Causal Reversibility for Timed Process Calculi with Lazy/Eager Durationless Actions and Time Additivity

Marco Bernardo and Claudio A. Mezzina

Dipartimento di Scienze Pure e Applicate, Università di Urbino, Urbino, Italy

Abstract. A reversible computing system features backward computations along which the effects of forward ones are undone when needed. This is accomplished by reverting executed actions from the last one. Since the last performed action may not be uniquely identifiable in a concurrent system, causal reversibility is considered: an executed action can be undone provided that all of its consequences have been undone already. We investigate causal reversibility in a timed setting by defining a reversible calculus in the style of Phillips and Ulidowski in which action execution is separated from time passing, actions can be lazy or eager, and time is described via numeric delays subject to time additivity. We show that the calculus meets causal reversibility through an adaptation of the technique of Lanese, Phillips, and Ulidowski that ensures a proper treatment of action laziness/eagerness as well as time-additive delays.

1 Introduction

In the 60's it was observed that irreversible computations cause heat dissipation into circuits because any logically irreversible manipulation of information, such as bit erasure or computation path merging, must be accompanied by a corresponding entropy increase [21,3]. Therefore, any logically reversible computation, in which no information is canceled, may be potentially carried out without releasing any heat, as verified in [7] and given a physical foundation in [15]. This suggested that low energy consumption could be achieved by resorting to *reversible computing*, in which there is no information loss because computation can go not only forward, but also backward by undoing the effects of the forward direction when needed. Nowadays, reversible computing has several applications such as biochemical reaction modeling [33,34], parallel discrete-event simulation [31,37], robotics [26], control theory [38], fault-tolerant systems [13,40,22,39], and concurrent program debugging [16,24].

Reversibility in a computing system has to do with the possibility of reverting the last performed action. In a sequential computing system this is very simple because at each step only one action is executed, hence the only challenge is how to store the information needed to reverse that action. As far as concurrent and distributed systems are concerned, a critical aspect of reversibility is that there may not be a total order over executed actions, hence the last performed

action may not be uniquely indentifiable. This led to the introduction of the notion of *causal reversibility* [12], according to which a previously executed action can be undone provided that all of its consequences, if any, have been undone beforehand. It is worth noting that the concept of causality is used in place of the concept of time to decide whether an action can be undone or not.

In a pure nondeterministic process algebraic setting, two distinct approaches have been developed to deal with causal reversibility. The *dynamic* approach of [12,20], which is adequate for very expressive calculi and programming languages, attaches external stack-based memories to process terms so as to record all the executed actions. In contrast, the *static* approach of [32], which is convenient for handling labeled transition systems and basic process calculi, makes all process algebraic operators static – in particular action prefix and choice – so that they are kept within the derivative process term of any transition. The two approaches have been shown to be equivalent in terms of labeled transition system isomorphism [23] and the common properties they exploit to ensure causal reversibility have been systematically studied in [25].

When timed systems are considered, other notions of reversibility may come into play. This is the case with *time reversibility* for stochastic processes like continuous-time Markov chains [19]. It ensures that the stochastic behavior of a *shared-resource* computing system remains the same when the direction of time is reversed and is instrumental to develop performance evaluation methods that cope with state space explosion and numerical stability problems. In [5] causal reversibility and time reversibility have been jointly investigated in the setting of a Markovian process calculus defined according to the static approach of [32], where every action is extended with a positive real number quantifying the rate of the corresponding exponentially distributed random duration.

In this paper we address the reversibility of *real-time* computing systems. Unlike [5], time flows *deterministically*, instead of stochastically, and is described *orthogonally* to actions, i.e., through a specific delay prefix operator, instead of being integrated within actions. In the rich literature of deterministically timed process calculi – timed CSP [36], temporal CCS [29], timed CCS [41], real-time ACP [2], urgent LOTOS [9], CIPA [1], TPL [17], ATP [30], TIC [35], and PAFAS [11] – the differences are due to the following time-related options:

- *Durationless actions* versus *durational actions*. In the first case, actions are instantaneous events and time passes in between them; hence, functional behavior and time are *orthogonal*. In the second case, every action takes a certain amount of time to be performed and time passes only due to action execution; hence, functional behavior and time are *integrated*.
- *Relative time* versus *absolute time*. Assume that timestamps are associated with the events observed during system execution. In the first case, each timestamp refers to the time instant of the previous observation. In the second case, all timestamps refer to the starting time of the system execution.
- *Global clock* versus *local clocks*. In the first case, a single clock governs time passing. In the second case, several clocks associated with the various system parts elapse independently, although defining a unique notion of global time.

In addition, there are several different interpretations of action execution in terms of whether and when the execution can be delayed:

- *Eagerness*: actions must be performed as soon as they become enabled, i.e., without any delay, thereby implying that their execution is *urgent*.
- *Laziness*: after getting enabled, actions can be delayed arbitrarily long before they are executed.
- *Maximal progress*: enabled actions can be delayed unless they are independent of the external environment, in which case their execution is urgent.

The two major combinations of the aforementioned options yield the *two-phase functioning principle*, according to which actions are durationless, time is relative, and there is a single global clock, and the *one-phase functioning principle*, according to which actions are durational, time is absolute, and several local clocks are present. In [10] the two principles have been investigated under the various action execution interpretations through a bisimilarity-preserving encoding from the latter principle to the former, whilst the inverse encoding was provided in [4] along with a pair of encodings for the case of stochastic delays.

In this paper we focus on the two-phase functioning principle, yielding action transitions separated from delay transitions like in temporal CCS [29], and develop a reversible timed process calculus with lazy/eager actions, whose syntax and semantics adhere to the static approach of [32]. Then we show that the calculus is causally reversible through notions of [12] and the technique of [25]. The following adaptations turn out to be necessary with respect to [32,12,25]:

- Similar to executed actions, which are decorated with communication keys so as to remember who synchronized with whom when going backward [32], elapsed delays have to be decorated with keys to ensure that all subprocesses of an alternative or parallel composition go back in time in a well-paired way, as time cannot solve choices or decide which parallel process advances [29].
- The necessary condition for reversibility known as loop property [12,32,25], which establishes the presence of both a forward transition and a backward transition with the same label between any pair of connected states, has to deal with delay transitions too, in a way consistent with time additivity [29].
- Conflicting transitions, from which concurrent transitions [12] are then derived, and causal equivalence [12], which is needed to identify computations differing for the order of concurrent action transitions, have to be extended with additional conditions specific to delay transitions.
- Backward transitions independence, one of the properties studied in [25] to ensure causal reversibility, has to be paired with a novel property, which we have called backward triangularity, due to time additivity [29]. Furthermore, the semantic rules implementing laziness and maximal progress have to be carefully designed to guarantee the parabolic lemma [12,25].

This paper is organized as follows. In Section 2 we present the syntax and the semantics of our reversible timed process calculus. In Section 3 we prove that the calculus satisfies causal reversibility and illustrate some examples. In Section 4 we conclude with final remarks and directions for future work.

$$\begin{array}{|l|}\hline P,Q \;::=\; \underline{0} \mid a\,.\,P \mid (n)\,.\,P \mid P+Q \mid P \parallel_L Q \\ R,S \;::=\; P \mid a[i]\,.\,R \mid (n)^{[i]}.\,R \mid R+S \mid R \parallel_L S \\ \hline\end{array}$$

Table 1. Syntax of forward processes (top) and reversible processes (bottom)

2 Reversible Timed Process Calculus: Actions vs. Delays

In this section we present the syntax and the semantics of RTPC – Reversible Timed Process Calculus, which are inspired by temporal CCS [29] and tailored for a reversible setting according to the static approach of [32].

Given a countable set $\mathcal{A}$ of actions – ranged over by a, b – including an invisible or silent action denoted by τ, the syntax of RTPC is shown in Table 1. A standard *forward process* P is one of the following: the terminated process $\underline{0}$; the action-prefixed process $a\,.\,P$, which is able to perform an action a and then continues as process P; the delay-prefixed process $(n)\,.\,P$, which lets $n \in \mathbb{N}_{>0}$ time units pass and then continues as process P; the nondeterministic choice $P+Q$; or the parallel composition $P \parallel_L Q$, indicating that processes P and Q execute in parallel and must synchronize only on actions in $L \subseteq \mathcal{A} \setminus \{\tau\}$.

We assume *time determinism* [29], i.e., time passing cannot solve choices or decide which parallel process advances, hence the same amount of time must pass in all subprocesses of a nondeterministic choice or parallel composition. Under eagerness, the execution of all actions is urgent and $\underline{0}$ cannot let time pass. Under laziness, action execution can be delayed arbitrarily long and $\underline{0}$ lets time pass. Under maximal progress, only the execution of τ is urgent, because τ cannot be involved in synchronizations, and $\underline{0}$ lets time pass.

