HAL
archives-ouvertes.fr

A compositional semantics for the reversible pi-calculus

Ioana Domnina Cristescu, Jean Krivine, Daniele Varacca

▶ **To cite this version:**

Ioana Domnina Cristescu, Jean Krivine, Daniele Varacca. A compositional semantics for the reversible pi-calculus. Logic in Computer Science, 2013, New Orleans, United States. pp.388-397, 2013, <10.1109/LICS.2013.45>. <hal-00840156>

HAL Id: hal-00840156
https://hal.archives-ouvertes.fr/hal-00840156

Submitted on 1 Jul 2013

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

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A compositional semantics for the reversible π-calculus

Ioana Domnina Cristescu Jean Krivine Daniele Varacca
Univ. Paris Diderot and CNRS,
Laboratoire PPS, UMR 7126, F-75205 Paris, France

***Abstract*—We introduce a labelled transition semantics for the reversible π-calculus. It is the first account of a compositional definition of a reversible calculus, that has both concurrency primitives and name mobility.**

The notion of reversibility is strictly linked to the notion of causality. We discuss the notion of causality induced by our calculus, and we compare it with the existing notions in the literature, in particular for what concerns the syntactic feature of scope extrusion, typical of the π-calculus.

I. Introduction

A. Reversible computations

Being able to reverse a computation is an important feature of computing systems although not always studied as a topic of its own. In sequential systems, step by step rewinding of a computation is a common way of debugging programs. Also (reversible) programs running on logically reversible gates are known to have good properties with respect to energy consumption [1]. In the concurrent world, reversibility is a key aspect in every system that needs to solve distributed consensus [2], [3] in order to escape local states where the consensus cannot be found. However in the concurrent case, rewinding a computation requires to define first what is a legitimate backward move from a given state, in a context where the computation is no longer functional.

A formal model for concurrent systems needs to address two challenges at the same time: (i) how to compute without forgetting and (ii) what is an optimal notion of legitimate backward moves. Roughly speaking, the first point –that needs to be answered in the sequential world as well– is about syntax: processes need to carry a memory that keeps track of everything that has been done (and of the choices that have not been made). The second point is tied to the choice of the computation's semantics. In a concurrent setting we do not want to undo the actions precisely in the opposite order than the one in which they were executed, as this order is immaterial. The concurrency relation between actions has to be taken into account. Semantics that represent explicitly the concurrency of actions usually come equipped with a notion of *causality*.

We argue that the most liberal notion of reversibility is the one that just respects causality: an action can be undone precisely after all the actions that causally depend on it have also been undone.

This work is supported by the French National Research Agency (ANR), project REVER ANR 11 INSE 007

B. Our contributions

We are not the first to observe that causality and reversibility are tightly connected notions [4], [5]. Also, there are already several accounts of reversible operational semantics for CCS [6]–[8], and even of the (higher-order) π-calculus [9]. In spite of that, we think this paper makes important contributions.

First of all, we believe the existing approaches are not fully satisfactory. Distributed computations done in CCS are rather limited in scope because of the absence of name mobility. As soon as name creation (and exchange) is enabled, computing without forgetting becomes difficult because of the variable substitutions and also because the scope of a name that may increase in forward computation, should decrease accordingly when backtracking. Also, although the reversible Hoπ that has been proposed [9] is a clear gain in expressivity over CCS, it is only given in terms of reduction semantics and therefore not compositional.

We believe that the present study addresses the challenges that have been left aside in the previous works, namely a compositional definition of a reversible calculus, that has both concurrency primitives and name mobility. As we will see, achieving compositionality is far from trivial, in the same way as the standard labelled transition semantics of the π-calculus is not a trivial extension of its reduction semantics.

As a byproduct of compositionality, our contribution lies also in the realm of the non-interleaving semantics of the π-calculus. We take here the stance that the *concrete* events of a computation are the *reductions*, i.e. the steps that a closed system does. Labelled transitions are then considered as *abstract* or incomplete events, that await a suitable context to become concrete. In other words, they only exist for the sake of compositionality. As a consequence, the concrete causality relation between reductions is the one that is induced by the prefix operator (the "dot") and propagated through communications, also called *structural* dependence. Which notion of causality should then be considered on labelled transitions? For a simple calculus like CCS the answer cannot be disputed because causality between labelled transitions is also structural. But this is no

longer true in the π-calculus due to the dependency induced by the scope extrusion. To be as liberal as possible, the causality between labelled transitions, that should not be violated when backtracking, is the *smallest* relation that is consistent with the structural causality between reductions. More precisely, there should be a causal relation between two consecutive labelled transitions of a process only if the corresponding reductions obtained by "completion" of those transitions (by parallel composition) are also causally related. This would guarantee that if a backward labelled transition is not derivable in our semantics, it is because any corresponding reduction would violate the structural causality. There are several works that add different notions of causality to the labelled transition system of the π-calculus [10], [11]. Although the causal semantics that is induced by our semantics is related to them, ours is the only one, to the best of our knowledge, that satisfies the above requirement, which is formalized by Theorem 5.6 of Section V.

C. Other notable features

- In the purely forward direction, our semantics is just a decoration over the classical π-calculus: by forgetting additional annotations, we retrieve the (late) labelled transition semantics of the standard π-calculus. This can be considered as a sanity check.
- Our semantics is not only *compositional* but also *structural*. That is, the semantics of a process is obtained by structural rules from the semantics of its direct subprocesses. Compositionality requires in particular rules for the scope extrusion. Making these rules reversible is one of the main technical challenges of the present work.
- The notion of causality that is induced by our semantics is *stable*: every event carries with itself its unambiguous causal history. This is in contrast with the causal semantics of the π-calculus proposed in Ref. [12]. A full comparison of the present work with the event structure semantics is our current interest.

D. Outline

This paper is organized as follows: in Section II we introduce the syntax and the labelled transition semantics for the reversible π-calculus and we show its main properties in Section III. In Section IV, we then define the notion of equivalence up-to permutation that is induced by the semantics of our calculus. We then show that backtracking is done according to any path that is equivalent to the forward computation. In Section V we discuss the notion of causality induced by our semantics and show that it is maximally liberal with respect to the structural causality of the reduction semantics. In Section VI we conclude with some perspectives that our work suggests. Although this paper is self-contained, we assume the reader has some familiarity with the π-calculus.

II. The reversible π-calculus

In this section we present the compositional semantics of the reversible π-calculus (Rπ). In order to define the reversible operational semantics (Section II-B), we need first to introduce our meta variables and go through a few definitions (Section II-A).

A. Statics

1) Terms: We use a, b, c to range over channel names and P, Q to range over π calculus processes, defined as follows:

$$P, Q ::= 0 \mid \pi.P \mid (P \mid Q) \mid \nu a(P)$$

where $\pi ::= b(c) \mid \overline{b}\langle a\rangle \mid \tau$ denotes traditional π prefixes. We introduce neither choice nor replication. This restriction of expressivity is only in order to simplify the presentation, and these operators would pose no technical issues in the following developments.

As in RCCS [6], Rπ processes are built upon simple π processes to which we add a memory that will keep track of past actions. Every entry in a memory is called a *(memory) event* and can be used to trigger backward moves. From now on the term *process* will refer to Rπ processes.

