BRIDGING CAUSAL CONSISTENT AND TIME REVERSIBILITY: A STOCHASTIC PROCESS ALGEBRAIC APPROACH

MARCO BERNARDO AND CLAUDIO ANTARES MEZZINA

Dipartimento di Scienze Pure e Applicate, Università di Urbino, Italy
e-mail address: marco.bernardo@uniurb.it, claudio.mezzina@uniurb.it

ABSTRACT. Causal consistent reversibility blends causality and reversibility. For a concurrent system, it says that an action can be undone provided that this has no consequences, thereby making it possible to bring the system back to a past consistent state. Time reversibility is instead considered in the performance evaluation field, mostly for efficient analysis purposes. A continuous-time Markov chain is time reversible if its stochastic behavior remains the same when the direction of time is reversed. We study how to bridge these two theories of reversibility by showing the conditions under which both causal consistent reversibility and time reversibility can be ensured by construction. This is done in the setting of a stochastic process calculus, which is then equipped with a notion of stochastic bisimilarity accounting for both forward and backward directions.

1. INTRODUCTION

The interest into computation reversibility dates back to the 60’s, when Landauer observed that irreversible computations cause heat dissipation into circuits [23]. More precisely, Landauer’s principle says that any logically irreversible manipulation of information, such as the erasure of a bit or the merging of two computation paths, must be accompanied by a corresponding entropy increase in non-information-bearing degrees of freedom of the information processing apparatus or its environment [4]. Hence, according to this principle, which has been recently verified in [6] and given a physical foundation in [12], any logically-reversible computation, in which no information (e.g., bits) is erased, may be potentially carried out without releasing any heat. This suggested that low energy consumption could be achieved by resorting to reversible computing, in which there is no information loss [3]. Nowadays, reversible computing has several applications ranging from modelling biochemical reactions [41, 40] and parallel discrete-event simulation [38, 43] to robotics [31], control theory [44], fault tolerant systems [9, 11, 24, 46], and program debugging [13, 28].

In a reversible system, we can observe two directions of computation: a forward one, coinciding with the normal way of computing, and a backward one, which is able to undo the effects of the forward one. In the literature, there exist different meanings of reversibility. For instance, in a Petri net model reversibility means that one can always reach the initial marking [2], while in distributed systems it amounts to the capability of returning to a past

Key words and phrases: Reversibility, Causality, Markov Chains, Process Calculi, Bisimilarity.
consistent state [8]. In contrast, in the performance evaluation field, reversibility is intended as time reversibility and is instrumental to develop efficient analysis methods [20].

Our focus is on the relationship between causal consistent reversibility and time reversibility, from a process algebraic perspective. Causal consistent reversibility stands for the capability of going back to a past state that is consistent with the computational history of a system; in this setting, quantitative aspects have been totally disregarded. On the other hand, the theory of time reversibility studies the conditions under which the stochastic behavior of a system remains the same when the direction of time is reversed; unfortunately, it has been applied to concurrent systems without explicitly taking causality into account. In this paper, we aim at bridging these two theories by showing how causal consistent reversibility and time reversibility can be jointly achieved. To this purpose, we consider a stochastic process calculus in which every action is equipped with a positive real number expressing the rate at which the action is executed. As is well known in the literature [17], the stochastic process underlying the calculus turns out to be a continuous-time Markov chain (CTMC) [21].

The contribution of this paper is threefold. Firstly, we apply for the first time the technique of [39] to a stochastic process calculus. In particular, we provide forward and backward operational semantic rules – featuring forward and backward actions and rates – and then we show that the resulting calculus is causal consistent reversible. The latter is accomplished by importing from the reduction semantics setting of [8] the notion of concurrent transitions, which is new in the structural operational semantics framework of [39] and is then managed by following the approach of [30].

Secondly, after observing that the CTMC underlying the calculus is stationary, we prove that time reversibility can be obtained by using, in the operational semantic rules, backward rates equal to the corresponding forward rates. This is quite different from the approach adopted for example in [14, 35], where time reversibility is verified a posteriori, as we instead produce a calculus in which time reversibility can be guaranteed by construction. The difference in the approach does not prevent us from importing compositionality results of those works in our setting.

Thirdly, we address behavioral equivalences for our reversible stochastic process calculus, in order to provide a means to identify systems possessing the same functional and performance properties. In particular, we focus on Markovian bisimilarity [17], which equates systems capable of stepwise mimicking each other’s functional and performance behavior. We extend a result of [10] to our stochastic setting by showing that a variant of Markovian bisimilarity on computations accounting for both forward and backward directions coincides with Markovian bisimilarity on states, thus inheriting the nice properties of the latter.

This paper, which is a revised and enriched version of [5], is organized as follows. In Sections 2 and 3 we recall background information about causal consistent reversibility and time reversibility, respectively. In Section 4 we develop our proposal of integration for these two forms of reversibility in the setting of a Markovian process calculus, which is then equipped in Section 5 with a forward and backward Markovian bisimilarity. Finally, in Section 6 we conclude with some directions for future work.

