected from a causal-consistent reversible formalism. In Section 4, we apply our approach to CCS. In Section 5 we show how to apply the same approach to systems built around concurrent X-machines. Section 6 compares with related approaches and presents directions for future work. For reviewers' convenience, proofs missing from the main part are collected in the Appendix.

2 Informal Presentation

We want to define a causal-consistent reversible extension for a given formal model or language. Assume that the model or the language is formally specified by a calculus with terms M whose behavior is described by an LTS with transitions of the form:

$$M \xrightarrow{u} M'$$

To define its causal-consistent reversible extension using our approach we need it to satisfy the following properties:

- The LTS is deterministic: If $M \xrightarrow{u} M_1$ and $M \xrightarrow{u} M_2$, then $M_1 = M_2$.
- The LTS is co-deterministic: If $M_1 \xrightarrow{u} M'$ and $M_2 \xrightarrow{u} M'$, then $M_1 = M_2$.

In other words, the label u should contain enough information on how to go forward and backward. This is clearly too demanding, hence the following question is natural:

What to do if the LTS is not deterministic or not co-deterministic ?

Note that this is usually the case. For example, in CCS [18], a label is either τ, a or $\bar{a}$, and we can have, e.g., $P \xrightarrow{\tau} P_1$ and $P \xrightarrow{\tau} P_2$ with $P_1 \neq P_2$.

What we can do is to *refine* the labels and, as a consequence, the calculus, by adding information to them. Therefore, if we have an LTS with terms M and labels α which is not deterministic or not co-deterministic, we have to define:

- A new set of labels ranged over by u.
- A new LTS with the same terms M and with labels u which is deterministic and co-deterministic.
- An interpretation $[\![u]\!] = \alpha$ for each label u such that:
  $M \xrightarrow{u} M'$ iff $M \xrightarrow{[\![u]\!]} M'$ (correctness of the refinement).

Remark 1 (A Naive Way of Refining Labels)**.**
A simple way of refining labels α is as follows:

- *Labels u are of the form (M, α, M') for each M, α and M' with $M \xrightarrow{\alpha} M'$.*

- *We have only transitions of the form* $M_1 \xrightarrow{(M_1,\alpha,M_2)} M_2$*. This LTS is trivially deterministic and co-deterministic.*
- *We define* $[\![(M,\alpha,M')]\!] = \alpha$*. The correctness of this refinement is trivial.*

Therefore, it is always possible to refine an LTS to ensure determinism and co-determinism. Unfortunately, as we will see later, this way of refining is not suitable for our aims, since we want some transitions, notably concurrent ones, to commute without changing their labels, and this is not possible with the refinement above.

Assume now that we have an LTS which is deterministic and co-deterministic with terms M and labels u, and a computation:

$$M_0 \xrightarrow{u_1} M_1 \dots \xrightarrow{u_n} M_n$$

From co-determinism, if we only have the labels $u_1, \dots u_n$ and the last term M_n, we can retrieve the initial term M_0 and all the intermediate terms $M_1 \dots M_{n-1}$. As a consequence, we can use the following notation without losing information:

$$M_0 \xrightarrow{u_1} \dots \xrightarrow{u_n} M_n$$

Therefore, only the labels are needed to describe the history of a particular execution that led to a given term.

Hence, to introduce reversibility it is natural to define configurations and transitions as follows:

- Configurations R are of the form (L,M) with L a sequence of labels $u_1, \dots u_n$ such that there exists M' such that $M' \xrightarrow{u_1} \dots \xrightarrow{u_n} M$ (we can notice that M' is unique).
- Forward transitions: If $M \xrightarrow{u} M'$, then $(L,M) \xrightarrow{u} ((L,u),M')$.
- Backward transitions: If $M \xrightarrow{u} M'$, then $((L,u),M') \xrightarrow{u^{-1}} (L,M)$.

In the LTS above, the terms are configurations R, and the labels are either of the form u (move forward) or u^{-1} (move backward). This new formalism is indeed reversible. This can be proved by showing that the Loop lemma [8, Lemma 6], requiring each step to have an inverse, holds. The main limitation of this way of introducing reversibility is that a configuration can only do the backward step that cancels the last forward step: If $R_1 \xrightarrow{u} R_2$ and $R_2 \xrightarrow{v^{-1}} R_3$, then $u = v$ and $R_1 = R_3$. This form of reversibility is suitable for a sequential setting, where actions are undone in reverse order of completion. In a concurrent setting, as already discussed in the Introduction, the suitable notion of reversibility is causal-consistent reversibility, where any action can be undone provided that all the actions depending on it (if any) have already been undone. We show now how to generalize our model so to obtain causal-consistent reversibility.

First, we require a symmetric relation $\perp$ on labels u. Intuitively, $u \perp v$ means that the actions described by u and v are independent and can be executed in any order. In concurrent systems, a sensible choice for $\perp$ is to have $u \perp v$ if and only if the corresponding transitions are concurrent. By choosing instead $u \perp v$ if and only if $u = v$ we recover the sequential setting. Indeed, causal-consistent reversibility coincides with sequential reversibility if no actions are concurrent.

The only property on $\perp$ that we require, besides being symmetric, is the following one:

If $M_1 \xrightarrow{u} M_2$, $M_2 \xrightarrow{v} M_3$ and $u \perp v$, then there exists M_2' such that $M_1 \xrightarrow{v} M_2'$ and $M_2' \xrightarrow{u} M_3$. (*co-diamond property*).

Thanks to this property, we can define an equivalence relation $\asymp$ on sequences L of labels: $L \asymp L'$ if and only if L' can be obtained by a sequence of permutations of consecutive u and v in L such that $u \perp v$.

We can now generalize the definition of configuration $R = (L,M)$ by replacing L by its equivalence class $[L]$ w.r.t. $\asymp$. In other words, a configuration R is now of the form $([L],M)$. Actually, $[L]$ is a Mazurkiewicz trace [17]. Transitions are generalized accordingly.

For example, if $u \perp v$ we can have transition sequences such as:

$$([L], M_1) \xrightarrow{u} ([(L,u)], M_2) \xrightarrow{v} ([(L,u,v)], M_3) \xrightarrow{u^{-1}} ([(L,v)], M_4)$$

because, since $(L,u,v) \asymp (L,v,u)$, we have $[(L,u,v)] = [(L,v,u)]$.

We will show that the formalism that we obtain in this way is reversible (the Loop lemma still holds) in a causal-consistent way.

Therefore, the work of defining a causal-consistent reversible extension of a given LTS can be split into the three steps below.

1. Refine the labels of the transitions.
2. Define a suitable relation $\perp$ on the newly defined labels.
3. Define the configurations R and the forward and backward transitions with the construction given above.

We can notice that step 1 depends on the semantics of the chosen formalism and step 2 depends on the chosen concurrency model. These two steps are not automatic. Step 3 instead is a construction that does not depend on the chosen formalism and which is completely mechanical. This modular approach has the advantage of allowing the reuse of theories and techniques, in particular the ones referred to step 3. Also, it allows one to better compare different approaches, by clearly separating the choices related to the concurrency model (step 2) from the ones related to the (sequential) semantics of the formalism (step 1).

Step 1 is tricky: we have to be careful when refining the labels. We must add enough information to labels so that the LTS becomes deterministic and co-deterministic. However, the labels must also remain unchanged when permuting two independent steps. Therefore, if we want to allow as much permutations as possible, we need to limit the amount of information added to the labels. For example, the refinement given in Remark 1 only allows trivial permutations, and in this case $\asymp$ can only be the identity.

3 Introducing reversibility, formally

To apply the construction informally described in Section 2, we need a formalism expressed as an LTS that satisfies the properties of Theory 1.

Theory 1. *We have the following objects:*

- *A set $\mathfrak{D}$ of labels with a symmetric relation $\perp$ on $\mathfrak{D}$.*
- *An LTS with terms M and transitions $\xrightarrow{u}$ with labels $u \in \mathfrak{D}$.*

The objects above satisfy the following properties:

- *Determinism: If $M \xrightarrow{u} M_1$ and $M \xrightarrow{u} M_2$, then $M_1 = M_2$.*
- *Co-determinism: If $M_1 \xrightarrow{u} M'$ and $M_2 \xrightarrow{u} M'$, then $M_1 = M_2$.*
- *Co-diamond property: If $M_1 \xrightarrow{u} M_2$, $M_2 \xrightarrow{v} M_3$ and $u \perp v$, then there exists M_2' such that $M_1 \xrightarrow{v} M_2'$ and $M_2' \xrightarrow{u} M_3$.*

In the rest of this section, we assume to have the objects and properties of Theory 1. For the formal definition of configurations we use Mazurkiewicz traces [17] (see Remark 2, later on). We give here a self-contained construction.

