UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

CAUSALITY AND REPLICATION IN CONCURRENT PROCESSES

Pierpaolo Degano, Fabio Gadducci and Corrado Priami

May 2003

Technical Report # DIT-03-021

Also: to appear in the Proc. of Andrei Ershov Fifth International Conference "Perspectives of System Informatics"

Causality and replication in concurrent processes*

Pierpaolo Degano[1], Fabio Gadducci[1], and Corrado Priami[2]

[1] Dept. of Informatics, Univ. of Pisa, via F. Buonarroti 2, 56127 Pisa, Italy
{degano, gadducci}@di.unipi.it
[2] Dept. of Inf. & Telecomm., Univ. of Trento, via Sommarive 14, 38050 Povo, Italy
priami@science.unitn.it

Abstract. The replication operator was introduced by Milner for obtaining a simplified description of recursive processes. The standard interleaving semantics denotes the replication of a process P, written $!P$, a shorthand for its unbound parallel composition, operationally equivalent to the process $P \mid P \mid \ldots$, with P repeated as many times as needed.
Albeit the replication mechanism has become increasingly popular, investigations on its causal semantics has been scarce. In fact, the correspondence between replication and unbound parallelism makes it difficult to recover basic properties usually associated with these semantics, such as the so-called *concurrency diamond*.
In this paper we consider the interleaving semantics for the operator proposed by Sangiorgi and Walker, and we show how to refine it in order to capture causality. Furthermore, we prove it coincident with the standard causal semantics for recursive process studied in the literature, for processes defined by means of constant invocations.

Keywords: Causal semantics, process calculi, replication operator.

1 Introduction

The replication operator offers a suitable tool for the description of distributed systems with resources that can serve iterated requests. For instance, the handler of a printer can be seen as a process which replicates itself any time a request arrives. Even if there is a single printer, a copy of it handles the current request, while another is willing to accept new requests. The syntax is $!P$ (*bang* P), intuitively meaning $P \mid P \mid \ldots$ as many times as needed.

According to its operational semantics, the execution of an action from a process $!P$ results in $!P \mid Q$, where Q is the residual of a finite process $P \mid \ldots \mid P$ after the action, and the new $!P$ is willing to generate other copies of P if needed. For this reason, quite often $!P$ is postulated equivalent to $!P \mid P$.

* This work has been partially supported by the Italian MIUR project COMETA (*Computational Metamodels*); and by EU within the FET – Global Computing initiative, project DEGAS IST-2001-32072 (*Design Environments for Global Applications*).

The replication mechanism was not the original proposal for the handling of potentially infinite behaviour in process calculi. Standard specifications are based either on recursion operators or on constant definitions, on the form $A = P$, with A possibly occurring in P. In [12], Milner showed how to translate a process built up with constant definitions into a process containing only occurrences of the replication operator, at the price of an extra synchronisation that has to be ignored to preserve the semantics of processes.

The late introduction of the replication operator may explain why, to the best of our knowledge, investigations on its causal semantics are scarce in the literature. We are only aware of the work by Engelfriet (see e.g. [9]), where a Petri nets semantics for the π-calculus with replication is proposed, using possibly infinite markings. Such a state of affairs is unfortunate, since causality is often advocated to sharpen the description of distributed systems, because it may lead to more accurate guidelines to implementors and it offers a more precise analysis. In fact, causal semantics for concurrent systems in the setting of process calculi has been largely presented in the literature, see e.g. [1–4, 6, 7, 10, 14]. It is remarkable that for CCS, maybe the best known calculus [11], all of them do agree, so either description is *the* causal semantics of the calculus.

Our goal here is to define a causal semantics for CCS extended with replication that agrees with the standard causal semantics of the calculus, in which constant definition is adopted for the specification of recursive processes. In order to accomplish that, we find it difficult to stick to the usual operational presentation of the operator, resorting instead to a proposal in [15]. Then, we exploit the technique used in [7] to define non interleaving semantics for the π-calculus [13], and we extend it to include the case of the replication operator, considering, for the sake of presentation, only pure CCS [11]. The essence of the enhanced SOS semantics proposed in [7, 8] consists in encoding (portions of) the proof of a transition in its label. The resulting labeled transition system is thus called *proved*. Enhanced labels are sufficient to derive a causal semantics of the calculus.