A *reversible process* R is built on top of forward processes. The syntax of reversible processes differs from the one of forward processes due to the fact that, in the former, action and delay prefixes are decorated. As in [32], each action prefix is decorated with a *communication key* i belonging to a countable set $\mathcal{K}$. A process of the form $a[i]\,.\,R$ expresses that in the past the process synchronized with the environment on a and this synchronization was identified by key i. Keys are thus attached only to executed actions and are necessary to remember who synchronized with whom when undoing actions; keys could be omitted in the absence of parallel composition. Similarly, $(n)^{[i]}.\,R$ means that n time units elapsed in the past. Here communication keys are needed to ensure time determinism in the backward direction, so that all subprocesses go back in time in the same way; keys could be omitted in the absence of choice and parallel composition.

We denote by $\mathcal{P}$ the set of processes generated by the production for R in Table 1, while we use predicate $\texttt{std}(\_)$ to identify the standard forward processes that can be derived from the production for P in the same table. For example, $a\,.\,(5)\,.\,b\,.\,\underline{0}$ is a standard forward process that can perform action a, let 5 time units pass, and then execute b, while $a[i]\,.\,(5)^{[j]}.\,b\,.\,\underline{0}$ is a non-standard reversible process that can either go back by 5 time units and undo action a, or perform action b. Note that $a\,.\,(5)^{[j]}.\,b\,.\,\underline{0}$ and $a\,.\,(5)\,.\,b[j]\,.\,\underline{0}$ are not in $\mathcal{P}$ because a future action or delay cannot precede a past one in the description of a process.

$$(\textsc{Act1})\ \frac{\mathtt{std}(R)}{a\,.\,R \xrightarrow{a[i]}_{\mathtt{a}} a[i]\,.\,R} \qquad (\textsc{Act1}^\bullet)\ \frac{\mathtt{std}(R)}{a[i]\,.\,R \overset{a[i]}{\dashrightarrow}_{\mathtt{a}} a\,.\,R}$$

$$(\textsc{Act2})\ \frac{R \xrightarrow{b[j]}_{\mathtt{a}} R' \quad j \neq i}{a[i]\,.\,R \xrightarrow{b[j]}_{\mathtt{a}} a[i]\,.\,R'} \qquad (\textsc{Act2}^\bullet)\ \frac{R \overset{b[j]}{\dashrightarrow}_{\mathtt{a}} R' \quad j \neq i}{a[i]\,.\,R \overset{b[j]}{\dashrightarrow}_{\mathtt{a}} a[i]\,.\,R'}$$

$$(\textsc{Act3})\ \frac{R \xrightarrow{b[j]}_{\mathtt{a}} R'}{\delta(n,i)\,.\,R \xrightarrow{b[j]}_{\mathtt{a}} \delta(n,i)\,.\,R'} \qquad (\textsc{Act3}^\bullet)\ \frac{R \overset{b[j]}{\dashrightarrow}_{\mathtt{a}} R'}{\delta(n,i)\,.\,R \overset{b[j]}{\dashrightarrow}_{\mathtt{a}} \delta(n,i)\,.\,R'}$$

$$(\textsc{Cho})\ \frac{R \xrightarrow{a[i]}_{\mathtt{a}} R' \quad \mathtt{npa}(S)}{R + S \xrightarrow{a[i]}_{\mathtt{a}} R' + S} \qquad (\textsc{Cho}^\bullet)\ \frac{R \overset{a[i]}{\dashrightarrow}_{\mathtt{a}} R' \quad \mathtt{npa}(S)}{R + S \overset{a[i]}{\dashrightarrow}_{\mathtt{a}} R' + S}$$

$$(\textsc{Par})\ \frac{R \xrightarrow{a[i]}_{\mathtt{a}} R' \quad a \notin L \quad i \notin \mathtt{keys}_{\mathtt{a}}(S)}{R \parallel_L S \xrightarrow{a[i]}_{\mathtt{a}} R' \parallel_L S} \qquad (\textsc{Par}^\bullet)\ \frac{R \overset{a[i]}{\dashrightarrow}_{\mathtt{a}} R' \quad a \notin L \quad i \notin \mathtt{keys}_{\mathtt{a}}(S)}{R \parallel_L S \overset{a[i]}{\dashrightarrow}_{\mathtt{a}} R' \parallel_L S}$$

$$(\textsc{Coo})\ \frac{R \xrightarrow{a[i]}_{\mathtt{a}} R' \quad S \xrightarrow{a[i]}_{\mathtt{a}} S' \quad a \in L}{R \parallel_L S \xrightarrow{a[i]}_{\mathtt{a}} R' \parallel_L S'} \qquad (\textsc{Coo}^\bullet)\ \frac{R \overset{a[i]}{\dashrightarrow}_{\mathtt{a}} R' \quad S \overset{a[i]}{\dashrightarrow}_{\mathtt{a}} S' \quad a \in L}{R \parallel_L S \overset{a[i]}{\dashrightarrow}_{\mathtt{a}} R' \parallel_L S'}$$

Table 2. Structural operational semantic rules for RTPC action transitions

Let $\mathcal{A}_\mathcal{K} = \mathcal{A} \times \mathcal{K}$ and $\mathbb{N}_\mathcal{K} = \mathbb{N}_{>0} \times \mathcal{K}$, with $\mathcal{L} = \mathcal{A}_\mathcal{K} \cup \mathbb{N}_\mathcal{K}$ ranged over by ℓ. Let $\delta(n,i)$ denote $(n)^{[i]}$ or $\langle n^i \rangle$, with the use of the latter being explained later and terms of the form $\langle n^i \rangle\,.\,R$ being added to the syntax thus yielding $\mathcal{P}'$. The semantics for RTPC is the labeled transition system $(\mathcal{P}', \mathcal{L}, \longmapsto)$. The transition relation $\longmapsto \subseteq \mathcal{P}' \times \mathcal{L} \times \mathcal{P}'$ is given by $\longmapsto = \longrightarrow \cup \dashrightarrow$ where in turn the *forward transition relation* is given by $\longrightarrow = \longrightarrow_{\mathtt{a}} \cup \longrightarrow_{\mathtt{d}}$ and the *backward transition relation* is given by $\dashrightarrow = \dashrightarrow_{\mathtt{a}} \cup \dashrightarrow_{\mathtt{d}}$. In the definitions of the transition relations, we make use of the set $\mathtt{keys}_{\mathtt{a}}(R)$ of action keys in a process $R \in \mathcal{P}'$:

$$\begin{aligned}
\mathtt{keys}_{\mathtt{a}}(P) &= \emptyset \\
\mathtt{keys}_{\mathtt{a}}(a[i]\,.\,R) &= \{i\} \cup \mathtt{keys}_{\mathtt{a}}(R) \\
\mathtt{keys}_{\mathtt{a}}(\delta(n,i)\,.\,R) &= \mathtt{keys}_{\mathtt{a}}(R) \\
\mathtt{keys}_{\mathtt{a}}(R + S) &= \mathtt{keys}_{\mathtt{a}}(R) \cup \mathtt{keys}_{\mathtt{a}}(S) \\
\mathtt{keys}_{\mathtt{a}}(R \parallel_L S) &= \mathtt{keys}_{\mathtt{a}}(R) \cup \mathtt{keys}_{\mathtt{a}}(S)
\end{aligned}$$

as well as of predicate $\mathtt{npa}(\_)$ to establish whether the considered process $R \in \mathcal{P}'$ contains no past actions (note that $\mathtt{std}(R)$ ensures $\mathtt{npa}(R)$):

$$\begin{aligned}
\mathtt{npa}(P) &= \mathtt{true} \\
\mathtt{npa}(a[i]\,.\,R) &= \mathtt{false} \\
\mathtt{npa}(\delta(n,i)\,.\,R) &= \mathtt{npa}(R) \\
\mathtt{npa}(R + S) &= \mathtt{npa}(R) \wedge \mathtt{npa}(S) \\
\mathtt{npa}(R \parallel_L S) &= \mathtt{npa}(R) \wedge \mathtt{npa}(S)
\end{aligned}$$

The *action transition relations* $\longrightarrow_{\mathtt{a}} \subseteq \mathcal{P}' \times \mathcal{A}_\mathcal{K} \times \mathcal{P}'$ and $\dashrightarrow_{\mathtt{a}} \subseteq \mathcal{P}' \times \mathcal{A}_\mathcal{K} \times \mathcal{P}'$ are the least relations respectively induced by the forward rules in the left part of Table 2 and by the backward rules in the right part of the same table.

Rule Act1 handles processes of the form $a\,.\,P$, where P is written as R subject to $\mathtt{std}(R)$. In addition to transforming the action prefix into a transition label, it generates a key i that is bound to the action a thus yielding the label $a[i]$. As can be noted, according to [32] the prefix is not discarded by the application of the rule, instead it becomes a key-storing part of the target process that is necessary to offer again that action after coming back. Rule Act1$^\bullet$ reverts the action $a[i]$ of the process $a[i]\,.\,R$ provided that R is a standard process, which ensures that $a[i]$ is the only past action that is left to undo.

The presence of rules Act2 and Act2$^\bullet$ is motivated by the fact that rule Act1 does not discard the executed prefix from the process it generates. In particular, rule Act2 allows a process $a[i]\,.\,R$ to execute if R itself can execute, provided that the action performed by R picks a key j different from i so that all the action prefixes in a sequence are decorated with distinct keys. Rule Act2$^\bullet$ simply propagates the execution of backward actions from inner subprocesses that are not standard as long as key uniqueness is preserved, in such a way that past actions are overall undone from the most recent one to the least recent one. Rules Act3 and Act3$^\bullet$ play an analogous propagating role in a delay context; executed actions and elapsed delays are not required to feature different keys.