We use $\mathcal{I}$ for the set of event identifiers, with a distinguished symbol $* \in \mathcal{I}$ that will denote partial synchronisation. Let i, j, k range over elements of $\mathcal{I}$ and Δ, Γ range over subsets of $\mathcal{I}$. Rπ terms are built according to the following grammar:

(Event labels) $\alpha ::= \overline{b}\langle a\rangle \mid b[\star/c] \mid b[a/c]$
(Memory events) $e ::= \langle i, k, \alpha\rangle$
(Memory stacks) $m ::= \varepsilon \mid \langle\uparrow\rangle.m \mid e.m$
(Rπ processes) $R, S ::= 0 \mid m \rhd P \mid (R \parallel S) \mid \nu a_\Gamma(R)$

In the style of RCCS, Rπ memories are structured as stacks, the top element being on the left and the empty stack being denoted by ε. There are two types of information that can be pushed on a memory: either a fork symbol $\langle\uparrow\rangle$, which allows memory stacks to divide whenever processes are forking, or events which are triplets of the form $\langle i, k, \alpha\rangle$. For any event $e = \langle i, k, \alpha\rangle$, we say that i is the *identifier* of e, k is the identifier of its *contextual cause* and α its *label*. The label of an event is used to record the prefix that was consumed during a transition, but also acts as an explicit substitution that allows one not to lose information about variable scope. We will come back to this important point in Section II-A3.

The notations $id(e), c(e)$ and $\lambda(e)$ give access to the identifier, the contextual cause and the label of e respectively.

The restriction $\nu a_\Gamma(R)$ and the parallel composition of processes $R \parallel S$ reflect the corresponding operators of π-processes thanks to the following structural rules:

$$m \rhd (P_1 \mid P_2) \equiv_m \langle\uparrow\rangle.m \rhd P_1 \parallel \langle\uparrow\rangle.m \rhd P_2 \quad (1)$$

$$m \rhd \nu a(P) \equiv_m \nu a_\emptyset(m \rhd P) \text{ with } a \notin m \quad (2)$$

which distribute a memory whenever two π processes are forking (1), and push classical π calculus restrictions at the level of processes (2). Note that an Rπ restriction is indexed by a set $\Gamma \subset \mathcal{I}$ (initially empty) and behaves as a classical restriction only when $\Gamma = \emptyset$. It will be used to keep track of past variable scope whenever $\Gamma \neq \emptyset$ (see Section II-A2).

It is noteworthy that not all syntactically correct processes are semantically meaningful. Indeed processes contain a computation history composed of past interactions, stored in the memories, and past variable scope, recorded by the νa_Γ constructs. History consistency cannot be easily enforced statically[1]. For the present work we only consider *reachable* terms, i.e. the set of terms that contains the obviously sound process $\varepsilon \rhd P$ and that is closed under the operational semantics of Rπ.

2) Names, scope and substitutions: In a process R, a channel a can be *bound* ($a \in \mathrm{bn}(R)$), *free* ($a \in \mathrm{fn}(R)$) or *liberated* ($a \in \mathrm{lib}(R)$). While free and bound names are as usual, one may think of liberated names as channels that used to be under the scope of a restriction that is no longer there because of an extrusion. They are the names that fall under the scope of the construct $\nu a_{\Gamma \neq \emptyset}(R)$, which then behaves as the "ghost" of a restriction in R with the set Γ containing the identifiers of all the events that have extruded the name a out of R.

Free and liberated names are defined inductively on the structure of processes ($+$ and $-$ denote classical operations on sets, $f(a)$ denotes either $\mathrm{fn}(a)$ or $\mathrm{lib}(a)$ whenever the distinction is irrelevant):

$$\begin{array}{lcl} f(\nu a_\emptyset R) & = & f(R) - a \\ (\Gamma \neq \emptyset) \quad f(\nu a_\Gamma R) & = & f(R) + a \\ f(R \parallel S) & = & f(R) + f(S) \\ \mathrm{fn}(m \rhd P) & = & names(m) + \mathrm{fn}(P) \\ \mathrm{lib}(m \rhd P) & = & \emptyset \\ \mathrm{fn}(b(a).P) & = & b + (\mathrm{fn}(P) - \{a\}) \\ \mathrm{fn}(\overline{b}\langle a \rangle.P) & = & \mathrm{fn}(P) + a + b \end{array}$$

with $names(m)$ being all the names occurring in the memory m. It is obvious from the above definition that all liberated names are free. As usual, names which are not free in R are called bound.

The operational semantics of Rπ is built on top of the so called "late" semantics of the π-calculus, where substitutions on variables occur upon synchronisation. Since substitutions are forgetful operations that cannot always be reversed correctly, we replace them with *explicit* substitutions that are logged in the event labels (see Section II-B2). We will also see that a process communicating on a liberated channel, has to make an assumption on the identity of the event that made the channel public (via an extrusion), called its contextual cause. Since the initial assumption can be made more precise while more structure of the process is revealed by the LTS, the contextual cause may also be updated in a "late" fashion. We thus need to define the following special substitutions on processes:

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

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

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

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

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

There are 3 relations between events that we need to consider.

Definition 2.2 (Relations on events): Let R be a process, we define the following relations on events of R.

- *Structural causal relation*: $e' \sqsubset_R e$ if there exists $m \in R$ such that $m = m_2.e.m_1.e'.m_0$ for some (possibly empty) m_2, m_1, m_0.
- *Contextual causal relation*: $e' \prec_R e$ if $c(e) = id(e')$.
- *Instantiation relation*: $e' \rightsquigarrow_R e$ if $e' \sqsubset_R e$ and $\lambda(e') = b[a/c]$, for some name a, b, c and c is in subject position in $\lambda(e)$. Furthermore for all memories m, we write $inst_m(c) = i$ if there is an event of the form $\langle i, k, b[a/c] \rangle$ in m that instantiates c. Note that there is at most one such event in m. If no such event exists in m we write $inst_m(c) = *$.

Example 2.1: In the process

$$\nu a_{\{i_2\}}(\nu a_{\{i_1\}}(\langle i_1, *, \overline{b}\langle a \rangle \rangle.m_1 \rhd P_1 \| \langle i_3, i_1, a \rangle.m_2 \rhd P_2)$$

$$\| \langle i_3, *, \overline{c} \rangle.\langle i_2, *, \overline{d}\langle c \rangle \rangle.\langle i_1, *, b[a/c] \rangle.m_3 \rhd P_3)$$

we have:

$$\begin{array}{l} \langle i_1, *, b[a/c] \rangle \sqsubset \langle i_2, *, \overline{c} \rangle \quad \langle i_1, *, \overline{b}\langle a \rangle \rangle \prec \langle i_2, i_1, a \rangle \\ \langle i_1, *, b[a/c] \rangle \rightsquigarrow \langle i_2, *, \overline{d}\langle c \rangle \rangle \quad inst_{\langle i_2, *, \overline{c} \rangle.\langle i_1, *, b[a/c] \rangle.m_3}(c) = i_1 \end{array}$$

For any events $e \in R$ and $e' \in R$ such that $id(e) = i$ and $id(e') = j$, we use the overloaded notations $i \sqsubset_R j$, $i \prec_R j$ or $i \rightsquigarrow_R j$, if e and e' are in the corresponding relation. Note that there are at most two events e and e' such that

[1]It is possible to characterize well-formedness as a set of properties that insures that processes (i) have at most one synchronisation partner in their past and (ii) that Γ-restrictions are consistent with a possible past scope.