In the next section we briefly define the proved operational semantics of CCS with replication. In Section 3 we adapt the definition of dependency between transitions given in [8] To cope with !, proving in Section 4 that two transitions are concurrent if and only if they form a so-called *concurrency diamond* in the proved transition system, i.e., roughly, if they can execute in any order. Instead, in Section 5 we show that the new notion of causality agrees with the one defined in the literature for processes built up using constant definitions.

2 Proved Transition System

We start introducing the syntax of CCS. As usual, we assume a countable set of *atomic actions*, denoted by $\mathcal{A}$ and ranged over by a; a bijective function $\bar{\cdot} : \mathcal{A} \to \mathcal{A}$, assuming that $a = \overline{\overline{a}}$; and an *invisible action* $\tau \notin \mathcal{A}$, so that $\mathcal{A} \cup \{\tau\}$ is ranged over by μ. Processes (denoted by $P, Q, R, \ldots \in \mathcal{P}$) are built from actions and agents according to the following syntax

$$P ::= \mathbf{0} \mid \mu.P \mid P + P \mid P|P \mid (\nu a)P \mid !P$$

We assume that the operators have decreasing binding power, in the following order: $(\nu a), \mu., !, |, +$. Hereafter, both the parallel composition and the nondeterministic choice are left associative. Also, we usually omit the trailing $\mathbf{0}$, whenever clear from the context. We slightly modified the original syntax in [11]: the replication operator ! replaces the recursion operators, and relabelling is omitted because irrelevant for the present study.

We assume the reader to be familiar with the interleaving operational semantics of CCS [11], and we concentrate on its proved semantics. We start enriching the labels of the standard transition system of CCS, in the style of [2, 4]. This additional structure encodes some information on the *derivation* of the transitions, that is, on the inference rules actually used to obtain that derivation.

We jointly define the notion of *enhanced labels* and two functions on them. So, ℓ takes an enhanced label to the corresponding action. Instead, ∂ takes a label to its "static" component: thus, it basically discards all the information on nondeterministic choices from the labels, whilst replacing the ! operator with the occurrence of a $||_1$ operator (see also Theorem 1).

Definition 1. *Let ϑ range over the strings in $\{||_0, ||_1, +_0, +_1, !\}^*$. Then, the* enhanced labels *(denoted by metavariable θ) are defined by the following syntax*

$$\theta ::= \vartheta\mu \mid \vartheta\langle ||_0\vartheta_0 a, ||_1\vartheta_1\overline{a}\rangle$$

The function ℓ is defined as

$$\ell(\vartheta\mu) = \mu \qquad \ell(\vartheta\langle ||_0\vartheta_0 a, ||_1\vartheta_1\overline{a}\rangle) = \tau$$

The function ∂ is defined as (for $i = 0, 1$)

$$\partial(\mu) = \mu \qquad \partial(+_i\theta) = \partial(\theta) \qquad \partial(!\theta) = ||_1\partial(\theta)$$
$$\partial(||_i\theta) = ||_i\partial(\theta) \qquad \partial(\langle ||_0\vartheta_0 a, ||_1\vartheta_1\overline{a}\rangle) = \langle\partial(||_0\vartheta_0 a), \partial(||_1\vartheta_1\overline{a})\rangle$$

By abuse of notation we will sometimes write $\partial(\vartheta)$ for $\partial(\vartheta\mu)$.

The rules of the proved transition system for CCS are in Table 1, where we omitted the symmetric Par_1 and Sum_1. As it will be made precise later on by Definition 2, the leftmost tag of a transition is in correspondence with the top-level operator of its source process (except for (νa)). Accordingly, rule Par_0 (Par_1) adds to the label a tag $||_0$ ($||_1$) to record that the left (right) component is moving. Similarly for the rules Sum_i.[1] The rule Com has in its conclusion a pair instead of a τ to record the components which interact. The rule $Bang$ for replication has in its premise a transition, tagged by θ, from its "body" P to a target P'. This rule adds in its conclusion the tag ! to the label of the transition, that now exits from $!P$ and reaches a state where $!P$ itself is put in parallel with P'; similarly for $Bang_C$, where two copies of P communicate with each other.