If we have the following sequence of transitions:

$$M_0 \xrightarrow{u_1} M_1 \xrightarrow{u_2} \dots \xrightarrow{u_{n-1}} M_{n-1} \xrightarrow{u_n} M_n$$

then the initial term M_0 and all the intermediate terms $M_1, \dots M_{n-1}$ can be retrieved from $u_1, \dots u_n$ and M_n. Therefore, by writing:

$$M_0 \xrightarrow{u_1}\xrightarrow{u_2} \dots \xrightarrow{u_{n-1}}\xrightarrow{u_n} M_n$$

we do not lose any information.

We would like to manipulate formally transition sequences as mathematical objects. Hence it makes sense to use the following notation:

- A sequence $L = u_1, \dots u_n$ of elements in $\mathfrak{D}$ is written $L = \xrightarrow{u_1} \dots \xrightarrow{u_n}$.
- The concatenation of L_1 and L_2 is written L_1L_2 and the empty sequence is written ε. $|L|$ denotes the length of sequence L.

Moreover, we want to consider sequences of transitions up to permutations of independent steps.

Definition 1. *The judgment $L \asymp L'$ is defined by the following rules:*

$$\frac{}{L \asymp L} \qquad \frac{L \asymp (L_1 \xrightarrow{u}\xrightarrow{v} L_2) \quad u \bot v}{L \asymp (L_1 \xrightarrow{v}\xrightarrow{u} L_2)}$$

In other words, $L \asymp L'$ iff L' can be obtained by doing permutations of consecutive independent labels. We can check that $\asymp$ satisfies the following properties:

Lemma 1 (Properties of $\asymp$)**.**

1. *$\asymp$ is an equivalence relation.*
2. *If $L_1 \asymp L_1'$ and $L_2 \asymp L_2'$, then $(L_1L_2) \asymp (L_1'L_2')$.*
3. *If $(L_1 \xrightarrow{u}) \asymp L_2$, then there exist L_3 and L_4 such that $L_2 = (L_3 \xrightarrow{u} L_4)$, $L_1 \asymp (L_3L_4)$ and for all v in L_4, $v \neq u$ and $u \bot v$.*
4. *If $(L_1 \xrightarrow{u}) \asymp (L_2 \xrightarrow{u})$, then $L_1 \asymp L_2$.*
5. *If for all $v \in L$, $u \bot v$, then $(L \xrightarrow{u}) \asymp (\xrightarrow{u} L)$.*
6. *If $L_1 \asymp L_2$, then $|L_1| = |L_2|$.*

We now define formally the judgment MLM', representing a sequence of transitions (also called a computation) with labels in L starting in M and ending in M', and prove some of its properties in Lemma 2.

Definition 2. *The judgment MLM' is defined by the following rules:*

$$\frac{}{M\varepsilon M} \qquad \frac{MLM' \quad M' \xrightarrow{u} M''}{M(L \xrightarrow{u})M''}$$

Lemma 2 (Properties of MLM')**.**

1. *The notation MLM' is a conservative extension of the notation $M \xrightarrow{u} M'$.*
2. *If ML_1M' and $M'L_2M''$, then $M(L_1L_2)M''$.*
3. *If M_1LM' and M_2LM', then $M_1 = M_2$.*
4. *If $M(L_1L_2)M'$, then there exists M'' such that ML_1M'' and $M''L_2M'$.*
5. *If MLM' and $L \asymp L'$, then $ML'M'$.*

The following lemma will be necessary to prove Theorem 3:

Lemma 3. *If $(L_1L_3) \asymp (L_2L_4)$ and $(L_1L_5) \asymp (L_2L_6)$, then there exist L_7, L_8, L_3', L_4', L_5', L_6' such that $L_3 \asymp (L_7L_3')$, $L_4 \asymp (L_8L_4')$, $L_5 \asymp (L_7L_5')$, $L_6 \asymp (L_8L_6')$, $L_3' \asymp L_4'$ and $L_5' \asymp L_6'$.*

Now, we have all the tools to define formally a new LTS reversible and causal consistent extending the given one.

Definition 3 (Reversible and Causal-Consistent LTS)**.**
A configuration R is a pair $([L], M)$ of a sequence L modulo $\asymp$ and a term M, such that there exists M' such that $M'LM$. We write $[L]M$ for $([L], M)$.

The semantics of configurations is defined by the following rules:

$$\frac{M \xrightarrow{u} M'}{[L]M \xrightarrow{u} [L \xrightarrow{u}]M'} \qquad \frac{M \xrightarrow{u} M'}{[L \xrightarrow{u}]M' \xrightarrow{u^{-1}} [L]M}$$

For a given configuration $R = [L]M$, the unique M' such that $M'LM$ is independent from the choice of L in the equivalence class. We call such M' the initial term *of the configuration R. We also define the projection of a configuration on the last term as $[\![[L]M]\!] = M$.*

Remark 2. *In the definition above, $[L]$ is a Mazurkiewicz Trace [17].*

The above definition is well posed:

Lemma 4. *If R is a configuration and $R \xrightarrow{u} R'$ (resp. $R \xrightarrow{u^{-1}} R'$) then R' is a configuration with the same initial term.*

Proof. Straightforward. □

The calculus defined above is a conservative extension of the original one. Indeed its forward transitions exactly match the transitions of the original calculus:

Theorem 1 (Preservation of the Semantics)**.**

- *If $R_1 \xrightarrow{u} R_2$, then $[\![R_1]\!] \xrightarrow{u} [\![R_2]\!]$.*
- *If $[\![R_1]\!] \xrightarrow{u} M'$, then there exists R_2 such that $[\![R_2]\!] = M'$ and $R_1 \xrightarrow{u} R_2$.*

Proof. By construction. □

We can also show that the calculus is reversible by proving that the Loop Lemma [8, Lemma 6] holds.

Theorem 2 (Loop Lemma)**.** *$R \xrightarrow{u} R'$ iff $R' \xrightarrow{u^{-1}} R$.*

Proof. Trivial. □

We finally need to prove that our formalism is indeed *causal consistent*. A characterization of causal consistency has been presented in [8, Theorem 1]. It requires that two coinitial computations are cofinal iff they are equal up to causal equivalence, where causal equivalence is an equivalence relation on computations equating computations differing only for swaps of concurrent actions and simplifications of inverse actions (see Theorem 4 for a precise formalization).

Before tackling this problem we study when consecutive transitions can be swapped or simplified.

Lemma 5.

1. *If $R_1 \xrightarrow{u} R_2$, $R_2 \xrightarrow{v} R_3$ and $u \bot v$, then there exists R_2' such that $R_1 \xrightarrow{v} R_2'$ and $R_2' \xrightarrow{u} R_3$.*
2. *If $R_1 \xrightarrow{u^{-1}} R_2$, $R_2 \xrightarrow{v^{-1}} R_3$ and $u \bot v$, then there exists R_2' such that $R_1 \xrightarrow{v^{-1}} R_2'$ and $R_2' \xrightarrow{u^{-1}} R_3$.*
3. *If $R_1 \xrightarrow{u} R_2$ and $R_2 \xrightarrow{u^{-1}} R_3$, then $R_1 = R_3$.*

4. *If* $R_1 \xrightarrow{u^{-1}} R_2$ *and* $R_2 \xrightarrow{u} R_3$*, then* $R_1 = R_3$*.*
5. *If* $R_1 \xrightarrow{u} R_2$*,* $R_2 \xrightarrow{v^{-1}} R_3$ *and* $u \neq v$*, then there exists* R_2'*, such that* $R_1 \xrightarrow{v^{-1}} R_2'$*,* $R_2' \xrightarrow{u} R_3$ *and* $u \perp v$*.*

Given the previous result, we can define (Definition 4) a formal way to rearrange (like in the original calculus) and simplify a sequence of transitions. Note that each rule defining the transformation operator $\leq$ (but for reflexivity) is justified by an item of Lemma 5. Since some of these transformations are asymmetric, e.g., the simplification of a step with its inverse, the resulting formal system is not an equivalence relation but a partial pre-order. For example, the sequence $\xrightarrow{u}\xrightarrow{u^{-1}}$ can be transformed into ε but not the other way around. The reason is that if $M \xrightarrow{u}\xrightarrow{u^{-1}} M'$, then $M \varepsilon M'$. However, we may have $M \varepsilon M$ without necessarily having $M \xrightarrow{u}\xrightarrow{u^{-1}} M'$ (in particular, if M cannot perform u).