$id(e) = id(e')$, in which case (e, e') forms a *synchronisation pair*.

B. Dynamics

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

$$\begin{array}{lcl} \gamma & ::= & \alpha \mid \alpha^- \\ \alpha & ::= & b(c) \mid \overline{b}\langle a\rangle \mid \overline{b}(\nu a_\Gamma) \end{array}$$

where $\overline{b}(\nu a_\Gamma)$ corresponds to the bound output of the π-calculus, whenever $\Gamma = \emptyset$, and otherwise corresponds to a free output, decorated with a set of event identifiers.

For all labels γ of the form α or α^-, we write $subj(\gamma) = b$ if $\alpha \in \{b(c), \overline{b}\langle a\rangle, \overline{b}(\nu a_\Gamma)\}$ for some a. We also write $\mathrm{bn}(\gamma) = \{a\}$ whenever $\alpha = \overline{b}\langle \nu a_{\Gamma \neq \emptyset}\rangle$ for some b. A transition is *positive* whenever its label is of the form α, and *negative* if the label is of the form α^-. It is *derivable* if it can be obtained from the LTS presented in the next section.

As we already hinted at, Rπ substitutions are not executed directly but simply logged in event labels. As a consequence, processes need to search in their memories for the public name of a channel in order to check that a synchronisation is possible. Such operation is performed only on demand, when a process is trying to reduce its prefix (see IN+ and OUT+ axioms in Section II-B2).

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

$$\begin{array}{lcll} \varepsilon[a] & = & a & \\ m[b(c)] & = & \underline{m[b]}(c) & \\ m[\overline{b}\langle a\rangle] & = & \overline{m[b]}\langle m[a]\rangle & \\ (\langle i, k, b[c/a]\rangle.m)[a] & = & c & \\ (\langle i, k, b[\star/a]\rangle.m)[a] & = & a & \\ (\langle\uparrow\rangle.m)[a] & = & m[a] & \\ (e.m)[a] & = & m[a] & otherwise \end{array}$$

2) The labelled transition system (LTS): The labelled transition system of Rπ can be divided into positive and negative rules. The negative ones are derived from the positive ones by inversion (see Definition 2.4). The positive rules are presented in Table I, where for $i, j \in I$, $i =_* j$ means $* \in \{i, j\}$ or $i = j$.

Note that the complete positive LTS contains also the symmetrical rules for the COM+, CLOSE+ and PAR+ rules with respect to the $\|$ operator. For lack of space, we do not write them explicitly.

The backward rules are derived according to the following definition:

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

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

OPEN+
$$\frac{R \xrightarrow{(i,j,k):\alpha} R' \qquad \alpha = \overline{b}\langle a\rangle \vee \alpha = \overline{b}\langle \nu a_{\Gamma'}\rangle}{\nu a_\Gamma R \xrightarrow{(i,j,k):\overline{b}\langle \nu a_\Gamma\rangle} \nu a_{\Gamma+i} R'}$$

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

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

CLOSE+
$$\frac{R \xrightarrow{(i,j,k):\overline{b}\langle \nu a_\Gamma\rangle} R' \qquad S \xrightarrow{(i,j',k'):b(c)} S'}{R \parallel S \xrightarrow{(i,*,*):\tau} \nu a_\Gamma (R' \parallel S'_{[a/c]@i})} \begin{array}{l} k =_* j' \\ k' =_* j \end{array}$$

$$with\ a \notin \mathrm{fn}(S)\ whenever\ \Gamma = \emptyset$$

PAR+
$$\frac{R \xrightarrow{(i,j,k):\alpha} R'}{R \parallel S \xrightarrow{(i,j,k):\alpha} R' \parallel S} \mathrm{bn}(\alpha) \cap \mathrm{fn}(S) = \emptyset, i \notin S$$

MEM+
$$\frac{R \equiv_m S \xrightarrow{\zeta} S' \equiv_m R'}{R \xrightarrow{\zeta} R'}$$

NEW+
$$\frac{R \xrightarrow{\zeta} R'}{\nu a_\Gamma R \xrightarrow{\zeta} \nu a_\Gamma R'} a \notin \zeta$$

Table I
THE POSITIVE RULES OF THE LTS

Definition 2.4 (Inverting operation): Let $\alpha^{-1} = \alpha^-$ and $(\alpha^-)^{-1} = \alpha$. Let *opp* be the operation defined in a functorial manner on labelled transition systems:

$$opp(R \xrightarrow{(i,j,k):\gamma} S) = S \xrightarrow{(i,j,k):\gamma^{-1}} R$$

and on derivation rules:

$$opp\left(\frac{R \xrightarrow{\zeta} S}{R' \xrightarrow{\zeta'} S'}\right) = \frac{opp(R \xrightarrow{\zeta} S)}{opp(R' \xrightarrow{\zeta'} S')}$$

$$opp\left(\frac{R_1 \xrightarrow{\zeta_1} S_1 \qquad R_2 \xrightarrow{\zeta_2} S_2}{T \xrightarrow{\zeta'} T'}\right) =$$

$$\frac{opp(R_1 \xrightarrow{\zeta_1} S_1) \qquad opp(R_2 \xrightarrow{\zeta_2} S_2)}{opp(T \xrightarrow{\zeta'} T')}$$

Side conditions are invariant. For all processes R, let $\mathcal{L}^+(R) = (R, \rightarrow)$ be the *positive* LTS of R and $\mathcal{L}^-(R) = (R, opp(\rightarrow))$ its *negative* version. The *reversible operational semantics* of R is defined as $\mathcal{L}(R) = \mathcal{L}^+(R) \cup \mathcal{L}^-(R)$.

3) Discussion:

Axioms: IN+ and OUT+ add an event e into the memory and apply the necessary substitutions on the transition label. The event identifier is locally fresh, as ensured by the side condition $i \notin m$.

Name extrusion: In Rπ, the role of the Γ-restriction $\nu a_\Gamma(R)$ is to act as a boundary that delimitates the past and present scope of a in R. Intuitively any partial synchronisation (either input or output) on channel a emanating from R needs to pick inside Γ an event identifier which will act as a proof that some process in the context knows a. As a consequence, if $\Gamma = \emptyset$ no partial synchronisation on a may cross this boundary and $\nu a_\emptyset$ behaves as a classical π-calculus restriction. The role of the OPEN+ rule is to update Γ each time a process in R is sending a to the context[2] (see also Example 2.2).

Importantly, because of possible successive extrusions, Γ-restrictions may be nested inside each others (see Proposition 3.5). Each time a partial synchronisation on a liberated name crosses such boundary, the LTS updates the contextual cause (i.e. the proof that a complete synch may eventually occur) that was chosen so far. The role of the CAUSE REF+ rule is to make sure a partial synchronisation on a chooses a correct contextual cause. Critically for the unicity of derivations (see Proposition 3.2), the way a cause is updated is not arbitrary, as indicated by the side condition of the CAUSE REF+ rule. In a nutshell, when passing a Γ-restriction, a contextual cause k may either be preserved if $k \in \Gamma$ or replaced by any $k' \in \Gamma$ such that $k \rightsquigarrow_R k'$. We will see that there always exists at least a $k' \in \Gamma$ such that $k \rightsquigarrow_R k'$ (see Propositions 3.4 and 3.6) so the CAUSE REF+ rule is never blocking if $\Gamma \neq \emptyset$.