2. Causal Consistent Reversibility

Reversibility in a computing system implies the possibility of reverting the last performed action. In a sequential system, this is very simple as there exists just one last action.
Hence, the only challenge is how to store the information needed to reverse this last action. Since Bennett [3], several techniques have been developed to reverse the computations of a sequential program [32, 47, 15, 37].

In a concurrent system, the situation is more complex as there is no clear definition of last action. Indeed, there might be several concurrent last actions. One could resort to timestamps to decide which action is the last one, but having synchronized clocks in a distributed system is rather difficult. A good approximation is to consider as last action each action that has not caused any other action yet. This is at the basis of the so called causal consistent reversibility [8], which relates reversibility with causality. Intuitively, the definition says that, in a concurrent system, any action can be undone provided that all of its consequences, if any, are undone beforehand.

In the process algebra literature, specifically for CCS [36], two approaches have been developed to reverse a computation and to keep track of past actions: the dynamic approach of [8] and the static approach of [39]. The former relies on RCCS [8], a variant of CCS that uses stack-based memories attached to processes to record all the actions executed by the processes themselves. In contrast, [39] proposes a general method, of which CCSK is a result, to reverse calculi whose operational semantic rules are expressed in the path format of [1]. The basic idea of this method is to make all the operators of the calculus static and to univocally identify each executed action with a communication key.

Despite these two approaches are quite different, they have been recently shown to be equivalent in terms of labeled transition system isomorphism [26]. On the application side, the dynamic approach of [8] is more suitable for systems whose operational semantics is given in terms of reduction semantics, hence it is to be preferred in the case of very expressive calculi [27, 7, 25] as well as programming languages [34, 29]. On the other hand, the static approach of [39] is very handy when it comes to deal with labeled transition systems and CCS-like calculi, which is the case of this paper.

As mentioned above, since dynamic process algebraic operators – like for instance action prefix “.” and choice “+” – disappear after a reduction thus causing information loss, in [39] all of them are made static. For example, in the following CCS reduction:

\[ a.P + b.Q \xrightarrow{a} P \]

both the information about \( a \) and the right branch \( + b.Q \) are lost after performing the reduction. Hence, in order to get back to the initial state, the system has to remember both the executed prefix \( a \) and the discarded process \( b.Q \). In [39] the dynamic operators are syntactically maintained in the term, so that the process \( a.P + b.Q \) reduces as follows:

\[ a.P + b.Q \xrightarrow{a[i]} a[i].P + b.Q \]

where \( a[i].P + b.Q \) behaves like \( P \) in further forward reductions, while the colored parts can be seen as decorations of \( P \) to be used only in backward reductions. Alternatively, \( a[i].P + b.Q \) can be seen as \( C[P] \), where \( P \) is the active part and \( C = a[i].\bullet + b.Q \) is its decorated context. In this way, the use of explicit memories of [8] is avoided because the necessary information is syntactically maintained within processes. Note that action \( a \) is decorated with key \( [i] \) so as to distinguish among several actions executed in the past that have the same name.
3. Time Reversibility

In the performance evaluation field, a different notion of reversibility, called time reversibility, is considered. We illustrate it in the specific case of continuous-time Markov chains, which are discrete-state stochastic processes characterized by the memoryless property [21].

A stochastic process describes the evolution of some random phenomenon over time through a set of random variables, one for each time instant. A stochastic process \( X(t) \) taking values into a discrete state space \( S \) for \( t \in \mathbb{R}_{\geq 0} \) is a continuous-time Markov chain (CTMC) iff for all \( n \in \mathbb{N} \), time instants \( t_0 < t_1 < \cdots < t_n < t_{n+1} \in \mathbb{R}_{\geq 0} \), and states \( s_0, s_1, \ldots, s_n, s_{n+1} \in S \) it holds that

\[
\Pr\{X(t_{n+1}) = s_{n+1} \mid X(t_i) = s_i, 0 \leq i \leq n\} = \Pr\{X(t_{n+1}) = s_{n+1} \mid X(t_n) = s_n\},
\]

i.e., the probability of moving from one state to another does not depend on the particular path that has been followed in the past to reach the current state, hence that path can be forgotten.

A CTMC is irreducible iff each of its states is reachable from every other state. A state \( s \in S \) is recurrent iff the CTMC will eventually return to \( s \) with probability 1, in which case \( s \) is called positive recurrent iff the expected number of steps until the CTMC returns to it is finite. A CTMC is ergodic iff it is irreducible and all of its states are positive recurrent; ergodicity coincides with irreducibility in the case that the CTMC has finitely many states.

A CTMC can be represented as a labeled transition system or as a state-indexed matrix. In the first case, each transition is labeled with some probabilistic information describing the evolution from its source state to its target state. In the second case, the same information is stored into an entry, indexed by those two states, of a matrix. The value of this probabilistic information is, in general, a function of the time at which the state change takes place.