Definition 4. *We write* $\mathfrak{D}^c$ *for the set of* α *of the form* u *or* u^{-1}*. Also, we define* $(u^{-1})^{-1} = u$*. A sequence* $\mathfrak{L}$ *of elements* $\alpha_1, \dots \alpha_n$ *is written* $\xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_n}$*. Also, we write* $\mathfrak{L}^{-1}$ *for* $\xrightarrow{\alpha_n^{-1}} \dots \xrightarrow{\alpha_1^{-1}}$*.*

The judgment $R\mathfrak{L}R'$ *is defined by the following rules:*

$$\frac{}{R\varepsilon R} \qquad \frac{R_1\mathfrak{L}R_2 \quad R_2 \xrightarrow{\alpha} R_3}{R_1(\mathfrak{L}\xrightarrow{\alpha})R_3}$$

The judgment $\mathfrak{L} \leq \mathfrak{L}'$ *is defined by the following rules:*

$$\frac{}{\mathfrak{L} \leq \mathfrak{L}} \qquad \frac{\mathfrak{L}_1 \leq (\mathfrak{L}_2 \xrightarrow{u}\xrightarrow{v} \mathfrak{L}_3) \quad u \perp v}{\mathfrak{L}_1 \leq (\mathfrak{L}_2 \xrightarrow{v}\xrightarrow{u} \mathfrak{L}_3)} \qquad \frac{\mathfrak{L}_1 \leq (\mathfrak{L}_2 \xrightarrow{u^{-1}}\xrightarrow{v^{-1}} \mathfrak{L}_3) \quad u \perp v}{\mathfrak{L}_1 \leq (\mathfrak{L}_2 \xrightarrow{v^{-1}}\xrightarrow{u^{-1}} \mathfrak{L}_3)}$$

$$\frac{\mathfrak{L}_1 \leq (\mathfrak{L}_2 \xrightarrow{u}\xrightarrow{u^{-1}} \mathfrak{L}_3)}{\mathfrak{L}_1 \leq (\mathfrak{L}_2\mathfrak{L}_3)} \qquad \frac{\mathfrak{L}_1 \leq (\mathfrak{L}_2 \xrightarrow{u^{-1}}\xrightarrow{u} \mathfrak{L}_3)}{\mathfrak{L}_1 \leq (\mathfrak{L}_2\mathfrak{L}_3)} \qquad \frac{\mathfrak{L}_1 \leq (\mathfrak{L}_2 \xrightarrow{u}\xrightarrow{v^{-1}} \mathfrak{L}_3) \quad u \neq v}{\mathfrak{L}_1 \leq (\mathfrak{L}_2 \xrightarrow{v^{-1}}\xrightarrow{u} \mathfrak{L}_3)}$$

We can now prove some properties of the judgments above.

Lemma 6.

1. *The notation* $R\mathfrak{L}R'$ *is a conservative extension of the notation* $R \xrightarrow{\alpha} R'$*.*
2. *If* $R\mathfrak{L}_1R'$ *and* $R'\mathfrak{L}_2R''$*, then* $R(\mathfrak{L}_1\mathfrak{L}_2)R''$*.*
3. *If* $R(\mathfrak{L}_1\mathfrak{L}_2)R'$*, then there exists* R'' *such that* $R\mathfrak{L}_1R''$ *and* $R''\mathfrak{L}_2R'$*.*
4. $\leq$ *is a partial pre-order.*
5. *If* $\mathfrak{L}_1 \leq \mathfrak{L}_2$ *and* $R\mathfrak{L}_1R'$*, then* $R\mathfrak{L}_2R'$*.*
6. *If* $L_1 \asymp L_2$ *then* $L_1 \leq L_2$ *and* $L_1^{-1} \leq L_2^{-1}$*.*

Proof. Follows from Lemma 5 with the structure of proofs similar to the ones found in Lemma 1 and Lemma 2. □

Using the transformations above, we can transform any transition sequence into the form $L_1^{-1}L_2$ where L_1 and L_2 are sequences of forward transitions: $L_1^{-1}L_2$ is composed by a sequence of backward steps followed by a sequence of forward steps (Theorem 3). Intuitively, this means that any configuration can be reached by first going to the beginning of the computation and then going only forward. This result corresponds to the one in [8, Lemma 10], which is sometimes called the Parabolic lemma. In addition, we show that if two computations are coinitial and cofinal, then they have a common form $L_1^{-1}L_2$. As we will see below, this is related to causal consistency [8, Theorem 1].

Theorem 3 (Asymmetrical Causal Consistency). *If $R\mathfrak{L}_1R'$ and $R\mathfrak{L}_2R'$, then there exist L_1 and L_2 such that $\mathfrak{L}_1 \leq L_1^{-1}L_2$ and $\mathfrak{L}_2 \leq L_1^{-1}L_2$.*

Proof. Suppose we have $R = [L]M$ and $R' = [L']M'$. By simplifying the occurrences of $\xrightarrow{u}\xrightarrow{u^{-1}}$ and by replacing occurrences of $\xrightarrow{u}\xrightarrow{v^{-1}}$ by $\xrightarrow{v^{-1}}\xrightarrow{u}$ when $u \neq v$ in $\mathfrak{L}_1$, we can prove that there exist L_1 and L_2 such that $\mathfrak{L}_1 \leq L_1^{-1}L_2$. Therefore, there exist a configuration $R_1 = [L_3]M_1$ such that $RL_1^{-1}R_1$ and R_1L_2R'. Hence, R_1L_1R. So, $L_3L_1 \asymp L$ and $L_3L_2 \asymp L'$. Similarly, we can prove that there exist L_4, L_5 and L_6 such that $\mathfrak{L}_2 \leq L_4^{-1}L_5$, $L_6L_4 \asymp L$ and $L_6L_5 \asymp L'$. Therefore, $L_3L_1 \asymp L_6L_4$ and $L_3L_2 \asymp L_6L_5$. By Lemma 3, there exist L_7, L_8, L'_1, L'_4, L'_2, L'_5 such that $L_1 \asymp L_7L'_1$, $L_4 \asymp L_8L'_4$, $L_2 \asymp L_7L'_2$, $L_5 \asymp L_8L'_5$, $L'_1 \asymp L'_4$ and $L'_2 \asymp L'_5$. Hence, $\mathfrak{L}_1 \leq L_1^{-1}L_2 \leq L_1'^{-1}L_7^{-1}L_7L'_2 \leq L_1'^{-1}L'_2$. Similarly, $\mathfrak{L}_2 \leq L_4'^{-1}L'_5$. We also have $L_1'^{-1}L'_2 \asymp L_4'^{-1}L'_5$. Therefore, $\mathfrak{L}_1 \leq L_1'^{-1}L'_2$ and $\mathfrak{L}_2 \leq L_1'^{-1}L'_2$. □

Remark 3. *Our Theorem 3 could be stated with the terminology in [8] as follows:*

If s_1 and s_2 are coinitial and cofinal computations, then there exists s_3 which is a simplification of both s_1 and s_2.

We will show below that this is stronger than the implication "if two computations are coinitial and cofinal then they are causal equivalent" in the statement of causal consistency [8, Theorem 1]. Moreover, the (easier) implication "if two computations are causal equivalent than they are coinitial and cofinal" of [8, Theorem 1] is also true by construction in our framework.

In order to define causal equivalence we need to restrict to valid reduction sequences. This is needed since otherwise the transformations defining causal equivalence, differently from the ones defining $\leq$, may not preserve validity of reduction sequences. The usual definition of a valid reduction sequence of length n is described by $(n+1)$ configurations $(R_i)_{0\leq i\leq n}$ and n labels $(\alpha_i)_{1\leq i\leq n}$ such that:

$$R_0 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_n} R_n$$

However, by determinism and co-determinism, we can retrieve R_0, $\dots R_{n-1}$ from α_1, $\dots \alpha_n$ and R_n. Therefore, we will use the following equivalent definition:

Definition 5 (Valid Sequences). *A valid sequence s is an ordered pair $(\mathfrak{L},R)$ such that there exists R' with $R'\mathfrak{L}R$. Then, R' is unique, R' and R are called the inital and final configuration of s, and we write $R'sR$. We write E for the set of valid sequences.*

Following [8], we can define *causal equivalence* $\sim$ as follows: if s_2 is a rewriting of s_1 (valid permutation, or valid simplification), then $s_1 \sim s_2$. More formally:

Definition 6 (Causal Equivalence). *Causal equivalence $\sim$ is the least equivalence relation on E closed under composition satisfying the following equivalences (provided both the terms are valid):*

- *we can swap independent actions: if $u \perp v$ then $\xrightarrow{u}\xrightarrow{v} \sim \xrightarrow{v}\xrightarrow{u}$, $\xrightarrow{u^{-1}}\xrightarrow{v^{-1}} \sim \xrightarrow{v^{-1}}\xrightarrow{u^{-1}}$ and $\xrightarrow{u}\xrightarrow{v^{-1}} \sim \xrightarrow{v^{-1}}\xrightarrow{u}$;*
- *we can simplify inverse actions: $\xrightarrow{u}\xrightarrow{u^{-1}} \sim \varepsilon$ and $\xrightarrow{u^{-1}}\xrightarrow{u} \sim \varepsilon$.*

Theorem 4 (Causal Consistency). *Assume that s_1 and s_2 are valid sequences. Then we have $s_1 \sim s_2$ if and only if s_1 and s_2 are coinitial and cofinal.*

Proof.