Unlike the classical rules for nondeterministic choice [28], according to [32] rule Cho does not discard the part of the overall process that has not contributed to the executed action. If process R does an action, say $a[i]$, and becomes R', then the entire process $R + S$ becomes $R' + S$ as the information about $+\,S$ is necessary for offering again the original choice after coming back. Once the choice is made, only the non-standard process R' can proceed further, with process S – which is standard or contains past delays – constituting a dead context of R'. Note that, in order to apply rule Cho, at least one of R and S must contain no past actions, meaning that it is impossible for two processes containing past actions to execute if they are composed by a choice operator. Rule Cho$^\bullet$ has precisely the same structure as rule Cho, but deals with the backward transition relation; if R' is standard, then the dead context S will come into play again. The symmetric variants of Cho and Cho$^\bullet$, in which it is S to move, are omitted.

The semantics of parallel composition is inspired by [18]. Rule Par allows process R within $R \parallel_L S$ to individually perform an action $a[i]$ provided $a \notin L$. It is also checked that the executing action is bound to a fresh key $i \notin \mathtt{keys_a}(S)$, thus ensuring the uniqueness of communication keys across parallel composition too. Rule Coo instead allows both R and S to move by synchronizing on any action in the set L as long as the communication key is the same on both sides, i.e., $i \in \mathtt{keys_a}(R') \cap \mathtt{keys_a}(S')$. The resulting cooperation action has the same name and the same key. Rules Par$^\bullet$ and Coo$^\bullet$ respectively have the same structure as Par and Coo. The symmetric variants of Par and Par$^\bullet$ are omitted.

To illustrate the need of communication keys, consider for instance the standard forward process $(a\,.\,P_1 \parallel_\emptyset a\,.\,P_2) \parallel_{\{a\}} (a\,.\,P_3 \parallel_\emptyset a\,.\,P_4)$, which may evolve to $(a[i]\,.\,P_1 \parallel_\emptyset a[j]\,.\,P_2) \parallel_{\{a\}} (a[i]\,.\,P_3 \parallel_\emptyset a[j]\,.\,P_4)$ after doing a forward $a[i]$-transition followed by a forward $a[j]$-transition. When going backward, in the absence of communication keys i and j we could not know that the a preceding P_1 (resp. P_2) synchronized with the a preceding P_3 (resp. P_4).

(IDLING1) $\underline{0} \xrightarrow{(n)^{[i]}}_{\mathtt{d}} \langle n^i \rangle . \underline{0}$	(IDLING1$^\bullet$) $\langle n^i \rangle . \underline{0} \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} \underline{0}$
(IDLING2) $\dfrac{\mathtt{std}(R)}{a . R \xrightarrow{(n)^{[i]}}_{\mathtt{d}} \langle n^i \rangle . a . R}$	(IDLING2$^\bullet$) $\dfrac{\mathtt{std}(R)}{\langle n^i \rangle . a . R \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} a . R}$
(IDLING3) $\dfrac{\mathtt{std}(R) \quad a \neq \tau}{a . R \xrightarrow{(n)^{[i]}}_{\mathtt{d}} \langle n^i \rangle . a . R}$	(IDLING3$^\bullet$) $\dfrac{\mathtt{std}(R) \quad a \neq \tau}{\langle n^i \rangle . a . R \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} a . R}$
(DELAY1) $\dfrac{\mathtt{std}(R)}{(n) . R \xrightarrow{(n)^{[i]}}_{\mathtt{d}} (n)^{[i]} . R}$	(DELAY1$^\bullet$) $\dfrac{\mathtt{std}(R)}{(n)^{[i]} . R \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} (n) . R}$
(DELAY2) $\dfrac{R \xrightarrow{(n)^{[j]}}_{\mathtt{d}} R'}{a[i] . R \xrightarrow{(n)^{[j]}}_{\mathtt{d}} a[i] . R'}$	(DELAY2$^\bullet$) $\dfrac{R \overset{(n)^{[j]}}{\dashrightarrow}_{\mathtt{d}} R'}{a[i] . R \overset{(n)^{[j]}}{\dashrightarrow}_{\mathtt{d}} a[i] . R'}$
(DELAY3) $\dfrac{R \xrightarrow{(m)^{[j]}}_{\mathtt{d}} R' \quad j \neq i}{\delta(n,i) . R \xrightarrow{(m)^{[j]}}_{\mathtt{d}} \delta(n,i) . R'}$	(DELAY3$^\bullet$) $\dfrac{R \overset{(m)^{[j]}}{\dashrightarrow}_{\mathtt{d}} R' \quad j \neq i}{\delta(n,i) . R \overset{(m)^{[j]}}{\dashrightarrow}_{\mathtt{d}} \delta(n,i) . R'}$
(TADD1) $\dfrac{R \xrightarrow{(m)^{[j]}}_{\mathtt{d}} R' \quad \mathtt{std}(R) \quad j \neq i}{(n) . R \xrightarrow{(n+m)^{[i]}}_{\mathtt{d}} (n)^{[i]} . R'}$	(TADD1$^\bullet$) $\dfrac{R \overset{(m)^{[j]}}{\dashrightarrow}_{\mathtt{d}} R' \quad \mathtt{std}(R') \quad j \neq i}{(n)^{[i]} . R \overset{(n+m)^{[i]}}{\dashrightarrow}_{\mathtt{d}} (n) . R'}$
(TADD2) $\dfrac{\mathtt{std}(R) \quad n = n_1 + n_2}{(n) . R \xrightarrow{(n_1)^{[i]}}_{\mathtt{d}} (n_1)^{[i]} . (n_2) . R}$	(TADD2$^\bullet$) $\dfrac{\mathtt{std}(R) \quad n = n_1 + n_2}{(n_1)^{[i]} . (n_2) . R \overset{(n_1)^{[i]}}{\dashrightarrow}_{\mathtt{d}} (n) . R}$
(TCHO1) $\dfrac{R \xrightarrow{(n)^{[i]}}_{\mathtt{d}} R' \quad S \xrightarrow{(n)^{[i]}}_{\mathtt{d}} S' \quad \mathtt{npa}(R+S)}{R + S \xrightarrow{(n)^{[i]}}_{\mathtt{d}} R' + S'}$	(TCHO1$^\bullet$) $\dfrac{R \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} R' \quad S \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} S' \quad \mathtt{npa}(R+S)}{R + S \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} R' + S'}$
(TCHO2) $\dfrac{R \xrightarrow{(n)^{[i]}}_{\mathtt{d}} R' \quad \neg\mathtt{npa}(R) \quad \mathtt{npa}(S)}{R + S \xrightarrow{(n)^{[i]}}_{\mathtt{d}} R' + S}$	(TCHO2$^\bullet$) $\dfrac{R \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} R' \quad \neg\mathtt{npa}(R) \quad \mathtt{npa}(S)}{R + S \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} R' + S}$
(TCOO) $\dfrac{R \xrightarrow{(n)^{[i]}}_{\mathtt{d}} R' \quad S \xrightarrow{(n)^{[i]}}_{\mathtt{d}} S'}{R \parallel_L S \xrightarrow{(n)^{[i]}}_{\mathtt{d}} R' \parallel_L S'}$	(TCOO$^\bullet$) $\dfrac{R \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} R' \quad S \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} S'}{R \parallel_L S \overset{(n)^{[i]}}{\dashrightarrow}_{\mathtt{d}} R' \parallel_L S'}$

Table 3. Structural operational semantic rules for RTPC delay transitions

The *delay transition relations* $\longrightarrow_{\mathtt{d}} \subseteq \mathcal{P}' \times \mathbb{N}_{\mathcal{K}} \times \mathcal{P}'$ and $\dashrightarrow_{\mathtt{d}} \subseteq \mathcal{P}' \times \mathbb{N}_{\mathcal{K}} \times \mathcal{P}'$ are the least relations respectively induced by the forward rules in the left part of Table 3 and by the backward rules in the right part of the same table.

Rules IDLING1 and IDLING2 encode laziness: $\underline{0}$ can let time pass and every action can be delayed arbitrarily long. Rules IDLING1 and IDLING3 encode maximal progress: $\underline{0}$ can let time pass and every action other than τ can be delayed arbitrarily long. Rules IDLING1$^\bullet$, IDLING2$^\bullet$, and IDLING3$^\bullet$ play the corresponding roles in the backward direction; all the six rules are dropped under eagerness. Note that the three forward rules introduce a dynamic delay prefix $\langle n^i \rangle$ in the target process, which is then removed from the source process by the three backward rules. The need for $\langle n^i \rangle$ will be illustrated in Example 1.

The pairs of rules DELAY1 and DELAY1$^\bullet$, DELAY2 and DELAY2$^\bullet$, and DELAY3 and DELAY3$^\bullet$ are respectively the delay counterparts of the pairs of rules ACT1 and ACT1$^\bullet$, ACT2 and ACT2$^\bullet$, and ACT3 and ACT3$^\bullet$. We remind that executed actions and elapsed delays are allowed to share the same keys.