Synchronisations: Two partial synchronisations may compose only if they agree on the public channel name in subject position (in the rule COM+ and CLOSE+ rules this is channel b). Such public name is deduced in the LTS at the level of the axiom applications. The side conditions of both synch rules proceed with the following intuition: if the left premise of transition i learned the name b thanks to an earlier communication j, then $j \neq *$ in the transition label. There are then two cases for the right premise of transition i: either $k' = *$, in which case no assumption was made on the contextual cause of this transition and the synchronisation may occur (since $j =_* *$), or $k' \neq *$. In the latter case, the leftmost sub-derivation had to cross a Γ-restriction and one must make sure that the chosen contextual cause k' coincides with the instantiator of the left derivation, i.e. $k' = j$. The argument is symmetric if one starts with the instantiator of the leftmost derivation.

Example 2.2: Consider the process (empty memory stacks and empty π-processes are omitted):

$$R = \nu a_\emptyset \left((\overline{b}\langle a \rangle \| \overline{c}\langle a \rangle) \| a \right)$$

The following trace is derivable (we use integers for identifiers and $(i, j, k) : \alpha$ is written $i : \alpha$ whenever $j = k = *$):

$$\begin{array}{rl} R & \xrightarrow{1:\overline{b}\langle \nu a_\emptyset \rangle} \quad \nu a_1 \left((\langle 1, *, \overline{b}\langle a \rangle \rangle \| \overline{c}\langle a \rangle) \| a \right) \\ & \xrightarrow{2:\overline{c}\langle \nu a_1 \rangle} \quad \nu a_{\{1,2\}} \left((\langle 1, *, \overline{b}\langle a \rangle \rangle \| \langle 2, *, \overline{c}\langle a \rangle \rangle) \| a \right) = R' \end{array}$$

There are now two possibilities to reduce the rightmost prefix a of R': the first one assuming that event 1 is the reason why a is "known" in the context, and the other one making the complementary assumption, namely that event 2 is the culprit. This yields the following two derivable transitions from R':

$$R' \xrightarrow{(3,*,1):a} \nu a_{\{1,2\}} \left((\langle 1, *, \overline{b}\langle a \rangle \rangle \| \langle 2, *, \overline{c}\langle a \rangle \rangle) \| \langle 3, 1, a \rangle \right) = T_1$$
$$R' \xrightarrow{(3,*,2):a} \nu a_{\{1,2\}} \left((\langle 1, *, \overline{b}\langle a \rangle \rangle \| \langle 2, *, \overline{c}\langle a \rangle \rangle) \| \langle 3, 2, a \rangle \right) = T_2$$

Notice here that T_1 (resp. T_2) may rollback event 2 (resp. event 1) while event 1 (resp. event 2) is backward blocked: indeed it is impossible to derive $T_1 \xrightarrow{1:\overline{b}\langle a \rangle^-}$ since the PAR- rule would require $1 \notin \langle 2, *, \overline{c}\langle a \rangle \rangle \| \langle 3, 1, a \rangle$. In fact, we will see Section III that backtracking respects both contextual and structural causes.

In order to illustrate how synchronisation is compositionally defined, let us consider the above derivations in a context where R is in parallel with $S = b(d).\overline{d}$ (which is called a reduction context in Section V). From S one may derive the following transition, that complements event 1:

$$S \quad \xrightarrow{1:b(d)} \quad \langle 1, *, b[\star/d] \rangle \rhd \overline{d}$$

Using the CLOSE+ rule, one may now compose both transitions identified by 1 (since $* =_* *$) and one gets:

$$\begin{array}{l} (R \| S) \xrightarrow{1:\tau} \nu a_\emptyset \left(\nu a_1 \left(\langle 1, *, \overline{b}\langle a \rangle \rangle \| \overline{c}\langle a \rangle \| a \right) \| \langle 1, *, b[a/d] \rangle \rhd \overline{d} \right) \\ \xrightarrow{2:\overline{c}\langle \nu a_1 \rangle} \\ \nu a_2 \left(\nu a_{\{1,2\}} \left(\langle 1, *, \overline{b}\langle a \rangle \rangle \| \langle 2, *, \overline{c}\langle a \rangle \rangle \| a \right) \| \langle 1, *, b[a/d] \rangle \rhd \overline{d} \right) \end{array}$$

using the PAR+ rule for the second transition. Now recall that there are two possible derivations from S in order to reduce the a prefix at the center of the above term. However only the first one can be composed with a transition on the $\overline{d}$ prefix on the right, since d is instantiated to a at event 1. Thus the only possibility[3] is to use the first derivation (with target T_1) in the COM+ rule composed with the derivation:

$$\langle 1, *, b[a/d] \rangle \rhd \overline{d} \xrightarrow{(3,1,*):\overline{a}} \langle 3, *, \overline{d} \rangle . \langle 1, *, b[a/d] \rangle$$

the side condition of the COM+ rule being satisfied.

Other rules: The rule PAR+ ensures freshness for the bound names and for the identifier i. In the PAR- rule, the side condition $i \notin S$ prevents a part of a synchronisation to backtrack by itself. Rule MEM+- rewrites the process in a form in which it can trigger a transition. Importantly only the MEM- rule allows one to pop the $\langle \uparrow \rangle$ symbol out of a

[2]Conversely, OPEN- will decrease the number of identifiers in Γ in order to take into account the fact that there is one less extruder for a.

[3]The second derivation from R' is still applicable but can only be used for a synchronisation occurring later in the context of the process.

memory. This ensures that a child process cannot backtrack below its spawning point, without reuniting first with its sibling. Lastly, in rule NEW+ if $\Gamma = \emptyset$ the process cannot do a transition that has the bound name a in its subject. If $\Gamma \neq \emptyset$ then the side conditions forces the usage of rules OPEN+-, CAUSE REF+-.

Not all side conditions are necessary for the backward transitions, as most of them are in fact invariant of the history consistency of processes. For simplifying the presentation however we keep them in both directions.

III. Properties

After presenting some interesting properties of the LTS, Section III-A, that may shed light on some subtle point of its behaviour, we show in Section III-B that the forward interpretation of an Rπ process is strongly bisimilar to its projection in the π-calculus.

A. Basic properties

First of all we observe that every transition can be undone.

Proposition 3.1 (Loop): For R reachable and for every forward transition $t : R \xrightarrow{\zeta} R'$ there exists a backward transition $t' : R' \xrightarrow{\zeta^-} R$, and conversely.

This is a trivial consequence of the symmetries of the rules.

An interesting property of proof systems, is that each transition has a unique derivation. Given the complexity of our rules, in particular the choice of the contextual cause (rule CAUSE REF), it is not trivial that our system enjoys such a property. Not only it does, but it does in a stronger sense for backward transitions.

Proposition 3.2: Two derivation trees have the same conclusion:

$$\frac{\pi_1}{R \xrightarrow{\zeta} S} \qquad \frac{\pi_2}{R \xrightarrow{\zeta} S}$$

if and only if $\pi_1 = \pi_2$.