- If $s_1 \sim s_2$, by construction, each step in the derivation of $s_1 \sim s_2$ does not change the initial and final terms. Therefore, s_1 and s_2 are coinitial and cofinal.

- Assume that s_1 and s_2 are coinitial and cofinal. We want to prove that $s_1 \sim s_2$.
  First, by using Lemma 6 item 5, we can notice that for every $s = (\mathfrak{L}, R) \in E$ and $\mathfrak{L}'$, if $\mathfrak{L} \leq \mathfrak{L}'$, then by writing s' for $(\mathfrak{L}', R)$ we have $s' \in E$ and $s \sim s'$.
  By hypothesis, there exist R, R', $\mathfrak{L}_1$ and $\mathfrak{L}_2$ such that $s_1 = (\mathfrak{L}_1, R)$, $s_2 = (\mathfrak{L}_2, R)$, $R'\mathfrak{L}_1 R$ and $R'\mathfrak{L}_2 R$.
  By Theorem 3, there exists $\mathfrak{L}_3$ such that $\mathfrak{L}_1 \leq \mathfrak{L}_3$ and $\mathfrak{L}_2 \leq \mathfrak{L}_3$.
  Therefore, if we write s_3 for $(\mathfrak{L}_3, R)$, then $s_3 \in E$, $s_1 \sim s_3$ and $s_2 \sim s_3$. Hence, by the fact that $\sim$ is an equivalence relation, we have $s_1 \sim s_2$.

□

4 Making CCS Reversible

In this section we give a refinement of CCS with recursion so that we can apply the framework described in Section 3. See [18] for details of CCS.

Assume that we have a set of channels a and a set of process variables X. In our refinement, as specified in Section 3, terms are defined by the same grammar used in (standard) CCS:

$$\begin{array}{lcl} P,Q & ::= & \sum_{i\in I} \alpha_i.P_i \mid (P|Q) \mid \nu a.P \mid 0 \mid X \mid \mathrm{rec}X.P \\ \alpha & ::= & a \mid \overline{a} \end{array}$$

Action a denotes an input on channel a, while $\overline{a}$ is the corresponding output. Nondeterministic choice $\sum_{i\in I} \alpha_i.P_i$ can perform any action α_i and continue as P_i. $P|Q$ is parallel composition. Process $\nu a.P$ denotes that channel a is local to P. 0 is the process that does nothing. Construct $\mathrm{rec}X.P$ allows the definition of recursive processes. Transitions in CCS are of the form $P \xrightarrow{\alpha}_{ccs} P'$ where α is a, $\overline{a}$ or τ (internal synchronization).

The following grammar defines labels for our refinement:

$$u, v ::= ((\alpha_j, P_j)_{j\in I}, i) \mid (u|\bullet) \mid (\bullet|u) \mid (u|v) \mid \nu a.u \mid \mathrm{rec}X.P$$

We consider terms and labels up to α-equivalence (of both variables X and channels a). Therefore, we can define the subsitution $P\{X := Q\}$ avoiding variable capture. We define the interpretation of labels as follows (partial function):

$$\begin{array}{lcllcll} [\![((\alpha_j,P_j)_{j\in I},i)]\!] & := & \alpha_i & [\![\mathrm{rec}X.P]\!] & := & \tau & \\ [\![(u|\bullet)]\!] & := & [\![u]\!] & [\![(\bullet|u)]\!] & := & [\![u]\!] & \\ [\![(u|v)]\!] & := & \tau & [\![\nu a.u]\!] & := & [\![u]\!] & ([\![u]\!] \notin \{a, \overline{a}\}) \end{array}$$

We define transitions with the rules in Table 1.

Proposition 1. *$P \xrightarrow{u} P'$ iff $[\![u]\!]$ exists and $P \xrightarrow{[\![u]\!]}_{ccs} P'$.*

Proof. By induction on $P \xrightarrow{u} P'$ for the forward implication, and by induction on $P \xrightarrow{[\![u]\!]}_{ccs} P'$ for the backward implication: each rule here corresponds to a rule in the semantics of CCS [18]. □

Now we only have to define a suitable $\perp$. Below, ξ stands for u or $\bullet$.

$$\frac{}{u \perp \bullet} \qquad \frac{}{\bullet \perp u} \qquad \frac{\xi_1 \perp \xi_1' \quad \xi_2 \perp \xi_2'}{(\xi_1|\xi_2) \perp (\xi_1'|\xi_2')} \qquad \frac{u \perp v}{\nu\alpha.u \perp \nu\alpha.v}$$

Informally, $u \perp v$ means that the transitions described by u and v operate on separate processes.

Theorem 5. *The LTS and the relation $\perp$ defined above satisfy Theory 1.*

Thanks to this result, we can apply the framework of Section 3 to obtain a causal-consistent reversible semantics for CCS. We also have for free results such as the Loop lemma or causal consistency.

$$\frac{i \in I}{\sum_{j\in I} \alpha_j.P_j \xrightarrow{((\alpha_j,P_j)_{j\in I},i)} P_i} \qquad \frac{P \xrightarrow{u} P'}{(P|Q) \xrightarrow{(u|\bullet)} (P'|Q)}$$

$$\frac{Q \xrightarrow{u} Q'}{(P|Q) \xrightarrow{(\bullet|u)} (P|Q')} \qquad \frac{P \xrightarrow{u} P' \quad [\![u]\!] \notin \{a,\overline{a}\}}{\nu a.P \xrightarrow{\nu a.u} \nu a.P'}$$

$$\frac{P \xrightarrow{u} P' \quad Q \xrightarrow{v} Q' \quad ([\![u]\!] = \alpha \wedge [\![v]\!] = \overline{\alpha}) \vee ([\![u]\!] = \overline{\alpha} \wedge [\![v]\!] = \alpha)}{(P|Q) \xrightarrow{(u|v)} (P'|Q')}$$

$$\frac{}{\text{rec}X.P \xrightarrow{\text{rec}X.P} P\{X := \text{ rec}X.P\}}$$

Table 1: Refined transitions for CCS

Example 1. *Consider the CCS process* $a.b.0|\overline{b}.c.0$*. We have, e.g., the two computations below:*
$a.b.0|\overline{b}.c.0 \xrightarrow{((a,b.0),1)|\bullet} b.0|\overline{b}.c.0 \xrightarrow{\bullet|((\overline{b},c.0),1)} b.0|c.0$
$a.b.0|\overline{b}.c.0 \xrightarrow{((a,b.0),1)|\bullet} b.0|\overline{b}.c.0 \xrightarrow{((b,0),1)|((\overline{b},c.0),1)} 0|c.0 \xrightarrow{\bullet|((c,0),1)} 0|0$
In the first computation $((a,b.0),1)|\bullet \perp \bullet|((\overline{b},c.0),1)$*, hence the two actions can be reversed in any order. In the second computation neither* $((a,b.0),1)|\bullet \perp ((b,0),1)|((\overline{b},c.0),1)$ *nor* $((b,0),1)|((\overline{b},c.0),1) \perp \bullet |((c,0),1)$ *hold, hence the actions are necessarily undone in reverse order. These two behaviors agree with both the standard notion of concurrency in CCS, and with the behaviors of the causal-consistent reversible extensions of CCS in the literature [8, 19]. Indeed we conjecture that our semantics and the ones in the literature are equivalent.*

More in general, in $(P|Q)$ reductions of P and of Q are concurrent as expected, and the choice of reducing first P, then Q or the opposite has no impact: if $P \xrightarrow{u} P'$ and $Q \xrightarrow{v} Q'$, then we can permute $(P|Q) \xrightarrow{(u|\bullet)} (P'|Q) \xrightarrow{(\bullet|v)} (P'|Q')$ into $(P|Q) \xrightarrow{(\bullet|v)} (P|Q') \xrightarrow{(u|\bullet)} (P'|Q')$. Also, e.g., in the rule for right parallel composition, the label $(u|\bullet)$ is independent from Q. For this reason we can permute the order of two concurrent transitions without changing the labels.

Labels can be seen as derivation trees in the original CCS with some information removed. Indeed, for every derivation rule in CCS, there is a production in the grammar of the labels. When extracting the labels from the derivation trees:

- We must keep enough information from the original derivation tree to preserve determinism and co-determinism.
- We must remove enough information so that we can permute concurrent transitions without changing the labels. For example, in the label $(u|\bullet)$, there is no information on the term which is not reduced.
- Our labels and transition rules are close to the ones of the causal semantics of Boudol and Castellani [5]. Actually, our labels are a refinement of the ones of Boudol and Castellani: we need this further refinement since their transitions are not co-deterministic.