Rules TADD1 and TADD2 encode *time additivity* [29], i.e., several consecutive delays can jointly elapse as a single delay and, on the other hand, a single delay can be split into several shorter delays that elapse consecutively. Rules TADD1$^\bullet$ and TADD2$^\bullet$ play the corresponding roles in the backward direction.

Rules TCHO1 and TCOO encode time determinism. Time advances in the same way in the two subprocesses of a choice or a parallel composition, without making any decision. Rule TCHO2 is necessary for dealing with the case in which the nondeterministic choice has already been resolved due to the execution of an action by R. Rules TCHO1$^\bullet$, TCOO$^\bullet$, and TCHO2$^\bullet$ play the corresponding roles in the backward direction. The symmetric variants of TCHO2 and TCHO2$^\bullet$, in which it is S to move, are omitted.

To illustrate the need of communication keys also for delays, consider the standard forward process $(n) \,.\, \underline{0} \parallel_\emptyset (n) \,.\, \underline{0}$, which may evolve to $(n)^{[i]} \,.\, \underline{0} \parallel_\emptyset (n)^{[i]} \,.\, \underline{0}$ after that delay n has elapsed. When going backward under laziness or maximal progress, in the absence of key i it may happen that $\underline{0}$ in either subprocess lets n time units pass backward – due to IDLING1$^\bullet$ – with this pairing with undoing delay n in the other subprocess – due to DELAY1$^\bullet$ – which results in a process where one delay n can elapse forward again whereas the other one cannot. The presence of keys forces the idling of either $\underline{0}$ to synchronize with the idling of the other $\underline{0}$. The same situation would take place with $+$ in lieu of $\parallel_\emptyset$. In contrast, the problem does not show up under eagerness. In that case, elapsed delays can be uniformly decorated, for example with † like in [5].

Process syntax prevents future actions or delays from preceding past ones. However, this is not the only necessary limitation, because not all the processes generated by the considered grammar are semantically meaningful. On the one hand, in the case of a choice at least one of the two subprocesses has to contain no past actions, hence for instance $a[i] \,.\, \underline{0} + b[j] \,.\, \underline{0}$ is not admissible as it indicates that both branches have been selected. On the other hand, key uniqueness must be enforced within processes containing past actions, so for example $a[i] \,.\, b[i] \,.\, \underline{0}$ and $a[i] \,.\, \underline{0} \parallel_\emptyset b[i] \,.\, \underline{0}$ are not admissible either.

In the following we thus consider only *reachable processes*, whose set we denote by $\mathbb{P}$. They include processes from which a computation can start, i.e., standard forward processes, as well as processes that can be derived from the previous ones via finitely many applications of the semantic rules. Given a reachable process $R \in \mathbb{P}$, if $\mathtt{npa}(R)$ then $\mathtt{keys}_{\mathtt{a}}(R) = \emptyset$ while $\mathtt{keys}_{\mathtt{a}}(R') \neq \emptyset$ for any other process R' reachable from R in which at least one of the actions occurring in R has been executed, as that action has been equipped with a key inside R'.

We conclude by showing the validity of time determinism and time additivity. The former holds in the forward direction up to the keys associated with the considered delay, which is formalized via syntactical substitutions of delay keys, because DELAY1 creates a fresh key (whereas DELAY1$^\bullet$ uses the existing one). As for the latter, two distinct processes may be reached in the forward direction because elapsed delays are kept within processes. This is illustrated in Figure 1 by the 2-delay forward transition on the right and the two consecutive 1-delay forward transitions on the left. Dually, in the backward direction, the starting processes may be different, as exemplified by the 2-delay backward transition on the right and the two consecutive 1-delay backward transitions on the left.

Proposition 1 (time determinism). *Let $R, S_1, S_2 \in \mathbb{P}$, $n \in \mathbb{N}_{>0}$, $i_1, i_2 \in \mathcal{K}$, and $j \in \mathcal{K}$ be not occurring associated with past delays in S_1 and S_2. Then:*

- *If $R \xrightarrow{(n)^{[i_1]}}_{\mathtt{d}} S_1$ and $R \xrightarrow{(n)^{[i_2]}}_{\mathtt{d}} S_2$, then $S_1\{^j/_{i_1}\} = S_2\{^j/_{i_2}\}$.*
- *If $R \xrightarrow{(n)^{[i_1]}}_{\mathtt{d}} S_1$ and $R \xrightarrow{(n)^{[i_2]}}_{\mathtt{d}} S_2$, then $S_1 = S_2$.* ■

Proposition 2 (time additivity). *Let $R, R', S, S' \in \mathbb{P}$, $n, h \in \mathbb{N}_{>0}$, $i \in \mathcal{K}$, and $m_1, \ldots, m_h \in \mathbb{N}_{>0}$ be such that $\sum_{1 \leq l \leq h} m_l = n$. Then:*

- *$R \xrightarrow{(n)^{[i]}}_{\mathtt{d}} S$ iff $R \xrightarrow{(m_1)^{[i_1]}}_{\mathtt{d}} \ldots \xrightarrow{(m_h)^{[i_h]}}_{\mathtt{d}} S'$.*
- *$R \xrightarrow{(n)^{[i]}}_{\mathtt{d}} S$ iff $R' \xrightarrow{(m_1)^{[i_1]}}_{\mathtt{d}} \ldots \xrightarrow{(m_h)^{[i_h]}}_{\mathtt{d}} S$.* ■

3 Causal Reversibility of RTPC

We now prove the causal reversibility of RTPC. This means that each reachable process of RTPC is able to backtrack *correctly*, i.e., without encountering previously inaccessible states, and *flexibly*, i.e., along any path that is causally equivalent to the one undertaken in the forward direction. This is accomplished through the notion of concurrent transitions of [12] and the technique of [25].

A necessary condition for reversibility is the *loop property* [12,32,25]. It establishes that each executed action can be undone and that each undone action can be redone, which in a timed setting needs to be extended to delays. Therefore, when considering the states associated with two arbitrary reachable processes, either there is no transition between them, or there is a pair of identically labeled transitions such that one is a forward transition from the first to the second state while the other is a backward transition from the second to the first state.

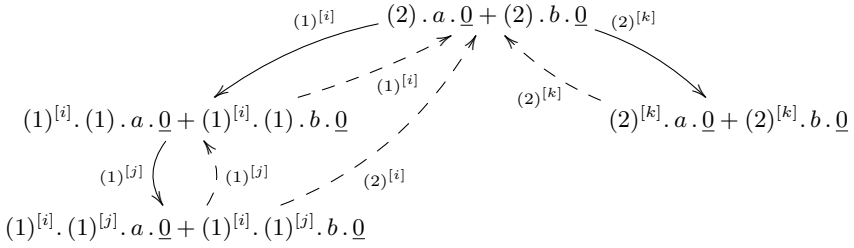

Fig. 1. Time additivity and loop property (only delay transitions are depicted)

To be precise, due to time additivity, a delay transition in one direction may be matched by a sequence of delay transitions in the opposite direction, such that the label of the former is equal to the sum of the labels of the latter. This can be seen in Figure 1. Each of the four 1-delay transitions on the left and of the two 2-delay transitions on the right is matched by a single identically labeled delay transition in the opposite direction, whereas the 2-delay backward transition on the left is matched only by a sequence of two 1-delay forward transitions.

Proposition 3 (loop property). *Let $R, S \in \mathbb{P}$, $a[i], (n)^{[i]} \in \mathcal{L}$, $h \in \mathbb{N}_{>0}$, and $m_1, \ldots, m_h \in \mathbb{N}_{>0}$ be such that $\sum_{1 \leq l \leq h} m_l = n$. Then:*

- $R \xrightarrow{a[i]}_{\mathtt{a}} S$ *iff* $S \dashrightarrow_{\mathtt{a}} R$.
- *If* $R \xrightarrow{(n)^{[i]}}_{\mathtt{d}} S$ *then* $S \dashrightarrow_{\mathtt{d}} R$.
- $R \xrightarrow{(n)^{[i]}}_{\mathtt{d}} S$ *iff* $S \dashrightarrow_{\mathtt{d}} \ldots \dashrightarrow_{\mathtt{d}} R$.
- $R \dashrightarrow_{\mathtt{d}} S$ *iff* $S \xrightarrow{(m_1)^{[i_1]}}_{\mathtt{d}} \ldots \xrightarrow{(m_h)^{[i_h]}}_{\mathtt{d}} R$. ∎

Given a transition $\theta : R \overset{\ell}{\longmapsto} S$ with $R, S \in \mathbb{P}$, we call R the *source* of θ and S its *target*. If θ is a forward transition, i.e., $\theta : R \xrightarrow{\ell} S$, we denote by $\overline{\theta} : S \dashrightarrow R$ the corresponding backward transition. Two transitions are said to be *coinitial* if they have the same source and *cofinal* if they have the same target. Two transitions are *composable* when the target of the first transition coincides with the source of the second transition. A finite sequence of pairwise composable transitions is called a *path*. We use ϵ for the empty path and ω to range over paths, with $|\omega|$ denoting the length of ω, i.e., the number of transitions constituting ω. When ω is a forward path, we denote by $\overline{\omega}$ the corresponding backward path, where the order of the transitions is reversed, and by $\mathtt{time}(\omega)$ the duration of ω, i.e., the sum of the labels of its delay transitions. The notions of source, target, coinitiality, cofinality, and composability naturally extend to paths. We indicate with $\omega_1\omega_2$ the composition of the two paths ω_1 and ω_2 when they are composable.