Proposition 3.3: Suppose we have two negative transitions $R \xrightarrow{\zeta} S_1$ and $R \xrightarrow{\zeta} S_2$. Then $S_1 = S_2$.

The forward transitions do not have this property due to the nondeterminism in the choice of the synchronisation partners. In the backward case, however, this form of nondeterminism disappears.

The following propositions emphasize some important properties of well-formed terms, concerning Γ-restrictions within processes. First we notice that in any $T = C[\nu a_\Gamma R]$, event identifiers in Γ correspond exactly to extruders of a that occur in the memory of R.

Proposition 3.4: For all $T = C[\nu a_\Gamma R]$ reachable, for some context $C[\bullet]$, $i \in \Gamma$ if and only if $m_1.\langle i, \_, \overline{b}\langle c\rangle\rangle.m_2 \in R$ such that $m_2[c] = a$, and $\langle i, \_, d[a/e]\rangle \notin R$.

Then we show that, in a reachable process, all νa_Γ's on the same name a are nested.

Proposition 3.5: Let $\Gamma, \Delta \neq \emptyset$. In $T = C[\nu a_\Gamma(R) \parallel S]$ reachable, $\nu a_\Delta \notin S$.

A liberated name a occurs in a process or in its memory if the process was within the scope of the original $\nu a_\emptyset$ or if a was received through a synchronisation. This is formally stated in the following Proposition.

Proposition 3.6: In $T = C_1[C_2[\nu a_\Gamma R] \parallel S]$ reachable and $\Gamma \neq \emptyset$, if $a \in \mathrm{fn}(S)$ then $\langle i, \_, d[a/c]\rangle \in S$, for some $i \in \Gamma$.

B. Correspondence with the π-calculus

In Section II we have defined the reversible semantics of an Rπ process R as the LTS engendered by the union of $\mathcal{L}^+(R)$, the positive LTS of R, and $\mathcal{L}^-(R)$ its negative version. In order to claim that Rπ is a *reversible* π-calculus we need to prove that $\mathcal{L}^+(\varepsilon \rhd P)$, the positive interpretation of a π process P in $R\pi$, is bisimilar to P.

We define a forgetful operation which maps Rπ terms to π processes. To do so we simply need to:

- erase memories and νa_Γ annotations, whenever $\Gamma \neq \emptyset$;
- apply the substitutions stored in the memories before erasing them.

Formally we have:

Definition 3.7: Let ϕ be the forgetful map sending an $R\pi$ process R into the π-calculus, defined inductively on the structure of R as:

$$\begin{array}{lcl}
\phi(\varepsilon \rhd P) & = & P \\
\phi(R \parallel S) & = & \phi(R) \mid \phi(S) \\
\phi(\nu a_\emptyset R) & = & \nu a\,(\phi(R)) \\
\phi(\nu a_{\Gamma \neq \emptyset} R) & = & \phi(R) \\
\phi(\langle i, k, b[a/c]\rangle.m \rhd P) & = & \phi(m \rhd P\{a/c\}) \\
\phi(\langle \uparrow \rangle.m \rhd P) & = & \phi(m \rhd P) \\
\phi(e.m \rhd P) & = & \phi(m \rhd P) \quad \textit{otherwise}
\end{array}$$

The map ϕ naturally extends to transition labels with $\phi(i, j, k : \gamma) = \phi(\gamma)$ and:

$$\begin{array}{lcl}
\phi(\overline{b}\langle \nu a_{\Gamma \neq \emptyset}\rangle) & = & \overline{b}\langle a\rangle \\
\phi(\overline{b}\langle \nu a_\emptyset\rangle) & = & \overline{b}\langle \nu a\rangle \\
\phi(\alpha) & = & \alpha \quad \textit{otherwise}
\end{array}$$

For the π-calculus, we consider the *late* transition semantics, as presented for instance in [13]. Let $\rightarrow_\pi$ denote late transitions of the π-calculus, and $\rightarrow_+$ transitions of the positive LTS of Rπ.

Proposition 3.8: (Strong bisimulation between forward Rπ and its π-image) For all reachable process R, the pair $(R, \phi(R))$ is a (strong) bisimulation, i.e. we have:

1) If $R \xrightarrow{\zeta}_+ S$ then $\phi(R) \xrightarrow{\phi(\zeta)}_\pi \phi(S)$.
2) If $P \xrightarrow{\alpha}_\pi Q$ then $R \xrightarrow{\zeta}_+ S$ for all reachable R such that $\phi(R) = P$, for some S such that $\phi(S) = Q$ and with $\phi(\zeta) = \alpha$.

Although relatively straightforward the complete proof of Proposition 3.8 is quite lengthy and in practice, to

better carry it out, the translation is split into two parts: first removing the tagged restrictions and the memories, obtaining a π-calculus with explicit substitution. Then a second translation applies the substitutions.

As an immediate corollary of Proposition 3.8 one has that the pair $(\varepsilon \rhd P, P)$ is a bisimulation.

IV. Correctness of backtracking

In the introduction of this paper, we argued that we wanted our notion of reversibility to be as liberal as possible. As was already noted in RCCS [6] and subsequent work on reversible process algebra [8], [9], backtracking a trace should be allowed along any path that respects *causality*, or, said otherwise, backtracking can be done along any path that is *causally equivalent* to the path that was executed.

This property was formulated first in the work of the reversible CCS as a combination of a loop lemma, stating that any forward trace can be undone step by step, and a fundamental property that ensures that any two coinitial and cofinal traces are necessarily causally equivalent (see Fig. 1).

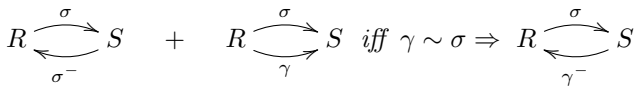


Figure 1. The conjunction of a loop property (left) and the fundamental property (right) ensures that after the forward trace σ, one may rollback to R along a causally equivalent past γ^-.

We already know that the loop property holds trivially for Rπ (Proposition 3.1). It remains to check that Rπ traces do exhibit the fundamental property, which depends on the equivalence on traces that is induced by the semantics of the language (denoted by $\sim$ in Fig. 1). For instance, the least liberal backtracking policy is obtained when the fundamental property holds only for trace equality.

This section follows closely the argument made in RCCS, in the context of Rπ. We will see (Lemma 4.3) that Rπ transitions contain enough information to characterize syntactically the concurrency and causality relations. This will let us define a notion of equivalence up-to permutation on traces (Definition 4.4) and prove the fundamental property for Rπ (Theorem 4.5). Later, in Section V, we will also argue that our notion of equivalence is, in a sense, optimal for reversing the π-calculus.

As usual, the causal equivalence class of a path is constructed by permuting the transitions that are concurrent. We proceed by defining the concurrency relation between transitions as the complement of causality.

Two transitions, t and t' are *composable*, written $t; t'$, if the target of t is the source of t'. A *trace*, denoted by $\sigma : t_1; \ldots; t_n$ is a sequence of composable transitions. The empty trace is denoted by ϵ. Two traces are *coinitial* if they have the same source and *cofinal* if they have the same target.