If we chose to take the whole derivation tree as a label, we would have the same problem as in Remark 1: we would have determinism and co-determinism, but we would not be able to have a definition of $\perp$ capturing the concurrency model of CCS.

5 Examples with X-machines

X-machines [10] (also called Eilenberg machines) are a model of computation that is a generalization of the concept of automaton. Basically, they are just automata where transitions are relations over a set D. The set D represents the possible values of a memory, and transitions modify the value of the memory.

Definition 7 (X-machines)**.**
An X-machine *on a set D is a tuple* $\mathscr{A} = (Q, I, F, \delta)$ *such that:*

- *Q is a finite set of states.*
- *I and F are subsets of Q, representing initial and final states.*
- *δ is a finite set of triplets (q, α, q') such that $q, q' \in Q$ and α is a binary relation on D, defining the transitions of the X-machine.*

The semantics of an X-machine is informally described as follows:

- The X-machine takes as input a value of D and starts in an initial state.
- When the X-machine changes its state with a transition, it applies the relation to the value stored in the memory.
- The value stored in the memory in a final state is the output.

This can be formalized as an LTS whose configurations are pairs (q,x) where q is the state of the X-machine and x the value of the memory, and transitions are derived using the following inference rule:

$$\frac{(q, \alpha, q') \in \delta \quad (x,y) \in \alpha}{(q,x) \xrightarrow{\alpha} (q', y)}$$

X-machines are naturally a good model of sequential computations (both deterministic and non-deterministic). In particular, a Turing machine can be described as an X-machine [10].

X-machines have also been used as models of concurrency [4]. Below we will consider only a simple concurrent model: several X-machines running concurrently and working on the same memory. This represents a set of sequential processes, one for each machine, interacting using a shared memory. We will start from the case where there are only two machines. We will refine this model so that the refinement belongs to Theory 1 and so that we can apply our framework.

First, we want to extend a single X-machine to make it reversible. Notice that a single X-machine is a sequential model, hence at this stage we have a trivial concurrency relation. The LTS may be not determinstic and/or not co-deterministic both because of the relation δ, and because of the relation α. Hence, we will need to refine labels. For δ, we can use the trick in Remark 1. For actions α, the trick is to split each action α into a family of (deterministic and co-deterministic) relations $(\alpha_i)_{i \in I}$ such that $\alpha = \bigcup_{i \in I} \alpha_i$, and add to the label the index i of the used α_i.

Definition 8. *Assume D is a set, and $\alpha, \beta \in \mathscr{P}(D \times D)$. We write $\alpha \perp \beta$ if and only if $\alpha \circ \beta = \beta \circ \alpha$, where $\circ$ is the composition of relations.*

Definition 9. *We call a* refined action *on D an object a such that:*

- *$H(a)$ is a set, representing the indices of the elements of the splitting.*
- *For all $i \in H(a)$, $a(i) \in \mathscr{P}(D \times D)$ with $a(i)$ and $a(i)^{-1}$ functional relations ($a(i)$ is deterministic and co-deterministic).*

We can notice that $\bigcup_{i\in H(a)} a(i)$ is indeed a relation on D. Therefore, by forgetting information added by the refinement (how the action is split), a refined action can be interpreted as a simple relation on D.

For example, the following relations on X^3:

$$\alpha = \{((x,y,z),(x,x,z)) \mid x,y,z \in X\} \quad \beta = \{((x,y,z),(x,y,x)) \mid x,y,z \in X\}$$

are not co-deterministic but can be refined as follows:

Example 2. *Assume X is a set and $D = X^3$. We define the refined actions a and b on D by:*

- *Setting $H(a) = H(b) = X$*
- *For all $y \in X$, $a(y) := \{((x,y,z),(x,x,z)) \mid x,z \in X\}$.*
- *For all $z \in X$, $b(z) := \{((x,y,z),(x,y,x)) \mid x,y \in X\}$.*

Here D represents a memory with three variables with values in X. The action a (resp. b) copies the first variable to the second (resp. third) variable, indeed $\alpha = \bigcup_{y\in X} a(y)$ and $\beta = \bigcup_{z\in X} b(z)$.

We can notice that for all $y,z \in X$, $a(y) \perp b(z)$. These actions a and b are indeed independent. Furthermore, when actions are permuted, the indices (here y and z) remain the same: $a(y)\circ b(z) = b(z)\circ a(y)$.

It is always possible to refine a given action by splitting it into singletons: For instance, the above actions α and β can also be refined as follows:

Example 3. *We define the refined actions a' and b' as follows:*

- *$H(a') := H(b') = X^3$.*
- *For all $x,y,z \in X$, $a'(x,y,z) := \{((x,y,z),(x,x,z))\}$.*
- *For all $x,y,z \in X$, $b'(x,z,z) := \{((x,y,z),(x,y,y))\}$.*

The refined action a' (resp. b') is another splitting of the action a (resp. b).

Unfortunately we generally do not have $a'(i)\circ b'(j) = b'(j)\circ a'(i)$. Therefore a (resp. b) and a' (resp. b') describe the same relation α (resp. β) on D but do not allow the same amount of concurrency. Indeed, a and b allow one to define a non trivial $\perp$ while a' and b' do not.

The two previous examples show that when refining an action, the splitting must not be too thin to have a reasonable amount of concurrency. In the examples above, a and b is a good refinement of α and β but a' and b' is not. The reason is the same as in Remark 1.

We can notice that, usually, we can refine any action that reads and writes parts of the memory and the splitting must be indexed by the erased information, if the original relation is not co-deterministic, and by the created information, if the original relation is not deterministic.

The next example shows how to model with a refined X-machine a simple imperative language:

Example 4. *An environment ρ is a total map from an infinite set of variables to $\mathbb{N}$ such that the set of variables x such that $\rho(x) \neq 0$ is finite. Let D be the set of environments.*

1. *Assume that $x_1, \ldots, x_n$, y are variables and f is a total map from $\mathbb{N}^n$ to $\mathbb{N}$. Let α be the action on D defined as follows:*

   $$\alpha := \{(\rho, \rho[y \leftarrow f(\rho(x_1),\ldots,\rho(x_n))] \mid \rho \in D\}$$

   Then, α can be refined by defining the action a as follows:

   - *$H(a) := \mathbb{N}$*
   - *For all $v \in H(a)$, $a(v) := \{(\rho,\rho') \mid \rho(y) = v \wedge \rho' = \rho[y \leftarrow f(x_1,\ldots,x_n)]\}$.*

   This refined action is written $y \leftarrow f(x_1,\ldots x_n)$. When f is injective, we do not need to refine α: It is already deterministic and co-deterministic.

2. *Similarly, we can define the action $y{+}{=}\ f(x_1,\ldots,x_n)$ when for all i, $x_i \neq y$. This is the form of assignment used by Janus [21], a language which is naturally reversible, and where reversibility is ensured by restricting the allowed constructs w.r.t. a conventional language. Indeed, we can see that the corresponding relation is deterministic and co-deterministic.*
3. *Assume that $x_1, \ldots, x_n$ are variables and u is a subset of $\mathbb{N}^n$. Let α be the action defined as follows:*
$$\alpha := \{(\rho,\rho) \mid \rho \in D \wedge (\rho(x_1),\ldots\rho(x_n)) \in u\}$$
This action is used to create a branching instruction. Relation α is already deterministic and co-deterministic, hence we do not need to refine it. It is written $(x_1,\ldots x_n) \in u?$.

Then, we can define orthogonality as usual: two actions are dependent if there is a variable that both write, or that one reads and one writes, orthogonal otherwise. The functions rv and wv below compute the sets of read variables and of written variables, respectively.

$$rv(y \leftarrow f(x_1 \ldots x_n)) := \{x_1,\ldots x_n\} \qquad wv(y \leftarrow f(x_1 \ldots x_n)) := \{y\}$$
$$rv((x_1 \ldots x_n) \in u?) := \{x_1,\ldots x_n\} \qquad wv((x_1,\ldots x_n) \in u?) := \emptyset$$
$$rv(y{+}{=}\ f(x_1,\ldots,x_n)) := \{x_1,\ldots x_n\} \qquad wv(y{+}{=}\ f(x_1,\ldots,x_n)) := \{y\}$$

We have $a \perp b$ if and only if all the following conditions are satisfied:

$rv(a) \cap wv(b) = \emptyset \qquad rv(b) \cap wv(a) = \emptyset \qquad wv(a) \cap wv(b) = \emptyset$

We can then check that if $a \perp b$, then $a \circ b = b \circ a$.