The interleaving transition system is derived from the proved one by relabelling each proved transition through function ℓ in Definition 1.

[1] This tag was not considered in [7], and it is really needed only in Definition 6 below.

$$Act: \frac{-}{\mu.P \xrightarrow{\mu} P} \qquad Sum_0: \frac{P \xrightarrow{\theta} P'}{P+Q \xrightarrow{+_0\theta} P'} \qquad Par_0: \frac{P \xrightarrow{\theta} P'}{P|Q \xrightarrow{||_0\theta} P'|Q}$$

$$Bang: \frac{P \xrightarrow{\theta} P'}{!P \xrightarrow{!\theta} !P|P'} \qquad Bang_C: \frac{P \xrightarrow{\vartheta_0 a} P', P \xrightarrow{\vartheta_1 \overline{a}} P''}{!P \xrightarrow{!\langle||_0\vartheta_0 a, ||_1\vartheta_1\overline{a}\rangle} !P|(P'|P'')}$$

$$Com: \frac{P \xrightarrow{\vartheta_0 a} P', Q \xrightarrow{\vartheta_1 \overline{a}} Q'}{P|Q \xrightarrow{\langle||_0\vartheta_0 a, ||_1\vartheta_1\overline{a}\rangle} P'|Q'} \qquad Res: \frac{P \xrightarrow{\theta} P'}{(\nu a)P \xrightarrow{\theta} (\nu a)P'} \quad \ell(\theta) \notin \{a, \overline{a}\}$$

Table 1: The proved transition system for CCS.

Let us provide now a simple example of a proved computation. Consider the process $P = !(a|!b)$. The first four steps of a computation originating from P are indicated by the sequence of transitions below

$$P \xrightarrow{!||_0 a} P|(\mathbf{0}|!b) \xrightarrow{||_1||_1!b} P|(\mathbf{0}|(!b|\mathbf{0})) \xrightarrow{||_0!||_1!b} \qquad (1)$$
$$(P|(a|(!b|\mathbf{0}))) \mid (\mathbf{0}|(!b|\mathbf{0})) \xrightarrow{||_1||_1||_0!b} \ldots$$

A few comments are in order now, to vindicate our choice of inference rules for the replication operator. Our proposal follows [15], and its rational is that making a copy of a process is an activity in itself, even if hidden, that models the invocation of a run-time support routine. The actions performed by subsequent copies will then be caused by the first activation. (Of course, according to our semantics, the process $!(a|b)$ can perform two causally independent actions; each of them, in turn, causes any further subsequent action.) The standard rule

$$\frac{!P|P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$$

would allow for a recursive activation of a copy of P at any depth. This solution would make it difficult to obtain any result concerning causality in our approach, which is based on enhanced labels, hence it relies on proofs by structural induction over the entailment of the possible derivations of a transition.

In the next sections, our informal argument in favour of the chosen semantics for the replication operator will also be supported by a correspondence with the causal semantics of recursive processes (Section 5) and by mimicking the standard results on concurrent processes, as typically exemplified by the occurrence of concurrency diamonds (Section 4).

3 Causality

We introduce now the notion of causal dependency. Following [7], an auxiliary relation of dependency is introduced on the transitions that may occur along a computation, making explicit the causality between their actions. From this relation, it is straightforward to recover the more standard representation of causality as a partial ordering of events, and that of concurrency (see [8]). Both notions do coincide with those defined in the literature (see, e.g., [2, 3]).