Before specifying when two transitions are concurrent [12], we need to present the notion of process context along with the set of causes – identified by action keys – that lead to a given communication key for actions.

A *process context* is a process with a hole $\bullet$ in it, generated by the grammar:

$$\mathcal{C} ::= \bullet \mid a[i]\,.\,\mathcal{C} \mid (n)^{[i]}.\,\mathcal{C} \mid \langle n^i \rangle\,.\,\mathcal{C} \mid R + \mathcal{C} \mid \mathcal{C} + R \mid R \parallel_L \mathcal{C} \mid \mathcal{C} \parallel_L R$$

We write $\mathcal{C}[R]$ to denote the process obtained by replacing the hole in $\mathcal{C}$ with R.

The *causal set* $\mathtt{cau}_\mathtt{a}(R,i)$ of $R \in \mathbb{P}$ until $i \in \mathcal{K}$ is inductively defined as:

$$\begin{array}{rcl}
\mathtt{cau}_\mathtt{a}(P,i) & = & \emptyset \\
\mathtt{cau}_\mathtt{a}(a[j]\,.\,R,i) & = & \begin{cases} \emptyset & \text{if } j = i \text{ or } i \notin \mathtt{keys}_\mathtt{a}(R) \\ \{j\} \cup \mathtt{cau}_\mathtt{a}(R,i) & \text{otherwise} \end{cases} \\
\mathtt{cau}_\mathtt{a}(\delta(n,j)\,.\,R,i) & = & \mathtt{cau}_\mathtt{a}(R,i) \\
\mathtt{cau}_\mathtt{a}(R+S,i) & = & \mathtt{cau}_\mathtt{a}(R,i) \cup \mathtt{cau}_\mathtt{a}(S,i) \\
\mathtt{cau}_\mathtt{a}(R \parallel_L S,i) & = & \mathtt{cau}_\mathtt{a}(R,i) \cup \mathtt{cau}_\mathtt{a}(S,i)
\end{array}$$

If $i \in \mathtt{keys}_\mathtt{a}(R)$, then $\mathtt{cau}_\mathtt{a}(R,i)$ represents the set of keys in R that caused i, with $\mathtt{cau}_\mathtt{a}(R,i) \subset \mathtt{keys}_\mathtt{a}(R)$ as on the one hand $i \notin \mathtt{cau}_\mathtt{a}(R,i)$ and on the other hand keys that are not causally related to i are not considered. A key j causes i if it appears syntactically before i in R; equivalently, i is inside the scope of j.

We are now in a position to define, for coinitial transitions, what we mean by concurrent transitions on the basis of the notion of conflicting transitions. As in previous works, the first condition below tells that a forward action transition is in conflict with a coinitial backward one whenever the latter tries to undo a cause of the key of the former, while the second one deems as conflictual two action transitions respectively generated by the two subprocesses of a choice.

The further conditions are specific to this timed setting. The third one views as conflictual two coinitial delay transitions, regardless of their directions. The fourth one considers as conflictual a forward action transition and a forward delay transition that are coinitial, whereas a similar situation cannot show up in the backward direction because, if a delay can be undone in all subprocesses, then no action can be undone (both cases will be illustrated in Example 1). The fifth one regards as conflictual a forward delay transition and a backward action transition that are coinitial and originated from the same subprocess. In the sixth one the action transition is forward and the delay transition is backward. Figure 2 illustrates the first and the fifth ones.

Definition 1 (conflicting and concurrent transitions). *Two coinitial transitions θ_1 and θ_2 from a process $R \in \mathbb{P}$ are* in conflict *if one of the following conditions holds, otherwise they are said to be* concurrent*:*

1. $\theta_1 : R \xrightarrow{a[i]}_\mathtt{a} S_1$ *and* $\theta_2 : R \overset{b[j]}{\dashrightarrow}_\mathtt{a} S_2$ *with* $j \in \mathtt{cau}_\mathtt{a}(S_1,i)$.
2. $R = \mathcal{C}[P_1 + P_2]$ *with* θ_k *deriving from* $P_k \xrightarrow{a_k[i_k]}_\mathtt{a} S_k$ *for* $k = 1,2$.
3. $\theta_1 : R \overset{(n)^i}{\longmapsto}_\mathtt{d} S_1$ *and* $\theta_2 : R \overset{(m)^j}{\longmapsto}_\mathtt{d} S_2$.
4. $\theta_1 : R \xrightarrow{a[i]}_\mathtt{a} S_1$ *and* $\theta_2 : R \xrightarrow{(m)^j}_\mathtt{d} S_2$.
5. $R = \mathcal{C}[a[i]\,.\,(n)\,.\,P]$ *with* θ_1 *deriving from* $(n)\,.\,P \xrightarrow{(m)^{[j]}}_\mathtt{d} S_1$ *and* θ_2 *deriving from* $a[i]\,.\,(n)\,.\,P \overset{a[i]}{\dashrightarrow}_\mathtt{a} S_2$.
6. $R = \mathcal{C}[\delta(n,i)\,.\,a\,.\,P]$ *with* θ_1 *deriving from* $a\,.\,P \xrightarrow{a[j]}_\mathtt{a} S_1$ *and* θ_2 *deriving from* $\delta(n,i)\,.\,a\,.\,P \overset{(n)^{[i]}}{\dashrightarrow}_\mathtt{d} S_2$. ■

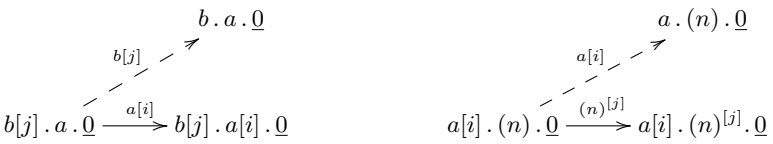

Fig. 2. Examples of conflicting transitions: conditions 1 (left) and 5 (right)

We prove causal reversibility by exploiting the technique of [25], according to which causal consistency stems from the *square property* – which amounts to concurrent transitions being confluent – *backward transitions independence* – which generalizes the concept of backward determinism used for reversible sequential languages [42] – and *past well foundedness* – which ensures that reachable processes have a finite past.

We start with the square property, which states that concurrent transitions can commute while conflicting ones cannot.

Lemma 1 (square property). *Let $\theta_1 : R \stackrel{\ell_1}{\longmapsto} S_1$ and $\theta_2 : R \stackrel{\ell_2}{\longmapsto} S_2$ be two coinitial transitions from a process $R \in \mathbb{P}$. If θ_1 and θ_2 are concurrent, then there exist two cofinal transitions $\theta_2' : S_1 \stackrel{\ell_2}{\longmapsto} S$ and $\theta_1' : S_2 \stackrel{\ell_1}{\longmapsto} S$ with $S \in \mathbb{P}$.* ■

Unlike [25], backward transitions independence holds as long as at least one of the two coinitial backward transitions is not a delay transition.

Lemma 2 (backward transitions independence). *Let $R \in \mathbb{P}$. Then two coinitial backward transitions $\theta_1 : R \stackrel{\ell_1}{\dashrightarrow} S_1$ and $\theta_2 : R \stackrel{\ell_2}{\dashrightarrow} S_2$ are concurrent provided that at least one of them is not a delay transition.* ■

For coinitial backward delay transitions, the novel *backward triangularity* property comes into play, which is exemplified by the backward delay transitions in the left part of Figure 1.

Lemma 3 (backward triangularity). *Let $R \in \mathbb{P}$. Whenever $R \stackrel{(n)^{[i]}}{\dashrightarrow_{\mathtt{d}}} S_1$ and $R \stackrel{(m)^{[j]}}{\dashrightarrow_{\mathtt{d}}} S_2$ with $m > n$, then $S_1 \stackrel{(m-n)^{[k]}}{\dashrightarrow_{\mathtt{d}}} S_2$.* ■

As far as past well foundedness is concerned, under laziness and maximal progress we observe that the adoption of the dynamic delay prefix $\langle n^i \rangle$ in rules IDLING1, IDLING2, IDLING3, IDLING1$^\bullet$, IDLING2$^\bullet$, IDLING3$^\bullet$ avoids the generation of backward self-loops, from which infinite sequence of backward transitions would be obtained.