Definition 4.1 (Causality and concurrency): Let $t_1 : R \xrightarrow{(i_1,j_1,k_1):\gamma_1} S$ and $t_2 : S \xrightarrow{(i_2,j_2,k_2):\gamma_2} T$ be two transitions, where $t_1 \neq opp(t_2)$. We say that t_1 is:

- a *structural cause* of t_2, written $t_1 \sqsubset t_2$, if $i_1 \sqsubset_T i_2$ or $i_2 \sqsubset_R i_1$
- a *contextual cause* of t_2, written $t_1 \prec t_2$, if $i_1 \prec_T i_2$ or $i_2 \prec_R i_1$.

We simply say that t_1 *causes* t_2, written $t_1 < t_2$, if either $t_1 \sqsubset t_2$ or $t_1 \prec t_2$. Otherwise we say that they are *concurrent*.

It is worth noticing that for two consecutive transitions t and t', one may decide whether $t \prec t'$ by looking at their respective labels only, thanks to the following proposition:

Proposition 4.2: Let $t_1 : R \xrightarrow{(i_1,j_1,k_1):\gamma_1} S$ and $t_2 : S \xrightarrow{(i_2,j_2,k_2):\gamma_2} T$. We have $t_1 \prec t_2$ if either both transitions are positive and $k_2 = i_1$ or both transitions are negative and $k_1 = i_2$.

Example 4.1: Consider the following trace (with the conventions of Example 2.2):

$$\nu a_\emptyset(\overline{b}\langle a\rangle.a) \xrightarrow{1:\overline{b}\langle \nu a_\emptyset\rangle} \xrightarrow{(2,*,1):a} \nu a_1(\langle 2,1,a\rangle.\langle 1,*,\overline{b}\langle a\rangle\rangle)$$

where the first transition is both a structural and a contextual cause of the second; and consider the trace:

$$\nu a_\emptyset(\overline{b}\langle a\rangle \| \overline{c}\langle a\rangle) \xrightarrow{1:\overline{b}\langle \nu a_\emptyset\rangle} \xrightarrow{2:\overline{c}\langle \nu a_1\rangle} \nu a_{\{1,2\}}(\langle 1,*,\overline{b}\langle a\rangle\rangle \| \langle 2,*,\overline{c}\langle a\rangle\rangle)$$

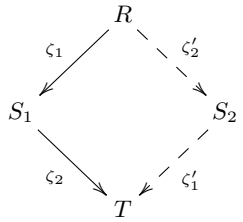
where the two transitions are this time concurrent.

We need now to show that the above syntactic definition of concurrency indeed coincides with commutability of transitions. We shall see that a particularity of Rπ is that commutation of concurrent transitions may not always be strictly label preserving but only *up-to* the label equivalence $=_\lambda$, defined as the least equivalence relation satisfying:

$$(i,j,k) : \overline{b}(\nu a_\Gamma) =_\lambda (i,j,k) : \overline{b}(\nu a_\Delta)$$

for all i, j, k, a, b and $\Gamma, \Delta \subset \mathcal{I}$.

Lemma 4.3 (Square): Consider two consecutive transitions $t_1 : R \xrightarrow{\zeta_1} S_1$ and $t_2 : S_1 \xrightarrow{\zeta_2} T$. If t_1 and t_2 are concurrent, there exist $t_2' : R \xrightarrow{\zeta_2'} S_2$ and $t_1' : S_2 \xrightarrow{\zeta_1'} T$, with $\zeta_i =_\lambda \zeta_i'$, such that the following diagram commutes:



Following the standard notation, we say that t_2 is the *residual* of t_2' after t_1 and write $t_2 = t_2'/t_1$.

Example 4.2: Back to Example 4.1, swapping the two concurrent transitions one obtains:

$$\nu a_\emptyset(\overline{b}\langle a\rangle \| \overline{c}\langle a\rangle) \xrightarrow{2:\overline{c}\langle \nu a_\emptyset\rangle} \xrightarrow{1:\overline{b}\langle \nu a_1\rangle} \nu a_{\{1,2\}}(\langle 1,*,\overline{b}\langle a\rangle\rangle \| \langle 2,*,\overline{c}\langle a\rangle\rangle)$$

Definition 4.4 (Equivalence up-to permutation): The equivalence *up-to permutation* of concurrent transitions, written $\sim$, is the least equivalence relation on traces satisfying:

$$t_1;(t_2/t_1) \sim t_2;(t_1/t_2) \qquad t;opp(t) \sim \epsilon$$

We can now state the fundamental property which proves that backtracking respects the causality induced by Rπ.

Theorem 4.5 (Fundamental property): Two traces are coinitial and cofinal if and only if they are equivalent.

The reader aware of the work on non-interleaving semantics for the π-calculus may have noticed that our semantics allows more transitions to commute than the standard ones [10], [11] in which the transitions of Example 4.2 are not considered concurrent. This is related to the fact that we let commutation preserve label up-to $=_\lambda$. We will come back to this important point in the next section.

V. Non interleaving semantics

Following the work of Lévy, on characterizing equivalence up-to permutation in the λ-calculus [14], it is well known that one may enrich the syntax of a calculus with concurrent redexes so as to track causal dependencies between reductions, either by annotating transition labels or directly by annotating the terms. This has been thoroughly studied in the 90s, in the context of CCS using static or dynamic locations and proof terms [15]–[18] and for the π-calculus as well although to a lesser extent [10], [11]. In this section we wish to view Rπ as an annotated version of the π-calculus, forgetting reversibility for a moment, and use the non-interleaving semantics it engenders in order to revisit, and update, standard concepts for the π-calculus.

In the absence of an indisputable definition of permutation equivalence for the LTS semantics of the π-calculus it is hard to assert the correctness of one definition over another.

As we have remarked in the introduction, in a closed system however (i.e. where only reductions are observed), there is a canonical definition of causality: the structural one. The semantics we have designed respects this intuition. Indeed, although contextual causality can be defined also for reductions, it is always hidden behind structural causality.

Proposition 5.1: Let $t_1 : R \xrightarrow{(i,*,*):\tau} S$ and $t_2 : S \xrightarrow{(j,*,*):\tau} T$. Then $t_1 < t_2$ if and only if $t_1 \sqsubset t_2$.

We want to justify contextual causality between labelled events as an "anticipation" of the structural causality between the reductions these events will generate. Or, dually, that if two labelled events are concurrent, then it is possible from them to generate two concurrent reductions. In order to formalize this intuition, given a process and one computation trace, we need a notion of *reduction context*, that provides a synchronising partner for every non-τ transition in the trace (see Definition 5.4).

Then the main result of this Section (Theorem 5.6) is that two non-τ transitions are concurrent if and only if there exists a reduction context that preserves concurrency.

In order to formulate the Theorem, we introduce the notion of projection that is used to retrieve from a synchronisation its two composing transitions.

In the following developments, we write $R \xrightarrow{i:\tau} S$ instead of $R \xrightarrow{(i,*,*):\tau} S$ whenever unambiguous.

Proposition 5.2: If $t : R \xrightarrow{i:\tau} S$ then there exists at most one context $C[\bullet]$ such that $t : C[R_1 \| R_2] \xrightarrow{i:\tau} C'[S_1 \| S_2]$ with $R_q \neq S_q, q \in \{1,2\}$.