Definition 2 (dependency relation). *The* dependency *relation on enhanced labels is the relation induced by the axioms below (for $i \in \{0,1\}$)*

1. $\mu \lessdot \theta$
2. $+_i\theta \lessdot \theta'$ *if* $\theta \lessdot \theta'$
3. $||_i\theta \lessdot ||_i\theta'$ *if* $\theta \lessdot \theta'$
4. $\langle\theta_0,\theta_1\rangle \lessdot \langle\theta'_0,\theta'_1\rangle$ *if* $\exists i\,.\,\theta_i \lessdot \theta'_i$
5. $\langle\theta_0,\theta_1\rangle \lessdot \theta'$ *if* $\exists i\,.\,\theta_i \lessdot \theta'$
6. $\theta \lessdot \langle\theta'_0,\theta'_1\rangle$ *if* $\exists i\,.\,\theta \lessdot \theta'_i$
7. $!\theta \lessdot ||_0\theta'$
8. $!\theta \lessdot ||_1\theta'$ *if* $\theta \lessdot \theta'$

Intuitively, each rule indicates that a dependency between the labels may occur. More precisely, $\theta_0 \lessdot \theta_1$ implies that if there exists a computation in which an occurrence of label θ_0 takes place before an occurrence of label θ_1, then the two associated transitions are causally related. For example, consider the computation $a.b.a.\mathbf{0} \xrightarrow{a} b.a.\mathbf{0} \xrightarrow{b} a.\mathbf{0} \xrightarrow{a} \mathbf{0}$. Now, the occurrence of the first a causes the transition labeled b, which in turns causes the second instance of a.[2]

So, the relation has to be specialised for each computation. From now onward, unless otherwise specified, ξ denotes a generic proved computation of the form $P_0 \xrightarrow{\theta_0} P_1 \xrightarrow{\theta_1} \ldots \xrightarrow{\theta_n} P_{n+1}$, whose *length* is n; furthermore, each label θ_i actually stands for the pair $\langle\theta_i, i\rangle$, thus uniquely identifying a single position of the label in the computation, hence the associated transition.

Definition 3 (causality partial order). *Let ξ be a computation, and let $i, j \in \{1,\ldots,n\}$ with $i < j$. Then, we say that θ_i* causes *θ_j in ξ (in symbols $\theta_i \lessdot_\xi \theta_j$) iff $\theta_i \lessdot \theta_j$.*

The relation $\preceq_\xi$ (or simply $\preceq$ when unambiguous) of causal *dependency is the reflexive and transitive closure of $\lessdot_\xi$, further restricted over the transitions θ with $\ell(\theta) \neq \tau$.*

The causality dependency is obtained for a computation ξ by first restricting $\lessdot$ to the transitions of ξ according to their occurrence in the computation, thus obtaining $\lessdot_\xi$; and then freely closing $\lessdot_\xi$ by reflexivity and transitivity. The resulting relation is a partial order, since antisymmetry is immediate by the condition $i < j$. Finally, the partial order is restricted to those transitions with visible actions, getting $\preceq_\xi$.

[2] In fact, the relation ensures that such a causal dependency holds for at least a computation, but it is neither reflexive, nor transitive. For instance, label $+_0||_0a$ never occurs twice in a computation; and $||_0b$ and $||_1c$ are not dependent, even if $||_0b \lessdot \langle||_0a, ||_1\overline{a}\rangle$ and $\langle||_0a, ||_1\overline{a}\rangle \lessdot ||_1c$, and the three labels occur exactly in that order in the unique computation starting from $(\nu a)(b.a|\overline{a}.c)$.

The intuitive idea behind $\theta_i \preceq_\xi \theta_j$ is that θ_i is a *necessary condition* in ξ for θ_j to occur. Such a condition is discovered on the enhanced labels by exploiting a prefix relation between them. Indeed, consider two transitions which are labeled ϑa and $\vartheta\vartheta' b$. Then, the action prefixes a and $\vartheta' b$ which originate the given transitions are within the same context. The dependency relation implies that two transitions have been derived by using the same initial set of inference rules, and this is verified when a and b are nested in the *same* prefix chain. Thus, b occurs in the process prefixed by a, and the occurrence of a is a necessary conditions for b to occur.