Lemma 4 (past well foundedness). *Let $R_0 \in \mathbb{P}$. Then there is no infinite sequence of backward transitions such that $R_i \stackrel{\ell_i}{\dashrightarrow} R_{i+1}$ for all $i \in \mathbb{N}$.* ■

Following [12,27], we also define a notion of *causal equivalence* over paths, which abstracts from the order of concurrent action transitions. In this way, paths obtained by swapping the order of their concurrent action transitions are identified with each other. Due to time determinism, the swap operation does not apply to delay transitions. Moreover, the composition of a transition with its inverse is identified with the empty path. Unlike other approaches, our causal equivalence has to deal with time additivity. More precisely, a path made out of forward (resp. backward) delay transitions followed by a path made out of backward (resp. forward) delay transitions returning to the origin is identified with the empty path provided that the two original paths have the same duration. This can be seen in Figure 1 if we take for example the delay path on the left $\xrightarrow{(1)^{[i]}}_{\mathtt{d}}\xrightarrow{(1)^{[j]}}_{\mathtt{d}}\dashrightarrow_{\mathtt{d}}$.

Definition 2 (causal equivalence). Causal equivalence $\asymp$ *is the smallest equivalence relation over paths that is closed under composition and satisfies the following:*

1. $\theta_1\theta_2' \asymp \theta_2\theta_1'$ *for every two coinitial concurrent action transitions* $\theta_1 : R \xmapsto{a[i]} R_1$ *and* $\theta_2 : R \xmapsto{b[j]} R_2$ *and every two cofinal action transitions* $\theta_2' : R_1 \xmapsto{b[j]} S$ *and* $\theta_1' : R_2 \xmapsto{a[i]} S$ *respectively composable with the previous ones.*
2. $\theta\overline{\theta} \asymp \epsilon$ *and* $\overline{\theta}\theta \asymp \epsilon$ *for every transition* θ.
3. $\omega_1\overline{\omega_2} \asymp \epsilon$ *and* $\overline{\omega_2}\omega_1 \asymp \epsilon$ *for every two coinitial and cofinal forward paths* ω_1 *and* ω_2 *with delay transitions only such that* $\mathtt{time}(\omega_1) = \mathtt{time}(\omega_2)$. ■

The further property below, called *parabolic lemma* in [25], states that every path can be seen as a backward path followed by a forward path. As observed in [12], up to causal equivalence one can always reach for the maximum freedom of choice among transitions by going backward and only then going forward (not the other way around). Intuitively, one could depict computations as parabolas: the system first draws potential energy from its memory, by undoing all the executed actions, and then restarts.

In this timed setting the parabolic lemma has to be proven directly. Unlike [25], it does not stem from the square property and backward transitions independence as the latter does not hold for all coinitial backward transitions.

Lemma 5 (parabolic lemma). *For each path* ω*, there exist two forward paths* ω_1 *and* ω_2 *such that* $\omega \asymp \overline{\omega_1}\omega_2$ *and* $|\omega_1| + |\omega_2| \leq |\omega|$. ■

Example 1. If rules IDLING1, IDLING2, IDLING3, IDLING1$^\bullet$, IDLING2$^\bullet$, IDLING3$^\bullet$ did not respectively introduce and retract dynamic delay prefixes of the form $\langle n^i \rangle$, then the parabolic lemma would not hold.

Consider the process $R = a\,.\,(n)\,.\,\underline{0} \parallel_\emptyset (n)\,.\,\underline{0}$. Under eagerness it can initially perform only $R \xrightarrow{a[i]}_{\mathtt{a}} a[i]\,.\,(n)\,.\,\underline{0} \parallel_\emptyset (n)\,.\,\underline{0} = S_1$. Under laziness it can also perform $R \xrightarrow{(n)^{[j]}}_{\mathtt{d}} \langle n^j \rangle\,.\,a\,.\,(n)\,.\,\underline{0} \parallel_\emptyset (n)^{[j]}.\,\underline{0} = S_2$ because in R the execution of

action a on the left can be postponed via IDLING2 by as many time units as there are in delay (n) on the right where DELAY1 applies, then TCOO is used.

If we keep going forward, we obtain $S_1 \xrightarrow{(n)^{[j]}}_{\mathtt{d}} a[i] \,.\, (n)^{[j]} \,.\, \underline{0} \parallel_{\emptyset} (n)^{[j]} \,.\, \underline{0} = S_1'$ while on the lazy side $S_2 \xrightarrow{a[i]}_{\mathtt{a}} \langle n^j \rangle \,.\, a[i] \,.\, (n) \,.\, \underline{0} \parallel_{\emptyset} (n)^{[j]} \,.\, \underline{0} = S_2'$ followed by $S_2' \xrightarrow{(n)^{[k]}}_{\mathtt{d}} \langle n^j \rangle \,.\, a[i] \,.\, (n)^{[k]} \,.\, \underline{0} \parallel_{\emptyset} (n)^{[j]} \,.\, \langle n^k \rangle \,.\, \underline{0} = S_2''$ because in S_2' the subprocess $\underline{0}$ on the right can let via IDLING1 as many time units pass as there are in delay (n) on the left. Note that the square property does not hold because S_1' and S_2'' are different. Indeed, the two initial transitions of R are in conflict according to condition 4 of Definition 1; moreover, from S_1' delay $(n)^{[j]}$ can be undone, whereas action $a[i]$ cannot.

When going backward, by virtue of the presence of dynamic delays $\langle n^j \rangle$ and $\langle n^k \rangle$ all the transitions above are undone in the reverse order and the same states are traversed thanks to IDLING1$^\bullet$ and IDLING2$^\bullet$. However, if dynamic delays were not adopted by the aforementioned idling rules, so that on the lazy side we would end up in $a[i] \,.\, (n)^{[k]} \,.\, \underline{0} \parallel_{\emptyset} (n)^{[j]} \,.\, \underline{0}$, then, observing that $(n)^{[k]}$ and $(n)^{[j]}$ have different keys and hence cannot be undone together via TCOO$^\bullet$, either $(n)^{[k]}$ is undone via IDLING1$^\bullet$ applied to the subprocess $\underline{0}$ on the right, or $(n)^{[j]}$ is undone via IDLING1$^\bullet$ applied to the subprocess $\underline{0}$ on the left. In the latter case, the new state $a[i] \,.\, (n)^{[k]} \,.\, \underline{0} \parallel_{\emptyset} (n) \,.\, \underline{0}$ is encountered, from which it is only possible to redo $(n)^{[j]}$ via IDLING1 applied to the subprocess $\underline{0}$ on the left; note that $(n)^{[k]}$ cannot be undone because $(n) \,.\, \underline{0}$ cannot let time pass backward.

The presence of this new state would violate the parabolic lemma. In particular, the path traversing R, S_2, S_2', S_2'', and the new state could not be causally equivalent to anyone composed of a backward path followed by a forward one. ■

We conclude by obtaining a property called *causal consistency* in [25], which establishes that being coinitial and cofinal is necessary and sufficient in order for two paths to be causally equivalent, i.e., to contain concurrent action transitions in different orders (swap) or to be one the empty path and the other a transition followed by its reverse or a delay path followed by an identically lasting delay path in the reverse direction (cancelation).

Theorem 1 (causal consistency). *Let ω_1 and ω_2 be two paths. Then $\omega_1 \asymp \omega_2$ iff ω_1 and ω_2 are both coinitial and cofinal.* ■

Theorem 1 shows that causal equivalence characterizes a space for admissible rollbacks that are (i) correct as they do not lead to states not reachable by some forward path and (ii) flexible enough to allow on the one hand undo operations to be rearranged with respect to the order in which the undone concurrent action transitions were originally performed and on the other hand time additivity to be taken into account. This implies that the states reached by any backward path could be reached by performing forward paths only. Therefore, we can conclude that RTPC meets causal reversibility.

Example 2. Following [29] we can define the timeout operator TIMEOUT(P, Q, t). It allows the process P to communicate with the environment within t time units.

After this time has passed, and P has not communicated yet, the process Q takes control. The operator is rendered in RTPC as $\textsc{Timeout}(P, Q, t) = P + (t) \,.\, \tau \,.\, Q$ under maximal progress.

Timeouts are usually employed in fault-tolerant systems to prevent some operations from blocking forever. As a matter of fact, the Erlang programming language provides a timeout facility on blocking receive. For example, let us consider the following snippet of Erlang code in which two actors (e.g., processes) execute in parallel:

```
1  process_A() ->                    7  process_B(Pid) ->
2    receive                         8    timer:sleep(100),
3      X -> handleMsg()              9    Pid! Msg   end.
4      after 50 ->                   10
5        handleTimeout()             11 PidA=spawn(?MODULE, process_A, []),
6    end end.                        12 spawn(?MODULE, process_B, [PidA]).
```

Process A (lines 1–6) awaits a message from the environment (e.g., from process B); if a message is received within 50 ms, then process A calls function handleMsg(), otherwise it calls function handleTimeout(). Process B (lines 7–9) sleeps for 100 ms and then sends a message to Pid, the identifier of process A.

The translation of the code for the two processes into RTPC is as follows:

$$\begin{aligned} A &= \textsc{Timeout}(a \,.\, P, Q, 50) \\ B &= (100) \,.\, a \,.\, \underline{0} \end{aligned}$$

where P encodes handleMsg() and Q encodes handleTimeout().