Now we can define a refined X-machine, suitable for reversibility:

Definition 10. *A* refined X-machine *on D is $\mathscr{A} = (Q,I,F,\delta)$ such that:*

- *Q is a finite set.*
- *I and F are subsets of Q.*
- *δ is a finite set of triplets (q,a,q') such that $q,q' \in Q$ and a is a refined action.*

If we forget the refinement of the action we exactly have an X-machine. In the semantics, each label would contain both the used action a and the index of the element of the split which is used.

We can now build systems composed by many X-machines interacting using a shared memory.

Example 5. *Assume we have two X-machines $\mathscr{A}_1 = (Q_1,I_1,F_1,\delta_1)$ and $\mathscr{A}_2 = (Q_2,I_2,F_2,\delta_2)$ on D. We want to describe a model composed by the two X-machines, acting on a shared memory.*

Terms M are of the form (q_1,q_2,x) where $q_1 \in Q_1$ is the current state of the first X-machine, $q_2 \in Q_2$ is the current state of the second X-machine and $x \in D$ is the value of the memory. Labels u for the transitions are of the form (k,q,a,q',i) with $k \in \{1,2\}$, $(q,a,q') \in \delta_k$ and $i \in H(a)$. Here k indicates which X-machine moves, q, a and q' indicate which transition the moving X-machine performs, and i indicates which part of the relation is used.

The relation $\perp$ is defined as follows:
$$(k,q_1,a,q'_1,i) \perp (k',q_2,b,q'_2,j) \qquad \textit{iff} \qquad k \neq k' \wedge a(i) \perp a(j)$$
It means that two steps are independent if and only if they are performed by two distinct X-machines and the performed actions are independent.

Transitions are defined by the following rules:
$$\frac{(q_1,a,q'_1) \in \delta_1 \quad i \in H(a) \quad (x,y) \in a(i)}{(q_1,q_2,x) \xrightarrow{(1,q_1,a,q'_1,i)} (q'_1,q_2,y)}$$
$$\frac{(q'_2,a,q'_2) \in \delta_2 \quad i \in H(a) \quad (x,y) \in a(i)}{(q_1,q_2,x) \xrightarrow{(2,q_2,a,q'_2,i)} (q_1,q'_2,y)}$$

Then, we have the objects and properties of Theory 1. In particular, we have determinism and co-determinism, because in the label $u = (k, q, a, q', i)$, $a(i)$ is deterministic and co-deterministic.

Example 6. *Example 5 can be generalized to n refined X-machines (the terms M being of the form $(q_1, \ldots, q_n, x)$).*

Example 7. *By adding to the model in Example 6 the restriction "All the X-machines are equal" we do not lose any expressiveness. This will be relevant for the next example. This is shown in Appendix A.*

Example 8. *If the set of initial states of each X-machine is a singleton, we can generalize Example 7 to a potentially infinite number of X-machines, where however only a finite amount of them are not in their initial state. However, an unbounded number of X-machines may have moved.*

More formally, if we have a refined X-machine $\mathscr{A} = (Q, \{i_0\}, F, \delta)$.

- *The labels u are of the form (k, q, a, q', i) with $k \in \mathbb{N}$, $(q, a, q') \in \delta$, and $i \in H(a)$.*
- *The terms M are of the form (f, x) with $x \in D$ and f a total map from $\mathbb{N}$ to Q such that the set of $k \in \mathbb{N}$ with $f(k) \neq i_0$ is finite.*
- *$\perp$ and $\xrightarrow{u}$ are defined similarly to their respective counterpart in Example 5.*

The objects above satisfy the properties of Theory 1.

In all the previous examples, as for CCS, we can apply the framework of Section 3 to define a causal-consistent reversible semantics and have for free Loop lemma and causal consistency.

Example 8 allows one to simulate the creation of new processes dynamically. Moreover, since we have no limitation on D, we can choose D so to represent an infinite set of communication channels. We can also add the notion of synchronization between two X-machines. Therefore we conjecture that by extending Example 8 we could get a reversible model as expressive as the π-calculus.

6 Conclusion and Future Work

We have presented a modular way to define causal-consistent reversible extensions of formalisms as different as CCS and X-machines. This contrasts with most of the approaches in the literature [8, 13, 7, 16, 11], where specific calculi or languages are considered, and the technique is heavily tailored to the chosen calculus. However, two approaches in the literature are more general. [19] allows one to define causal-consistent reversible extensions of calculi in a subset of the path format. The technique is fully automatic. Our technique is not, but it can tackle a much larger class of calculi. In particular, X-machines do not belong to the path format since their terms include an element of the set of values X, and X is arbitrary. [9] presents a categorical approach that concentrates on the interplay between reversible actions and irreversible actions, but provides no results concerning reversible actions alone.

As future work we plan to apply our approach to other formalisms, starting from the π-calculus, and to draw formal comparisons between the reversible models in the literature and the corresponding instantiations of our approach. We conjecture to be able to prove the equivalence of the models, provided that we abstract from syntactic details. A suitable notion of equivalence for the comparison is barbed bisimilarity. Finally, we could also show that the construction from terms to reversible configurations given in Section 3 is actually monadic and that the algebras of this monad are also relevant, since they allow one to inject histories into terms and make them reversible.

References

[1] T. Akgul & V. J. Mooney III (2004): *Assembly instruction level reverse execution for debugging*. *ACM Trans. Softw. Eng. Methodol.* 13(2), pp. 149–198.

[2] T. Altenkirch & J. Grattage (2005): *A Functional Quantum Programming Language*. In: *LICS*, IEEE Computer Society, pp. 249–258.

[3] A. Avizienis, J.-C. Laprie, B. Randell & C. E. Landwehr (2004): *Basic Concepts and Taxonomy of Dependable and Secure Computing*. *IEEE Trans. Dependable Sec. Comput.* 1(1), pp. 11–33.

[4] J. Barnard, J. Whitworth & M. Woodward (1996): *Communicating X-machines*. *Information and Software Technology* 38(6), pp. 401–407.

[5] G. Boudol & I. Castellani (1988): *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*, 354, Springer, pp. 411–427, doi:10.1007/BFb0013028.

[6] L. Cardelli & C. Laneve (2011): *Reversible structures*. In: *CMSB*, ACM, pp. 131–140, doi:10.1145/2037509.2037529.

[7] I. D. Cristescu, J. Krivine & D. Varacca (2013): *A Compositional Semantics for the Reversible Pi-calculus*. In: *LICS*, IEEE Press, pp. 388–397.

[8] V. Danos & J. Krivine (2004): *Reversible Communicating Systems*. In: *CONCUR*, *LNCS* 3170, Springer, pp. 292–307.

[9] V. Danos, J. Krivine & P. Sobocinski (2006): *General Reversibility*. In: *EXPRESS*, *ENTCS* 175(3), pp. 75–86.

[10] S. Eilenberg & B. Tilson (1976): *Automata, languages and machines. Volume B*. Pure and applied mathematics, Academic Press.

[11] E. Giachino, I. Lanese, C. A. Mezzina & F. Tiezzi (2015): *Causal-Consistent Reversibility in a Tuple-Based Language*. In: *PDP*, IEEE Computer Society Press, pp. 467–475, doi:10.1109/PDP.2015.98.

[12] R. Landauer (1961): *Irreversibility and heat generation in the computing process*. *IBM Journal of Research and Development* 5, pp. 183–191.

[13] I. Lanese, C. A. Mezzina & J.-B. Stefani (2010): *Reversing Higher-Order Pi*. In: *CONCUR*, *LNCS* 6269, Springer, pp. 478–493.

[14] I. Lanese, C. A. Mezzina & J.-B. Stefani (2016): *Reversibility in the higher-order π-calculus*. *Theor. Comput. Sci.* 625, pp. 25–84, doi:10.1016/j.tcs.2016.02.019.

[15] I. Lanese, C. A. Mezzina & F. Tiezzi (2014): *Causal-Consistent Reversibility*. *Bulletin of the EATCS* 114. Available at `http://eatcs.org/beatcs/index.php/beatcs/article/view/305`.

[16] M. Lienhardt, I. Lanese, C. A. Mezzina & J.-B. Stefani (2012): *A Reversible Abstract Machine and Its Space Overhead*. In: *FMOODS/FORTE*, *LNCS* 7273, Springer, pp. 1–17.

[17] A. W. Mazurkiewicz (1988): *Basic notions of trace theory*. In: *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pp. 285–363, doi:10.1007/BFb0013025.

[18] R. Milner (1989): *Communication and concurrency.* Prentice-Hall.

[19] I. Phillips & I. Ulidowski (2007): *Reversing Algebraic Process Calculi*. *J. Log. Algebr. Program.* 73(1-2), pp. 70–96.