More precisely, our definition of dependency relation inductively scans the common prefix of two labels. Item (1) of Definition 2 is one base case. It says that the action at the left-hand side was prefixed to the action at the right-hand side; so we postulate a causal dependency between the two labels, in the style of [7]. The other base case is in item (7). Any transition with a label starting with ! causes those with labels starting with a $||_0$. Indeed, the transition corresponding to the label on the right-hand side is fired by a replica of the !-process that fired the left-hand transition. The intuition is that a bang application generates a copy of itself and thus causes any subsequent action of that copy. Technically, subsequent applications of the same bang introduce on the labels of the corresponding transitions tags $||_0!$, .., $||_0^k!$, where $||_0^k$ is a string of k tags $||_0$. Indeed, the second application of a bang is in the left component of a parallel composition (see the conclusion of the rules for bang in Table 1). Instead, if the right-hand label starts with a $||_1$, we continue scanning labels, as item (8) says. This case corresponds to having the transition with the label on the right being originated by an action of the residual process after the original ! activation. Item (2) reflects the fact that a choice has been resolved in the transition occurring first, and therefore the "+" will no longer appear in its derivatives. In item (3), labels are scanned as long as they have the same initial tag ($||_i$). As far as synchronisations are concerned, item (4) applies the definition component-wise, while items (5) and (6) compare visible actions with the components of the synchronisation.

Consider again the computation (1) in Section 2. It turns out that $!||_0 a$ and $||_1||_1!b$ are not related, whilst $!||_0 a \preceq ||_0!||_1!b$ by item (4) in Definition 2. It holds also $||_1||_1!b \preceq ||_1||_1||_0!b$ by two applications of item (2) and one application of item (4) in Definition 2.

4 Concurrency

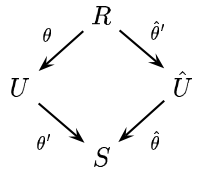
We now introduce the notion of *concurrency* between the transitions of a computation. It is simply the complement of causality as in [7].

Definition 4 (concurrency relation). *Let ξ be a computation, and let $i, j \in \{1, \ldots, n\}$ with $\tau \notin \{\ell(\theta_i), \ell(\theta_j)\}$. Then, we say that θ_i is* concurrent *with θ_j (in symbols $\theta_i \smile_\xi \theta_j$, or simply $\theta_i \smile \theta_j$ when unambiguous) iff neither $\theta_i \preceq_\xi \theta_j$ nor $\theta_j \preceq_\xi \theta_i$.*

The definition of concurrency says that no necessary condition exists between two transitions such that $\theta_i \smile \theta_j$. This leads to the more usual statement that the execution ordering of concurrent transitions is irrelevant.

The theorem below extends a classical property of concurrency to CCS processes including replication. It states that two concurrent transitions, originated by the same process P, can be fired one before the other *and* vice versa, thus forming a so-called concurrency diamond in the transition system.

Theorem 1. *Let $R \xrightarrow{\theta} U \xrightarrow{\theta'} S$ be a computation such that $\theta \smile \theta'$. Then, in the proved transition system there exists the concurrency diamond below, where $\partial(\theta) = \partial(\hat{\theta})$ and $\partial(\theta') = \partial(\hat{\theta}')$.*



Consider again the process $P = !(a|!b)$. Its computation

$$P \xrightarrow{!||_1!b} P|(a|(!b|\mathbf{0})) \xrightarrow{||_1||_0 a} P|(\mathbf{0}|(!b|\mathbf{0}))$$

originates a diamond with the first two transitions in (1), our example in Section 2. Note that the leftmost ! in the first transition above becomes a $||_1$ in the second transition of (1), and the ! in the first transition of (1) becomes $||_1$ in the second transition above. Our function ∂ handles the situation.

In fact, the main problem in defining a causal semantics for the replication operator was linked to obtaining a concurrency diamond property like the above. Our solution, based on the dependency relation in Definition 2, clearly states the dual nature of the bang: it behaves as a prefix $||_1$ when observed, even if it imposes a dependency on subsequent activations of its copies.

Given two concurrent transitions along a computation, it would be possible to prove, by repeated applications of the theorem above, the existence of another computation with the same source and the same target, in which those two transitions occur one after the other and in reverse order. This result establishes a classical property of causal semantics, often expressed by saying that "concurrent transitions may fire in any temporal order". Indeed, a *concurrent computation* could then be defined as the equivalence class of those computations that only differ in the order in which concurrent transitions occur, disregarding the actual identity of the processes involved.[3]

[3] The relation of "reachability by application of permutations" is clearly reflexive and transitive. In order to prove symmetry, a stronger version of Theorem 1 is needed, stating that a permutation gives back the initial computation when applied twice.