If we run the two processes in parallel (mimicking lines 11–12), we have the following forward execution under maximal progress:

$$\begin{aligned} A \parallel_{\{a\}} B \; &\xrightarrow{(50)^{[i]}}_{\mathtt{d}} \; (a \,.\, P + (50)^{[i]} \,.\, \tau \,.\, Q) \parallel_{\{a\}} ((50)^{[i]} \,.\, (50) \,.\, a \,.\, \underline{0}) \\ &\xrightarrow{\tau[j]}_{\mathtt{a}} \; (a \,.\, P + (50)^{[i]} \,.\, \tau[j] \,.\, Q) \parallel_{\{a\}} ((50)^{[i]} \,.\, (50) \,.\, a \,.\, \underline{0}) \end{aligned}$$

at which point Q takes over. If process B wants to revert its behavior (e.g., going back by 50 ms), it cannot do it alone as it has to wait for process A to first undo $\tau[j]$ and then undo $(50)^{[i]}$ together. This is clearly a causally consistent backward computation as no new process is encountered along the way. ■

4 Conclusions

In this paper we have studied the causal reversibility of timed process calculi. With respect to the reversible nondeterministic setting of [12,32,25], we have addressed a number of issues that are listed at the end of Section 1. With respect to the reversible timed setting of [8], which builds on TPL [17], we have considered not only laziness but also eagerness and, most importantly, like in temporal CCS [29] we have described time via numeric delays – instead of unitary delays – subject to time additivity, which results in a variation of the loop property and a restriction of backward transitions independence.

As future work, we plan to investigate suitable notions of bisimilarity for RTPC based on the approaches of [32,14,6]. Moreover, similar to [5], we would like to allow backward delays to be different from the corresponding forward delays. Finally, we are interested in moving from discrete time to dense time.

Acknowledgments. This work has been supported by the Italian MUR PRIN 2020 project *NiRvAna*, the French ANR project ANR-18-CE25-0007 *DCore*, and the INdAM-GNCS project CUP_E55F22000270001 *Proprietà Qualitative e Quantitative di Sistemi Reversibili*.

References

1. Aceto, L., Murphy, D.: Timing and causality in process algebra. Acta Informatica **33**, 317–350 (1996)
2. Baeten, J.C.M., Bergstra, J.A.: Real time process algebra. Formal Aspects of Computing **3**, 142–188 (1991)
3. Bennett, C.H.: Logical reversibility of computations. IBM Journal of Research and Development **17**, 525–532 (1973)
4. Bernardo, M., Corradini, F., Tesei, L.: Timed process calculi with deterministic or stochastic delays: Commuting between durational and durationless actions. Theoretical Computer Science **629**, 2–39 (2016)
5. Bernardo, M., Mezzina, C.A.: Bridging causal reversibility and time reversibility: A stochastic process algebraic approach. Logical Methods in Computer Science **19(2:6)**, 1–27 (2023)
6. Bernardo, M., Rossi, S.: Reverse bisimilarity vs. forward bisimilarity. In: Proc. of the 26th Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS 2023). LNCS, vol. 13992, pp. 265–284. Springer (2023)
7. Bérut, A., Arakelyan, A., Petrosyan, A., Ciliberto, S., Dillenschneider, R., Lutz, E.: Experimental verification of Landauer's principle linking information and thermodynamics. Nature **483**, 187–189 (2012)
8. Bocchi, L., Lanese, I., Mezzina, C.A., Yuen, S.: The reversible temporal process language. In: Proc. of the 42nd Int. Conf. on Formal Techniques for Distributed Objects, Components, and Systems (FORTE 2022). LNCS, vol. 13273, pp. 31–49. Springer (2022)
9. Bolognesi, T., Lucidi, F.: LOTOS-like process algebras with urgent or timed interactions. In: Proc. of the 4th Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE 1991). IFIP Transactions, vol. C-2, pp. 249–264 (1991)
10. Corradini, F.: Absolute versus relative time in process algebras. Information and Computation **156**, 122–172 (2000)
11. Corradini, F., Vogler, W., Jenner, L.: Comparing the worst-case efficiency of asynchronous systems with PAFAS. Acta Informatica **38**, 735–792 (2002)
12. Danos, V., Krivine, J.: Reversible communicating systems. In: Proc. of the 15th Int. Conf. on Concurrency Theory (CONCUR 2004). LNCS, vol. 3170, pp. 292–307. Springer (2004)
13. Danos, V., Krivine, J.: Transactions in RCCS. In: Proc. of the 16th Int. Conf. on Concurrency Theory (CONCUR 2005). LNCS, vol. 3653, pp. 398–412. Springer (2005)
14. De Nicola, R., Montanari, U., Vaandrager, F.: Back and forth bisimulations. In: Proc. of the 1st Int. Conf. on Concurrency Theory (CONCUR 1990). LNCS, vol. 458, pp. 152–165. Springer (1990)
15. Frank, M.P.: Physical foundations of Landauer's principle. In: Proc. of the 10th Int. Conf. on Reversible Computation (RC 2018). LNCS, vol. 11106, pp. 3–33. Springer (2018)

16. Giachino, E., Lanese, I., Mezzina, C.A.: Causal-consistent reversible debugging. In: Proc. of the 17th Int. Conf. on Fundamental Approaches to Software Engineering (FASE 2014). LNCS, vol. 8411, pp. 370–384. Springer (2014)
17. Hennessy, M., Regan, T.: A process algebra for timed systems. Information and Computation **117**, 221–239 (1995)
18. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall (1985)
19. Kelly, F.P.: Reversibility and Stochastic Networks. John Wiley & Sons (1979)
20. Krivine, J.: A verification technique for reversible process algebra. In: Proc. of the 4th Int. Workshop on Reversible Computation (RC 2012). LNCS, vol. 7581, pp. 204–217. Springer (2012)
21. Landauer, R.: Irreversibility and heat generated in the computing process. IBM Journal of Research and Development **5**, 183–191 (1961)
22. Lanese, I., Lienhardt, M., Mezzina, C.A., Schmitt, A., Stefani, J.B.: Concurrent flexible reversibility. In: Proc. of the 22nd European Symp. on Programming (ESOP 2013). LNCS, vol. 7792, pp. 370–390. Springer (2013)
23. Lanese, I., Medić, D., Mezzina, C.A.: Static versus dynamic reversibility in CCS. Acta Informatica **58**, 1–34 (2021)
24. Lanese, I., Nishida, N., Palacios, A., Vidal, G.: CauDEr: A causal-consistent reversible debugger for Erlang. In: Proc. of the 14th Int. Symp. on Functional and Logic Programming (FLOPS 2018). LNCS, vol. 10818, pp. 247–263. Springer (2018)
25. Lanese, I., Phillips, I., Ulidowski, I.: An axiomatic approach to reversible computation. In: Proc. of the 23rd Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS 2020). LNCS, vol. 12077, pp. 442–461. Springer (2020)
26. Laursen, J.S., Ellekilde, L.P., Schultz, U.P.: Modelling reversible execution of robotic assembly. Robotica **36**, 625–654 (2018)
27. Lévy, J.J.: An algebraic interpretation of the $\lambda\beta$K-calculus; and an application of a labelled λ-calculus. Theoretical Computer Science **2**, 97–114 (1976)
28. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
29. Moller, F., Tofts, C.: A temporal calculus of communicating systems. In: Proc. of the 1st Int. Conf. on Concurrency Theory (CONCUR 1990). LNCS, vol. 458, pp. 401–415. Springer (1990)
30. Nicollin, X., Sifakis, J.: The algebra of timed processes ATP: Theory and application. Information and Computation **114**, 131–178 (1994)
31. Perumalla, K.S., Park, A.J.: Reverse computation for rollback-based fault tolerance in large parallel systems - Evaluating the potential gains and systems effects. Cluster Computing **17**, 303–313 (2014)
32. Phillips, I., Ulidowski, I.: Reversing algebraic process calculi. Journal of Logic and Algebraic Programming **73**, 70–96 (2007)
33. Phillips, I., Ulidowski, I., Yuen, S.: A reversible process calculus and the modelling of the ERK signalling pathway. In: Proc. of the 4th Int. Workshop on Reversible Computation (RC 2012). LNCS, vol. 7581, pp. 218–232. Springer (2012)
34. Pinna, G.M.: Reversing steps in membrane systems computations. In: Proc. of the 18th Int. Conf. on Membrane Computing (CMC 2017). LNCS, vol. 10725, pp. 245–261. Springer (2017)
35. Quemada, J., de Frutos, D., Azcorra, A.: TIC: A timed calculus. Formal Aspects of Computing **5**, 224–252 (1993)
36. Reed, G.M., Roscoe, A.W.: A timed model for communicating sequential processes. Theoretical Computer Science **58**, 249–261 (1988)