[20] I. Phillips, I. Ulidowski & S. Yuen (2012): *A Reversible Process Calculus and the Modelling of the ERK Signalling Pathway*. In: *RC*, *LNCS* 7581, Springer, pp. 218–232.

[21] T. Yokoyama & R. Glück (2007): *A Reversible Programming Language and Its Invertible Self-interpreter*. In: *PEPM*, ACM Press, pp. 144–153, doi:10.1145/1244381.1244404.

A Proofs missing from the main part

Lemma 1 (Properties of $\asymp$)**.**

1. *$\asymp$ is an equivalence relation.*
2. *If $L_1 \asymp L_1'$ and $L_2 \asymp L_2'$, then $(L_1 L_2) \asymp (L_1' L_2')$.*
3. *If $(L_1 \xrightarrow{u}) \asymp L_2$, then there exist L_3 and L_4 such that $L_2 = (L_3 \xrightarrow{u} L_4)$, $L_1 \asymp (L_3 L_4)$ and for all v in L_4, $v \neq u$ and $u \perp v$.*
4. *If $(L_1 \xrightarrow{u}) \asymp (L_2 \xrightarrow{u})$, then $L_1 \asymp L_2$.*
5. *If for all $v \in L$, $u \perp v$, then $(L \xrightarrow{u}) \asymp (\xrightarrow{u} L)$.*
6. *If $L_1 \asymp L_2$, then $|L_1| = |L_2|$.*

Proof.

1. • Reflexivity: By definition.
   • Transitivity: We prove by induction on the derivation of $L_2 \asymp L_3$ that if $L_1 \asymp L_2$ and $L_2 \asymp L_3$, then $L_1 \asymp L_3$.
   • Symmetry: We prove by induction on the derivation of $L_1 \asymp L_2$ that if $L_1 \asymp L_2$, then $L_2 \asymp L_1$. In particular, we use the definition of $\asymp$ and transitivity.
2. First, we prove by induction on the derivation of $L_1 \asymp L_2$ that if $L_1 \asymp L_2$, then $(L_3 L_1 L_4) \asymp (L_3 L_2 L_4)$. Then, from $L_1 \asymp L_1'$ and $L_2 \asymp L_2'$, we can deduce $(L_1 L_2) \asymp (L_1' L_2)$ and $(L_1' L_2) \asymp (L_1' L_2')$. Hence, by transitivity $(L_1 L_2) \asymp (L_1' L_2')$.
3. By induction on the derivation of $(L_1 \xrightarrow{u}) \asymp L_2$. Intuitively, every v that moves to the right of the last u must satisfy $u \perp v$.
4. Corollary of item 3.
5. By induction on the length of L.
6. By induction on the derivation of $L_1 \asymp L_2$.

□

Lemma 2 (Properties of MLM')**.**

1. *The notation MLM' is a conservative extension of the notation $M \xrightarrow{u} M'$.*
2. *If ML_1M' and $M'L_2M''$, then $M(L_1L_2)M''$.*
3. *If M_1LM' and M_2LM', then $M_1 = M_2$.*
4. *If $M(L_1L_2)M'$, then there exists M'' such that ML_1M'' and $M''L_2M'$.*
5. *If MLM' and $L \asymp L'$, then $ML'M'$.*

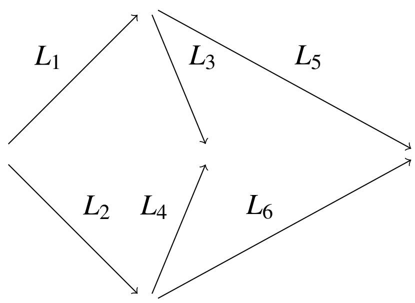
Proof.

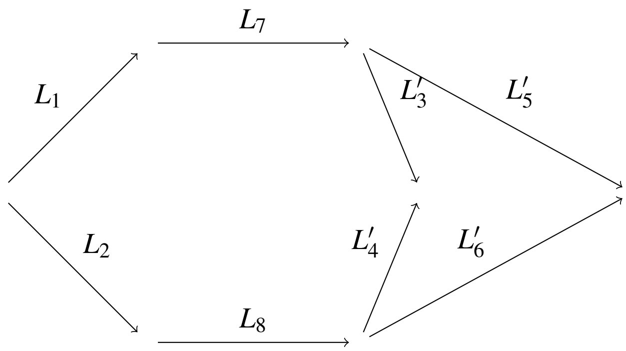
1. Straightforward.
2. By induction on the derivation of $M'L_2M''$.
3. By induction on the derivation of M_1LM'.
4. By induction on the derivation of ML_1L_2M'.

5. By induction on the derivation of $L \asymp L'$ and by using item 4.

□

Lemma 3. *If $(L_1L_3) \asymp (L_2L_4)$ and $(L_1L_5) \asymp (L_2L_6)$, then there exist L_7, L_8, L'_3, L'_4, L'_5, L'_6 such that $L_3 \asymp (L_7L'_3)$, $L_4 \asymp (L_8L'_4)$, $L_5 \asymp (L_7L'_5)$, $L_6 \asymp (L_8L'_6)$, $L'_3 \asymp L'_4$ and $L'_5 \asymp L'_6$.*





Proof. We prove the result by induction on $|L_3|+|L_4|+|L_5|+|L_6|$.

Assume that L_5 is not empty. Therefore, there exist u and L_7 such that $L_5 = L_7u$. Let n be the number of occurrences of u in $L_1L_5 = L_1L_7u$. Since $L_2L_6 \asymp L_1L_5$, L_2L_6 has n occurrences of u too.

We have a case analysis according to whether the n-th occurrence (from the left) of u in L_2L_6 is in L_2 or in L_6.

If it is in L_6, since $L_2L_6 \asymp L_1L_7u$, L_6 is of the form L_8uL_9 and for every v in L_9, $u \perp v$. Therefore, $L_6 \asymp L_8L_9u$, $L_1L_7u \asymp L_2L_8L_9u$ and $L_1L_7 \asymp L_2L_8L_9$. We can use the induction hypothesis with L_7 instead of L_5 and L_8L_9 instead of L_6 and conclude.

Now, if the n-th occurrence of L_2L_6 is in L_2, we have to prove the following intermediate result:

If $L_5 \asymp L_8vL_9$, this v is the m-th occurrence of v in $L_1L_8vL_9$, and the m-th occurrence of v in L_2L_6 is in L_2, then there exist L_{10}, w and k such that $L_5 \asymp wL_{10}$, this w is the k-th occurrence of w in L_1wL_{10} and the k-th occurrence of w in L_2L_6 is in L_2.

This intermediate result is proved by induction on the length of L_8:

- If L_8 is empty we can conclude.
- If $L_8 = L_{10}w$, $w \perp v$ and $w \neq v$, then $L_5 \asymp L_{10}vwL_9$. By induction hypothesis with L_{10} instead of L_8 and wL_9 instead of L_9, we can conclude.
- If $L_8 = L_{10}w$ and $w \perp v$ or $u = w$. There exist k such that this w is the k-th w in $L_1L_{10}wvL_9$. In every sequence equivalent to $L_1L_{10}wvL_9$, the k-th occurrence of w is at the left of the m-th occurrence of v. This is true, in particular, for L_2L_6. Since, the m-th occurrence of v in L_2L_6 is in L_2, so is the k-th occurrence of w. By induction hypothesis with L_{10} instead of L_8, w instead of v and vL_9 instead of L_9, we can conclude.

We can use the intermediate result to prove that there exist L_8, v and m such that $L_5 \asymp vL_8$, this v is the m-th occurrence of v in L_1vL_8 and the m-th occurrence of v in L_2L_6 is in L_2.

Therefore, L_2 has at least m occurrences of v. Hence, L_2L_4 has at least m occurrences of v. Since, $L_2L_4 \asymp L_1L_3$, L_1L_3 has at least m occurrences of v.

If the m-th occurrence of v in L_1L_3 is in L_1, then L_1 has at least m occurrences of v which is in contradiction with the fact that the m-th occurrence of v in L_1vL_8 is in not in L_1.

Therefore, the m-th occurrence of v in L_1L_3 must be in L_3. Hence, L_3 is of the form L_9vL_{10}.

- If, for every w in L_9, $w \perp v$, then $L_3 \asymp vL_9L_{10}$. We can apply the induction hypothesis with L_1v instead of L_1, L_8 instead of L_5 and L_9L_{10} instead of L_3.
- If not, there exist w and k such that we do not have $w \perp u$, the k-th occurrence of w in L_1L_3 is in L_3 and at the left of the m-occurrence of v. Therefore, the k-th occurrence of w in L_2L_4 is at the left of the m-th occurrence of v which is in L_2. Hence, the k-th occurrence of w in L_2L_4 is in L_2. So, L_2L_6 has at least k occurrence of w and the k-th occurrence of w in L_2L_6 is at the left of the m-th occurrence of v. Therefore, in L_1vL_8, the k-th occurrence of w is at the left of the m-th occurrence of v. Hence, it is in L_1. Therefore, L_1 has at least k occurrences of w which is in contradiction with the fact that the k-th occurrence of w in L_1L_3 is in L_3.