5 Causality for replication and recursion

We now compare our notion of causality with the others presented in the literature. More precisely, we show that the relation $\preceq$ coincides with the one in [6, 7], that has been proved to coincide with the causal relation defined in [3], hence accounting for *the* causal semantics of the calculus, as argued in the Introduction.

First, some definitions. Given a set of *process constants*, ranged over by $A, B, \ldots$, an environment $\mathcal{E}$ is a set of equations $A_i = P_i$, for disjoint constants A_i's and CCS processes P_i's. Those processes may possibly contain some occurrences of the constants, but no occurrence of the ! operator. The rule for constants is given by

$$\frac{P \xrightarrow{\theta}_{\mathcal{E}} P'}{A \xrightarrow{\theta}_{\mathcal{E}} P'} \quad A = P \in \mathcal{E}$$

and it is dependent from the chosen environment.

We let $\sqsubseteq^{\mathcal{E}}$ denote the causal dependency that is obtained (according to Definition 2 and Definition 3) on computations originating from processes with respect to a given environment $\mathcal{E}$, and with the inference rule presented above: it corresponds to the causality partial order given in [7], Definition 4.1.

We shall exploit the following translation of constant definitions into processes involving the ! operator proposed by Milner in [12] (see also [15]). We assume hereafter to consider environments with just one equation, the extension to the general case being rather straightforward.

Definition 5 (from constants to replication). *Let $\mathcal{E}$ be the environment defined by the equation $A = P$, and let a be an action not occurring in P. Then, the function $\phi_{\mathcal{E}}$, that translates processes possibly containing occurrences of the constant A into processes without them, is defined as*

$$\phi_{\mathcal{E}}(R) = R\{(\nu\, a)(\overline{a} \mid !a.P\{\overline{a}/A\})/A\}.$$

The process $(\nu\, a)(\overline{a} \mid !a.P\{\overline{a}/A\})$ replaces every occurrence of the constant A, insisting that a is a fresh action not occurring in P. The substitution of $\overline{a}$ for A in its body P is needed to activate other copies of P, possibly available later on. Indeed, the implementation performs a τ-transition (called below !-synchronisation) involving the $!a.P$ component of the translation, that enables a new instance of P. On the other hand, no τ occurs in the constant activation, so that the discrepancy has to be taken into account.

In the following, we will just fix an environment $\mathcal{E}$, composed by a single equation on the constant A, dropping all the subscripts referring to it.

Before comparing our causal relation $\preceq$ on the computations of R with the relation $\sqsubseteq$ on the computations of $\phi(R)$, we shall have a closer look at the labels of the transitions in corresponding computations. We start with the definition of the selector operator, $@\vartheta$, that applied to a process R singles out its subprocess P reachable at position ϑ in the abstract syntax tree (with leaves $\mu.Q$ or $\mathbf{0}$) of R, and it is not defined otherwise. It will be used to select within R its subprocess responsible for a transition $R \xrightarrow{\vartheta\mu} R'$.

Definition 6 (selectors). *Let P be a process and $\vartheta \in \{||_0, ||_1, +_0, +_1, !\}^*$. Then $P@\vartheta$ is defined by (for $i = 0, 1$)*

$$P@\epsilon = P \qquad (\nu\, a)P@\vartheta = P@\vartheta \qquad !P@!\vartheta = P@\vartheta$$
$$(P_0 \mid P_1)@||_i\vartheta = P_i@\vartheta \qquad (P_0 \ + \ P_1)@ +_i \vartheta = P_i@\vartheta$$

We now compare a computation of the process R with the corresponding computation of $\phi(R)$. Intuitively, the transitions not originated by a constant invocation have the same labels in both. So, R and its translation $\phi(R)$ pass through the same states, up to the substitution of the occurrences of a constant with its translation, until the first invocation of a constant is fired. At that point, $\phi(R)$ performs a !-synchronisation that enables the first action of (the body of) the activated constant. The !-synchronisation introduces a tag $||_1$ followed by a ! in its label: the tag ! is added by the application of rule *Bang* that creates a new copy of $!a.P\{\overline{a}/A\}$. This copy is responsible for a $||_1$ tag in the labels of the transitions fired from the right component in the conclusion of that rule. The other transitions fired by (the body of) the constant have an additional tag $||_1$, due to the context $\overline{a}|-$ responsible for the !-synchronisation in $\phi(R)$. In the proposition below we shall point out this additional !-synchronisation and its target state, S_i, that has no corresponding state in the selected computation of R. Afterward, the two processes R and $\phi(R)$ evolve similarly until the next constant invocation, and if their transitions have different enhanced labels, this is only because of the additional tags $||_1||_1$ mentioned above.