The intuition of the above Proposition is that if the τ transition is generated by a prefix of the form $\tau.P$ then no such context exists. Otherwise, we can separate the two synchronising partners using the first parallel operator that is above them in the syntax tree.

Definition 5.3: The projections to the left and to the right of a transition t are defined as follows:

- if $t : C_1[R_1 \| R_2] \xrightarrow{i:\tau} C_2[S_1 \| S_2]$ with $R_q \neq S_q, q \in \{1,2\}$, then considering the derivation:

$$\frac{R_1 \xrightarrow{(i,j,k):\alpha} S_1' \quad R_2 \xrightarrow{(i,j',k'):\overline{\alpha}} S_2'}{R_1 \| R_2 \xrightarrow{i:\tau} S_1 \| S_2}$$

  we define:

$$\pi_l(t) : R_1 \xrightarrow{(i,j,k):\alpha} S_1' \qquad \pi_r(t) : R_2 \xrightarrow{(i,j',k'):\overline{\alpha}} S_2',$$

- otherwise, $\pi_l(t) = \pi_r(t) = t$.

Definition 5.4 (Reduction contexts): Given a trace

$$\sigma : R_0 \xrightarrow{\zeta_1} R_1 ... R_{n-1} \xrightarrow{\zeta_n} R_n$$

with $\zeta_q = (i_q, j_q, k_q) : \alpha_q$, a context $C[\bullet]$ is a *reduction context* for σ if: $C[R_0] \xrightarrow{i_1:\tau} R_1' ... R_{n-1}' \xrightarrow{i_n:\tau} R_n'$ for some $R_1', .., R_n'$ and, furthermore for each $q \in \{0, .., n\}$ we have:

- if $\alpha_q \neq \tau$ then there is $x \in \{l, r\}$ such that:

$$\pi_x(R_q' \xrightarrow{i_q:\tau} R_{q+1}') = R_q \xrightarrow{(i_q,j_q,k_q):\alpha_q} R_{q+1}$$

- if $\alpha_q = \tau$ then $\exists C'[\bullet]$ such that $R_q' = C'[R_q], R_{q+1}' = C'[R_{q+1}]$.

The fact that reduction contexts always exist may be viewed as a sanity check of the LTS semantics: for every derivable partial reduction, there is a context that makes it whole. This property holds trivially for CCS and the π-calculus but is not obvious in a more complex LTS such as the one of Rπ.

Proposition 5.5: In Rπ reduction contexts exist for every finite trace.

It is interesting to note that the Proposition 5.5 is important because we *derive* a reduction semantics from the LTS and there is no guarantee *a priori* that every transition can be part of some reduction. This should be put in contrast with the approach of Leifer and Milner which proposed a technique to derive LTS's from reduction semantics [19]. The reduction contexts of Definition 5.4 and the substitution stored in the memories also appear in a similar fashion in Ref. [20], where Leifer and Milner's approach is applied to the π-calculus.

Example 5.1: With the convention of Example 2.2, let us consider the process: $R = \nu a_\emptyset(\overline{b}\langle a\rangle \parallel a)$ with the trace emanating from it:

$$\sigma : R \xrightarrow{i_1:\overline{b}\langle \nu a_\emptyset\rangle} \xrightarrow{(i_2,*,i_1):a} S$$

A reduction context for σ is $C[\bullet] = ([\bullet] \parallel b(u).\overline{u})$, and the closure of the trace is:

$$\hat{\sigma} : C[R] \xrightarrow{i_1:\tau} \xrightarrow{i_2:\tau} C'[S]$$

with $i_1 \prec_S i_2$ and $i_1 <_{C'[S]} i_2$ as stated in Proposition 5.1.

We are now in position to state the theorem on which we rely to claim that the permutation equivalence induced by Rπ is optimal with respect to the reduction semantics.

Theorem 5.6: Let t_1 and t_2 be two transitions of the form:

$$t_1 : R \xrightarrow{(i_1,j_1,k_1):\alpha_1} S \qquad t_2 : S \xrightarrow{(i_2,j_2,k_2):\alpha_2} T.$$

We have that t_1 and t_2 are concurrent if and only if there exists a reduction context $C[\bullet]$ such that:

$$t_1' : C[R] \xrightarrow{i_1:\tau} S' \qquad t_2' : S' \xrightarrow{i_2:\tau} T'$$

and t_1' is concurrent to t_2'.

Proof: (Sketch) If the two transitions are concurrent, a reduction context preserving concurrency is $C[\bullet] = [\bullet] \parallel (\varepsilon \rhd \overline{\alpha_1} \parallel \varepsilon \rhd \overline{\alpha_2})$.

The other direction is done by showing its contrapositive, namely that if t_1 and t_2 are causal, then so are t_1' and t_2'. We have that $e_1 = \langle i_1, k_1, \alpha_1\rangle \in T$ and $e_2 = \langle i_2, k_2, \alpha_2\rangle \in T$. From the properties of a reduction context we have that $e_q = \langle i_q, k_q, \alpha_q'\rangle \in T'$ where: $\alpha_q' = b[c/a]$ if $\alpha_q = b[*/a]$ and $\alpha_q' = \alpha_q$ otherwise, for $q \in \{1,2\}$ and $x \in \{l, r\}$. Hence if $e_1 \sqsubset_T e_2$ then $e_1 \sqsubset_{T'} e_2$ (the order in which the events are added to the memory does not change) and if $e_1 \prec_T e_2$ then $e_1 \prec_{T'} e_2$ (the events do not change their contextual cause).

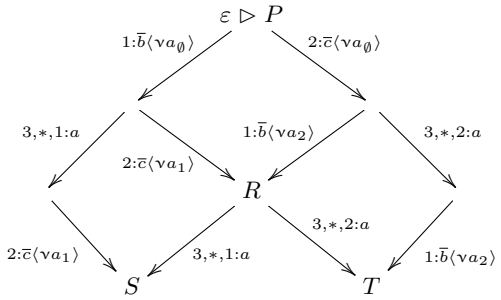
■

In particular the contrapositive of Theorem 5.6 implies that two transitions are causally related (either structurally or contextually) if and only if *for all* reduction contexts, the corresponding reductions are. This property is however not satisfied by the causal semantics of the π-calculus considered in earlier works, where the *first* extruder of a name a is considered to be the cause of any subsequent transition using a as a free name. In particular this prevents the transitions of Example 4.2 from commuting when there exists a reduction context which would let their closures commute.

While the proof of Theorem 5.6 is conceptually simple, some subtleties should nevertheless be pointed out. First, the fact that the causality relation is preserved by parallel composition is a design choice of our semantics, which is not shared by the other causal semantics in the literature. Also, even though causality is preserved, it is not obvious that context causality between labelled events should become structural causality after composition. But this is precisely what Proposition 5.1 says. We designed our semantics so that Theorem 5.6 is true, and the simplicity of the proof is just a consequence of this choice.