Therefore, we have proved that if L_5 is not empty, we can conclude. If L_3, L_4 or L_6 are not empty, we can do a similar proof. If all L_3, L_4, L_5 and L_6 are empty, then the result is trivial.

Therefore, the proof of the lemma is complete.

□

Lemma 5.

1. *If $R_1 \xrightarrow{u} R_2$, $R_2 \xrightarrow{v} R_3$ and $u \perp v$, then there exists R_2' such that $R_1 \xrightarrow{v} R_2'$ and $R_2' \xrightarrow{u} R_3$.*
2. *If $R_1 \xrightarrow{u^{-1}} R_2$, $R_2 \xrightarrow{v^{-1}} R_3$ and $u \perp v$, then there exists R_2' such that $R_1 \xrightarrow{v^{-1}} R_2'$ and $R_2' \xrightarrow{u^{-1}} R_3$.*
3. *If $R_1 \xrightarrow{u} R_2$ and $R_2 \xrightarrow{u^{-1}} R_3$, then $R_1 = R_3$.*
4. *If $R_1 \xrightarrow{u^{-1}} R_2$ and $R_2 \xrightarrow{u} R_3$, then $R_1 = R_3$.*
5. *If $R_1 \xrightarrow{u} R_2$, $R_2 \xrightarrow{v^{-1}} R_3$ and $u \neq v$, then there exists R_2', such that $R_1 \xrightarrow{v^{-1}} R_2'$, $R_2' \xrightarrow{u} R_3$ and $u \perp v$.*

Proof.

1. Straightforward from the fact that sequences in configurations are considered up to $\asymp$.
2. Corollary of the Loop lemma and the previous item.
3. There exist L_1, L_2, L_3, M_1, M_2 and M_3 such that $R_1 = [L_1]M_1$ and $R_2 = [L_2]M_2$, $R_3 = [L_3]M_3$. Therefore, $M_1 \xrightarrow{u} M_2$, $(L_1 \xrightarrow{u}) \asymp L_2$, $M_3 \xrightarrow{u} M_2$ and $L_2 \asymp (L_3 \xrightarrow{u})$. Hence, $(L_1 \xrightarrow{u}) \asymp (L_3 \xrightarrow{u})$. Then, $M_1 = M_3$ and, by Lemma 1, item 4, $L_1 \asymp L_3$. Therefore, $R_1 = R_3$.
4. There exist L_1, L_2, L_3, M_1, M_2 and M_3 such that $R_1 = [L_1]M_1$ and $R_2 = [L_2]M_2$, $R_3 = [L_3]M_3$. Therefore, $M_2 \xrightarrow{u} M_1$, $L_1 \asymp (L_2 \xrightarrow{u})$, $M_2 \xrightarrow{u} M_3$ and $(L_2 \xrightarrow{u}) \asymp L_3$. Hence, $M_1 = M_3$ and $L_1 \asymp L_3$. Therefore, $R_1 = R_3$.
5. There exist L_1, L_2, L_3, M_1, M_2 and M_3 such that $R_1 = [L_1]M_1$ and $R_2 = [L_2]M_2$, $R_3 = [L_3]M_3$. Therefore, $M_1 \xrightarrow{u} M_2$, $(L_1 \xrightarrow{u}) \asymp L_2$, $M_3 \xrightarrow{v} M_2$ and $L_2 \asymp (L_3 \xrightarrow{v})$. Hence, $(L_1 \xrightarrow{u}) \asymp (L_3 \xrightarrow{v})$. By Lemma 1, item 3, there exist L_4 and L_5 such that $(L_3 \xrightarrow{v}) = (L_4 \xrightarrow{u} L_5)$, $L_1 \asymp L_4L_5$ and for all

$w \in L_5$, $u \neq w$ and $u \perp w$. If L_5 is empty, then $(L_3 \xrightarrow{v}) = (L_4 \xrightarrow{u})$, and $u = v$. This is a contradiction. Therefore, L_5 is not empty and there exist L_6 and w such that $L_5 = (L_6 \xrightarrow{w})$. Hence, $(L_3 \xrightarrow{v}) = (L_4 \xrightarrow{u} L_6 \xrightarrow{w})$. Therefore, $w = v$, $L_3 = (L_4 \xrightarrow{u} L_6)$. Moreover, $L_1 \asymp (L_4 L_5) = (L_4 L_6 \xrightarrow{v})$ and $R_1 = [L_1]M_1 = [L_4 L_6 \xrightarrow{v}]M_1$ is a configuration. Hence, there exists M_0 such that $M_0(L_4 L_6 \xrightarrow{v})M_1$. Therefore, there exists M_2' such that $M_0(L_4 L_6)M_2'$ and $M_2' \xrightarrow{v} M_1$. Hence, we have $[L_4 L_6 \xrightarrow{v}]M_1 \xrightarrow{v^{-1}} [L_4 L_6]M_2'$. Let $R_2' := [L_4 L_6]M_2'$. Therefore, $R_1 \xrightarrow{v^{-1}} R_2'$.

Moreover, by the fact that $M_2' \xrightarrow{v} M_1$, $M_1 \xrightarrow{u} M_2$ and $u \perp v$ (because $v \in L_5 = (L_6 \xrightarrow{v})$), there exists M_1' such that $M_2' \xrightarrow{u} M_1'$ and $M_1' \xrightarrow{v} M_2$. By the fact that we also have $M_3 \xrightarrow{v} M_2$, we have $M_1' = M_3$. Hence, $M_2' \xrightarrow{u} M_3$. Therefore, $[L_4 L_6]M_2' \xrightarrow{u} [L_4 L_6 \xrightarrow{u}]M_3$. For all $w \in L_6$, $w \in L_5 = (L_6 \xrightarrow{v})$, and so $u \perp w$. By Lemma 2, item 5, $(L_6 \xrightarrow{u}) \asymp (\xrightarrow{u} L_6)$. Hence, $[L_4 L_6 \xrightarrow{u}]M_3 = [L_4 \xrightarrow{u} L_6]M_3 = [L_3]M_3 = R_3$. Therefore, $R_2' \xrightarrow{u} R_3$.

□

Theorem 1. *The LTS and the relation $\perp$ defined above satisfy Theory 1.*

Proof. Determinism and co-determinism are straightforward: the proof is by induction on the label, and we can notice that for each rule, from the label and a term, we have enough information to deduce the other term.

The proof of the co-diamond property is by induction on the first label. □

Details of the result in Example 7

We illustrate the reduction of the general case to the case where all the n X-machines are equal in the case $n = 2$ (which corresponds to Example 5). The generalization for n X-machines is straightforward.

- Let $D' := \{0,1,2\} \times \{0,1,2\} \times D$.
- Let $Q := \{i_0\} \cup Q_1 \cup Q_2$ (disjoint union).
- Let $I := \{i_0\}$.
- Let $F := F_1 \cup F_2$.
- For each refined action a on D, let $[\![a]\!]$ be the refined action on D' defined by:
  - $H([\![a]\!]) := H(a)$.
  - For all $i \in H(a)$, $[\![a]\!](i) := \{((k_1,k_2,x),(k_1,k_2,y)) \mid k_1,k_2 \in \{1,2\} \wedge k_1 \neq k_2 \wedge (x,y) \in a(i)\}$.
- If $k \in \{1,2\}$, then takerole_k is the refined action on D' defined by:
  - $H(\text{takerole}_k) := \{1,2\}$.
  - $\text{takerole}_k(1) := \{((0,k',x),(k,k',x)) \mid k' \in \{0,1,2\} \wedge k' \neq k \wedge x \in D\}$.
  - $\text{takerole}_k(2) := \{((k',0,x),(k',k,x) \mid k' \in \{0,1,2\} \wedge k' \neq k \wedge x \in D\}$.
- Let δ be a transition relation where elements are either:
  - $(i_0, \text{takerole}_k, q)$ with $k \in \{1,2\}$ and $q \in I_k$.
  - $(q, [\![a]\!], q')$ with $(q,a,q') \in \delta_k$ and $k \in \{1,2\}$.

This idea can be illustrated with a theatre play comparison:

- Every actor can play any role.
- Two different actors cannot play the same role.
- The play can only start when each role has been attributed to an actor.
- Even if an actor could have played any role, once he has chosen a role, he cannot change it.