Consider the process $c.A \mid R$, where $A = b.A \mid d$, which can evolve as

$$c.A \mid R \xrightarrow{||_0 c} A \mid R \xrightarrow{||_0||_0 b} (A|d) \mid R$$

Its translation is the process $c.(\nu\, a)(\overline{a}|!a.(b.\overline{a}|d)) \mid R$, and the computation corresponding to the one we have just seen is

$$c.(\nu\, a)(\overline{a}|!a.(b.\overline{a}|d)) \mid R \xrightarrow{||_0 c} (\nu\, a)(\overline{a}|!a.(b.\overline{a}|d)) \mid R \xrightarrow{||_0\langle||_0\overline{a},||_1!a\rangle}$$
$$(\nu\, a)(\mathbf{0}|(!a.(b.\overline{a}|d))|(b.\overline{a}|d)) \mid R \xrightarrow{||_0||_1||_1||_0 b} (\nu\, a)(\mathbf{0}|(!a.(b.\overline{a}|d))|(\overline{a}|d)) \mid R$$

In the last transition of the translation we can see where the additional tags $||_1||_1$ are needed. Note also that in the last transition of both computations, the leftmost $||_0$ corresponds to the context of A and of its translation, while the rightmost $||_0$ encodes the context of b within the body of A.

The next proposition formalises the considerations stated above. For the sake of simplicity, we restrict our attention to computations with a single constant activation, and with a visible action. Our argument could be easily repeated for subsequent activations, possibly occurring inside a communication.

Proposition 1. *Let $\mathcal{E}$ be the environment defined by the equation $A = P$, and let ξ be the computation*

$$R_0 \xrightarrow{\theta_0} R_1 \xrightarrow{\theta_1} \dots R_i \xrightarrow{\vartheta\theta_i} R_{i+1} \dots \xrightarrow{\theta_n} R_{n+1}$$

in which $\vartheta\theta_i$ is the only invocation of the constant, and such that $R_i@\vartheta = A$. Then, $\phi(R_0)$ may perform the computation $\phi(\xi)$ below

$$\phi(R_0) \xrightarrow{\theta_0} \phi(R_1) \xrightarrow{\theta_1} \dots \phi(R_i) \xrightarrow{\vartheta\langle||_0\overline{a},||_1!a\rangle} S_i \xrightarrow{\phi(\theta_i)} S_{i+1} \dots \xrightarrow{\phi(\theta_n)} S_{n+1}$$

where

$$\begin{array}{l} S_i = \phi(R_{i+1})[\partial(\vartheta) \mapsto Q] \\ S_j = \phi(R_j)[\partial(\vartheta) \mapsto Q_j] \quad \textit{for } j = i+1 \dots n+1 \\ \phi(\theta_i) = \partial(\vartheta)||_1||_1\theta_i \\ \phi(\theta_j) = \begin{cases} \partial(\vartheta)||_1||_1\theta & \textit{if } \theta_j = \partial(\vartheta)\theta \\ \theta_j & \textit{otherwise} \end{cases} \quad \textit{for } j = i+1 \dots n \end{array}$$

(for $R[\vartheta_R \mapsto S]$ denoting the process obtained from R by replacing the sub-process $R@\vartheta_R$ with the process S), and

$$\begin{array}{l} Q = (\nu\, a)(\mathbf{0} \mid (!a.P\{\overline{a}/A\} \mid P\{\overline{a}/A\})) \\ Q_j = Q[||_1||_1 \mapsto R_j\{\overline{a}/A\}@\partial(\vartheta)] \quad \textit{for } j = i+1 \dots n+1 \end{array}$$

Hence, we end up with a function ϕ between computations with a different presentation for recursive processes (the source may possibly contain constants, the target the replication operator), which also relates in a precise way the single transitions. This allows for showing that the causal relation $\preceq$ coincides with the relation $\sqsubseteq$ described in the literature, over computations that are related via ϕ.