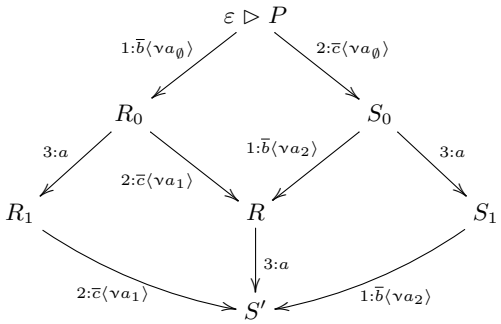
Example 5.2: Consider the π-calculus process $P = \nu a(\overline{b}\langle a\rangle \mid \overline{c}\langle a\rangle \mid a)$ with the trace $P \xrightarrow{\overline{b}\langle \nu a\rangle} \xrightarrow{\overline{c}\langle \nu a\rangle} \xrightarrow{a} Q$. A reduction context for P is $C[\bullet] = ([\bullet] \mid b(u).\overline{u} \mid c(v).\overline{v})$ and we have $C[P] \xrightarrow{\tau_b} \xrightarrow{\tau_c} \xrightarrow{\tau_a} Q'$. Remark that the transitions τ_b and τ_c are concurrent and that we can interchange them in the trace.

The last synchronisation on channel a corresponds to two different events: one engendered by the substitution on u and another by the substitution on v. In Rπ, the corresponding events choose as cause the transition on b and on c respectively. We can represent all the commutative transitions for P as follows:



Note that from process R, depending on the choice of contextual cause (either event 1 or 2) we can reach two distinct processes, that allow different backward paths from them. Previous permutation equivalence for the π-calculus in the literature would not allow the top diagram with source $\varepsilon \rhd P$ and target R to commute.

An alternative approach could have been not to choose a context cause, when many are available:



However, this would be conceptually incorrect, as it would allow the trace $\sigma : \varepsilon \rhd P \xrightarrow{+} R_0 \xrightarrow{+} R_1 \xrightarrow{+} S' \xrightarrow{-} S_1$ which would break the fact that in any reduction context of σ the first reduction is always a cause of the second (and therefore should not be backtracked first).

VI. Conclusion

We have presented the first labelled reversible semantics for the π-calculus. As reversibility is linked to causality, we also provide a novel causal semantics for the π-calculus. We have argued that our notion of causality is canonical, as entirely sound with respect to the precedence operator between reductions.

In our presentation, we have omitted the replication and the choice operator. We believe including these would be a straightforward import from the technique used in RCCS [21], to the price of a more involved syntax for terms.

The causal semantics of π-calculus we defined guarantees that events have each a unique causal history. As mentioned, this is not the case for the event structure semantics presented in [12]. We plan to find a way to lift the semantics presented here to event structures.

Another interesting continuation of our work consists in reformulating the correctness criteria of our permutation equivalence, in terms of a Galois connection between the (concrete) world of reductions and the (abstract) one of labelled transitions, taking inspiration, for instance, from the approach of Feret [22].

Lastly, we are also interested in studying meaningful equivalence relations for reversible processes. A notion of *forward-reverse* bisimulation can be defined, which coincides with Hereditary History preserving bisimulation [8]. These bisimulations are however quite discriminative and do not abstract away from internal actions. The weak bisimulation, usually employed in the π-calculus, is no longer useful in a reversible context since a reversible process is weakly bisimilar to any of its derivatives [9]. The problem of finding interesting weak equivalences between reversible systems is still an open issue and we hope that our labelled semantics can be used as a good starting point.

References

[1] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, pp. 183–191, 1961.

[2] L. Bougé, "On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes," *Acta Inf.*, vol. 25, no. 2, pp. 179–201, 1988.

[3] C. Palamidessi, "Comparing the expressive power of the synchronous and asynchronous pi-calculi," *Mathematical Structures in Computer Science*, vol. 13, no. 5, pp. 685–719, 2003.

[4] R. D. Nicola, U. Montanari, and F. Vaandrager, "Back and forth bisimulations," in *Proceedings of CONCUR'90*, ser. LNCS, vol. 458, 1990, pp. 152–165.

[5] V. Danos and J. Krivine, "Transactions in RCCS," in *In Proc. of CONCUR, LNCS 3653*. Springer, 2005, pp. 398–412.

[6] ——, "Reversible communicating systems," in *Proceedings of 15th CONCUR*, ser. Lecture Notes in Computer Science, vol. 3170. Springer, 2004, pp. 292–307.

[7] I. Phillips and I. Ulidowski, "Reversing algebraic process calculi," in *Proceedings of FOSSAC'06*, ser. LNCS, vol. 3921, 2006, pp. 246–260.

[8] ——, "Reversibility and models for concurrency," *Electr. Notes Theor. Comput. Sci.*, vol. 192, no. 1, pp. 93–108, 2007, proceedings of SOS 2007.

[9] I. Lanese, C. A. Mezzina, and J.-B. Stefani, "Reversing higher-order pi," in *Proceedings of 21st CONCUR*, ser. Lecture Notes in Computer Science, vol. 6269. Springer, 2010, pp. 478–493.

[10] M. Boreale and D. Sangiorgi, "A fully abstract semantics for causality in the π-calculus," *Acta Inf.*, vol. 35, no. 5, pp. 353–400, 1998.

[11] P. Degano and C. Priami, "Non-interleaving semantics for mobile processes." *Theor. Comp. Sci.*, vol. 216, no. 1-2, pp. 237–270, 1999.

[12] S. Crafa, D. Varacca, and N. Yoshida, "Event structure semantics of the parallel extrusion in the pi -calculus," in *Proceedings of 15TH FOSSACS*, ser. Lecture Notes in Computer Science, vol. 7213. Springer, 2012, pp. 225–239.

[13] D. Sangiorgi and D. Walker, *The π-calculus*. Cambridge University Press, 2002.

[14] J.-J. Lévy, "Réduction optimales en λ-calcul," Ph.D. dissertation, Université Paris 7, 1978.

[15] G. Boudol and I. Castellani, "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*, ser. LNCS, vol. 354, 1989, pp. 411–427.

[16] P. Darondeau and P. Degano, "Causal trees," in *Proceedings of ICALP'89*, ser. LNCS, vol. 372, 1989, pp. 234–248.

[17] G. Boudol, I. Castellani, M. Hennesy, and A. Kiehn, "A theory of processes with localities," *Formal Aspect of Computing*, 1992.

[18] I. Castellani, "Observing distribution in processes: Static and dynamic localities," *International Journal of Foundations of Computer Science*, vol. 6, no. 4, pp. 353–393, 1995. [Online]. Available: citeseer.ist.psu.edu/castellani94observing.html

[19] J. J. Leifer and R. Milner, "Deriving bisimulation congruences for reactive systems," in *Proceedings of CONCUR*, ser. Lecture Notes in Computer Science, vol. 1877. Springer, 2000.

[20] P. D. Gianantonio, F. Honsell, and M. Lenisa, "Finitely branching labelled transition systems from reaction semantics for process calculi," in *WADT*, ser. Lecture Notes in Computer Science, A. Corradini and U. Montanari, Eds., vol. 5486. Springer, 2008, pp. 119–134.

[21] J. Krivine, "Algèbres de processus réversibles," Ph.D. dissertation, Université Paris 6 & INRIA-Rocquencourt, 2006.

[22] J. Feret, "Confidentiality analysis of mobile systems," in *Seventh International Static Analysis Symposium (SAS'00)*, ser. LNCS, no. 1824. Springer-Verlag, 2000.