37. Schordan, M., Oppelstrup, T., Jefferson, D.R., Barnes Jr., P.D.: Generation of reversible C++ code for optimistic parallel discrete event simulation. New Generation Computing **36**, 257–280 (2018)
38. Siljak, H., Psara, K., Philippou, A.: Distributed antenna selection for massive MIMO using reversing Petri nets. IEEE Wireless Communication Letters **8**, 1427–1430 (2019)
39. Vassor, M., Stefani, J.B.: Checkpoint/rollback vs causally-consistent reversibility. In: Proc. of the 10th Int. Conf. on Reversible Computation (RC 2018). LNCS, vol. 11106, pp. 286–303. Springer (2018)
40. de Vries, E., Koutavas, V., Hennessy, M.: Communicating transactions. In: Proc. of the 21st Int. Conf. on Concurrency Theory (CONCUR 2010). LNCS, vol. 6269, pp. 569–583. Springer (2010)
41. Yi, W.: CCS + time = an interleaving model for real time systems. In: Proc. of the 18th Int. Coll. on Automata, Languages and Programming (ICALP 1991). LNCS, vol. 510, pp. 217–228. Springer (1991)
42. Yokoyama, T., Glück, R.: A reversible programming language and its invertible self-interpreter. In: Proc. of the 13th ACM Workshop on Partial Evaluation and Semantics-based Program Manipulation (PEPM 2007). pp. 144–153. ACM Press (2007)

A Proofs of Results

Proof of Proposition 1.
Straightforward. ■

Proof of Proposition 2.
Straightforward. ■

Proof of Proposition 3.
By induction on the depth of the derivation of the transition on the left by noting that each forward (resp. backward) rule in Tables 2 and 3 has a corresponding backward (resp. forward) rule in the same table. ■

Proof of Lemma 1.
The proof is by case analysis on the direction of θ_1 and θ_2. We distinguish three cases according to whether the two transitions are both forward, both backward, or one forward and the other backward:

- If θ_1 and θ_2 are both forward, there are three subcases:
  - If their labels are actions, since θ_1 and θ_2 are concurrent, by virtue of condition 2 of Definition 1 the two transitions cannot originate from a choice operator. They must thus be generated by a parallel composition, but not through rule Coo because θ_1 and θ_2 must have different keys and hence cannot synchronize. Without loss of generality, we can assume that $R = R_1 \parallel_L R_2$ with $R_1 \xrightarrow{a[i]}_{\mathtt{a}} S_1$, $R_2 \xrightarrow{b[j]}_{\mathtt{a}} S_2$, $a, b \notin L$, and $i \neq j$. By applying rule Par we have $R_1 \parallel_L R_2 \xrightarrow{a[i]}_{\mathtt{a}} S_1 \parallel_L R_2 \xrightarrow{b[j]}_{\mathtt{a}} S_1 \parallel_L S_2$ as well as $R_1 \parallel_L R_2 \xrightarrow{b[j]}_{\mathtt{a}} R_1 \parallel S_2 \xrightarrow{a[i]}_{\mathtt{a}} S_1 \parallel_L S_2$.
  - If their labels are delays, by virtue of condition 3 of Definition 1 the two transitions cannot be concurrent, hence this subcase does not apply.
  - If one label is an action and the other is a delay, by virtue of condition 4 of Definition 1 the two transitions cannot be concurrent, hence this subcase does not apply.
- If θ_1 and θ_2 are both backward, there are again three subcases, with the first two being similar to the corresponding ones of the previous case, while the third one cannot show up because, if a delay can be undone in all subprocesses, then no action can be undone.
- If θ_1 is forward and θ_2 is backward, there are three subcases:
  - If their labels are actions, since θ_1 and θ_2 are concurrent, by virtue of condition 1 of Definition 1 it holds that θ_2 cannot remove any cause of θ_1. Since any subprocess of a choice or a parallel composition cannot perform a forward transition and a backward transition without preventing the backward one from removing a cause of the forward one, and in the case of a choice only one of the two subprocesses can perform transitions after the choice has been made (as would be in our case in which we are considering a backward transition), without loss of generality we can assume that $R = R_1 \parallel_L R_2$ with $R_1 \xrightarrow{a[i]}_{\mathtt{a}} S_1$, $R_2 \overset{b[j]}{\dashrightarrow}_{\mathtt{a}} S_2$, $a, b \notin L$,

and $i \neq j$. By applying rule PAR we have $R_1 \parallel_L R_2 \xrightarrow{a[i]}_{\mathtt{a}} S_1 \parallel_L R_2$ $\overset{b[j]}{\dashrightarrow}_{\mathtt{a}} S_1 \parallel_L S_2$ as well as $R_1 \parallel_L R_2 \overset{b[j]}{\dashrightarrow}_{\mathtt{a}} R_1 \parallel_L S_2 \xrightarrow{a[i]}_{\mathtt{a}} R_1 \parallel_L R_2$.

- If their labels are delays, by virtue of condition 3 of Definition 1 the two transitions cannot be concurrent, hence this subcase does not apply.
- If one label is an action and the other is a delay, since θ_1 and θ_2 are concurrent, by virtue of conditions 5 and 6 of Definition 1 the two transitions cannot be generated by the same subprocess. Hence, without loss of generality we can assume that $R = R_1 \parallel_L R_2$ with $a \notin L$. There are two further subcases:
  - $R_1 \xrightarrow{(m)^{[j]}}_{\mathtt{d}} S_1$ and $R_2 \overset{a[i]}{\dashrightarrow}_{\mathtt{a}} S_2$. Under eagerness the latter transition would prevail at the level of R and hence the subcase would not apply. Under laziness the former transition would be possible at the level of R if R_2 could perform an identically labeled forward delay transition via idling. But then both the forward $(m)^{[j]}$-transition and the backward $a[i]$-transition at the level of R would be generated by the same subprocess R_2, hence again the subcase would not apply.
  - $R_1 \overset{(m)^{[j]}}{\dashrightarrow}_{\mathtt{a}} S_1$ and $R_2 \xrightarrow{a[i]}_{\mathtt{d}} S_2$. Similar to the previous subcase. ■

Proof of Lemma 2.
By Definition 1 it is not possible for two backward transitions to be in conflict except when they are both delay transitions. ■

Proof of Lemma 3.
A straightforward consequence of time additivity. ■

Proof of Lemma 4.
By induction on $|\mathtt{keys}_{\mathtt{a}}(R_0)| + \mathit{past}(R_0)$, where $\mathit{past}(R_0)$ is the sum of the past delays in R_0. Indeed, every backward transition between two different processes either decreases by one the total number of past actions or diminishes the sum of the past delays, with both numbers being finite. ■

Proof of Lemma 5.
Let $d(\omega)$ be the number of discording pairs within path ω, where two forward transitions θ_1 and θ_2 form a discording pair iff θ_1 and $\overline{\theta_2}$ occur next to each other in that order inside ω.
If $d(\omega) = 0$, then ω is already formed by a (possibly empty) backward path followed by a forward one.
If $d(\omega) > 0$, the result follows by showing that there exists $\omega' \asymp \omega$ with $|\omega'| \leq |\omega|$ and $d(\omega') < d(\omega)$. Since $d(\omega) > 0$, ω contains at least one discording pair. Let the one formed by θ_1 and θ_2 be the earliest one, where $\omega = \overline{\omega_1}\theta_1\overline{\theta_2}\omega_2$ with ω_1 being forward.
If $\theta_1 = \theta_2$, then trivially $\omega_1\theta_1\overline{\theta_2}\omega_2 \asymp \overline{\omega_1}\omega_2$ with $|\overline{\omega_1}\omega_2| < |\omega|$ and $d(\overline{\omega_1}\omega_2) < d(\omega)$.
If $\theta_1 \neq \theta_2$ with the two transitions being concurrent, by using the square property (Lemma 1) we can swap them thereby obtaining $\overline{\omega_1}\theta_1\overline{\theta_2}\omega_2 \asymp \overline{\omega_1}\overline{\theta_2}\theta_1\omega_2$ with $|\overline{\omega_1}\overline{\theta_2}\theta_1\omega_2| \leq |\omega|$. If ω_2 starts with a forward transition then $d(\overline{\omega_1}\overline{\theta_2}\theta_1\omega_2) < d(\omega)$, otherwise we keep moving right with θ_1 being part of the next earliest discording

pair to consider.
If $\theta_1 \neq \theta_2$ with the two transitions being in conflict, there are three cases based on Definition 1:

- θ_1 and θ_2 are two action transitions with $\overline{\theta_2}$ removing a cause of θ_1 (condition 1). Since it is not possible to perform such a $\overline{\theta_2}$ after θ_1, this case does not apply.
- θ_1 and θ_2 are two delay transitions (condition 3). Either the time elapsed with θ_1 is reverted due to time additivity by a sequence of backward delay transitions starting with $\overline{\theta_2}$, or it remains. In the former subcase $\omega = \overline{\omega_1}\theta_1\overline{\theta_2}\overline{\omega_2'}\omega_2''$ with $\mathtt{time}(\theta_1) = \mathtt{time}(\overline{\theta_2}\overline{\omega_2'})$, hence $\overline{\omega_1}\theta_1\overline{\theta_2}\overline{\omega_2'}\omega_2 \asymp \overline{\omega_1}\omega_2''$ with $|\overline{\omega_1}\omega_2''| < |\omega|$ and $d(\overline{\omega_1}\omega_2'') < d(\omega)$. The latter subcase does not apply due to time determinism, in the sense that if n time units elapse forward then it is not possible to go back by less than n time units.
- One is an action transition and the other is a delay transition, with both being generated by the same subprocess (conditions 5 and 6). Similar to the first case. ■

Proof of Theorem 1.
It follows from past well foundedness and the parabolic lemma thanks to [25]. ■