Theorem 2. *Let ξ be a computation originated by a finite* CCS *term, possibly involving some constant invocation. Then, $\sqsubseteq_\xi$ coincides with $\preceq_{\phi(\xi)}$, in the sense that for any two transitions θ_i, θ_j in ξ, $\theta_i \sqsubseteq_\xi \theta_j$ if and only if $\phi(\theta_i) \preceq_{\phi(\xi)} \phi(\theta_j)$.*

In other words, $\sqsubseteq_\xi$ and $\preceq_{\phi(\xi)}$ represent the same partial order of transitions, up to the relabelling induced by ϕ. In fact, the two partial orders are actually isomorphic, since all the transitions in $\phi(\xi)$ which are not image of a transition in ξ are communications.

6 Conclusions

We extended the results in [7] showing how proved transition systems may help in defining non interleaving semantics of calculi including the replication operator. Our causal semantics for CCS with replication enjoys the classical "concurrency diamond" property (concerning the execution order of independent transitions), and it coincides with the usual semantics for CCS with constant definition.

Our result extends a previous attempt concerning the causal semantics of processes with an operator for *replicated input*, denoted $!_a$, such that the process $!_a P$ may perform a move labeled a, becoming $!_a P \mid P$ [5]. In fact, our proved semantics for the replication operator is based on the intuition that making a copy of a process requires to invoke a run-time support routine, and thus subsequent copies will be causally linked. We believe that our proposal follows the spirit of Milner's implementation of recursive processes *via* replication, since replicated inputs do suffice for it. See also our discussion on the inference rules for replication in Section 2.

Our proposal seems robust: no problem arises when dealing with name passing, because the replication operator only induces structural dependencies on transitions. Hence, a causal semantics for the π-calculus with ! can be defined, carrying over the present definitions the relevant ones of [7].

References

1. M. Boreale and D. Sangiorgi. A fully abstract semantics for causality in the π-calculus. *Acta Informatica*, 35:353-400, 1998.
2. G. Boudol and I. Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fundamenta Informaticae*, 11:433–452, 1988.
3. Ph. Darondeau and P. Degano. Causal trees. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Automata, Languages and Programming*, volume 372 of *Lect. Notes in Comp. Science*, pages 234–248. Springer, 1989.
4. P. Degano, R. De Nicola, and U. Montanari. Partial ordering derivations for CCS. In L. Budach, editor, *Fundamentals of Computation Theory*, volume 199 of *Lect. Notes in Comp. Science*, pages 520–533. Springer, 1985.
5. P. Degano, F. Gadducci, and C. Priami. A concurrent semantics for CCS via rewriting logic. *Theoret. Comput. Sci.*, 275:259–282, 2002.
6. P. Degano and C. Priami. Proved trees. In W. Kuich, editor, *Automata, Languages and Programming*, volume 623 of *Lect. Notes in Comp. Science*, pages 629–640. Springer, 1992.
7. P. Degano and C. Priami. Non interleaving semantics for mobile processes. *Theoret. Comput. Sci.*, 216:237–270, 1999.
8. P. Degano and C. Priami. Enhanced operational semantics: A tool for describing and analysing concurrent systems. *ACM Computing Surveys*, 33:135–176, 2001.
9. J. Engelfriet. A multiset semantics for the pi-calculus with replication. *Theoret. Comput. Sci.*, 153:65–94, 1996.
10. A. Kiehn. Comparing causality and locality based equivalences. *Acta Informatica*, 31:697–718, 1994.
11. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
12. R. Milner. The polyadic π-calculus: A tutorial. In F.L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *Nato ASI Series F*, pages 203–246. Springer, 1993.
13. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. Part I and II. *Information and Computation*, 100:1–77, 1992.
14. D. Sangiorgi. Locality and interleaving semantics in calculi for mobile processes. *Theoret. Comput. Sci.*, 155:39–83, 1996.
15. D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.