For the sake of simplicity, we restrict ourselves to time-homogeneous CTMCs, in which conditional probabilities of the form \( \Pr\{X(t + t') = s' \mid X(t) = s\} \) do not depend on \( t \), so that the considered information is simply a positive real number. This is called the rate at which the CTMC moves from state \( s \) to state \( s' \) and uniquely characterizes the exponentially distributed time taken by the considered move. It can be shown that the sojourn time in any state \( s \in S \) is exponentially distributed with rate given by the sum of the rates of the moves of \( s \). The average sojourn time in \( s \) is the inverse of such a sum and the probability of moving from \( s \) to \( s' \) is proportional to the corresponding rate.

Every time-homogeneous, ergodic CTMC \( X(t) \) is stationary, which means that \( (X(t_i + t'))_{1 \leq i \leq n} \) has the same joint distribution as \( (X(t_j))_{1 \leq j \leq n} \) for all \( n \in \mathbb{N}_{\geq 1} \) and \( t_1 < \cdots < t_n, t' \in \mathbb{R}_{\geq 0} \). Specifically, \( X(t) \) has a unique steady-state probability distribution \( \pi \) that for all \( s \in S \) fulfills

\[
\pi(s) = \lim_{t \to \infty} \Pr\{X(t) = s \mid X(0) = s'\}
\]

for any \( s' \in S \). These probabilities can be computed by solving the linear system of global balance equations \( \pi \cdot Q = 0 \) subject to \( \sum_{s \in S} \pi(s) = 1 \) and \( \pi(s) \in \mathbb{R}_{>0} \) for all \( s \in S \). The infinitesimal generator matrix \( Q \) contains for each pair of distinct states the rate of the corresponding move, which is 0 in the absence of a direct move between them, with \( q_{s,s'} = -\sum_{s' \neq s} q_{s,s'} \) for all \( s \in S \) so that every row of \( Q \) sums up to 0.

Due to state space explosion and numerical stability problems [45], the calculation of the solution of the global balance equation system is not always feasible. However, it can be tackled in the case that the behavior of the considered CTMC remains the same when the direction of time is reversed. A CTMC \( X(t) \) is time reversible iff \( (X(t_i))_{1 \leq i \leq n} \) has the same joint distribution as \( (X(t' - t_i))_{1 \leq i \leq n} \) for all \( n \in \mathbb{N}_{\geq 1} \) and \( t_1 < \cdots < t_n, t' \in \mathbb{R}_{\geq 0} \), in which case \( X(t) \) and its reversed version \( X^\tau(t) = X(t' - t) \) are stochastically identical; in particular, \( X(t) \) and \( X^\tau(t) \) share the same steady-state probability distribution \( \pi \) if any.
In order for a stationary CTMC $X(t)$ to be time reversible, it is necessary and sufficient that the partial balance equations $\pi(s) \cdot q_{s,s'} = \pi(s') \cdot q_{s',s}$ are satisfied for all $s, s' \in S$ such that $s \neq s'$ or, equivalently, that $q_{s_1,s_2} \cdots q_{s_{n-1},s_n} \cdot q_{s_n,s_1} \cdots q_{s_2,s_1} = q_{s_1,s_n} \cdots q_{s_n,s_{n-1}} \cdots q_{s_2,s_1}$ for all $n \in \mathbb{N}_{\geq 2}$ and distinct $s_1, \ldots, s_n \in S$ [20].

Time reversibility of CTMC-based compositional models of concurrent systems has been investigated in [14]. More precisely, conditions relying on the conservation of total exit rates of states and of rate products around cycles are examined, which support the hierarchical and compositional reversal of stochastic process algebra terms. These conditions also lead to the efficient calculation of steady-state probability distributions in a product form typical of queueing theory [22], thus avoiding the need of solving the global balance equation system. More recently, in [35] similar conditions have been employed to characterize the class of $\rho$-reversible stochastic automata. Under certain constraints, the joint steady-state probability distribution of the composition of two such automata is the product of the steady-state probability distributions of the two automata.

4. Integrating Causal and Time Reversibility

In this section, we integrate the two concepts of causal consistent reversibility and time reversibility recalled in the previous two sections. We start with a simple calculus called RMPC – Reversible Markovian Process Calculus, in which actions are paired with rates, whose syntax and semantics are inspired by [39]. Then, we show that the reversibility induced by RMPC is causal consistent by using the notion of concurrent transitions of [8] and the technique of [30]. Finally, we exhibit the conditions under which time reversibility is achieved too and we compare our setting, in which time reversibility is ensured by construction, with those of [14, 35], from which we import a product form result.

4.1. Syntax and Semantics for RMPC. The syntax of RMPC is shown in Table 1. A forward process $P$ can be one of the following: the idle process $0$; the prefixed process $(a, \lambda).P$, which is able to perform an action $a$ at rate $\lambda$ – called an exponentially timed action as its duration follows an exponential distribution of parameter $\lambda$ – and then continues as process $P$; the nondeterministic choice $P + Q$ between processes $P$ and $Q$; or the parallel composition $P \parallel_L Q$, indicating that processes $P$ and $Q$ execute in parallel and must synchronize only on actions prescribed by the set $L$.

A reversible process $R$ is built on top of forward processes. As in [39], the syntax of reversible processes differs from the one of forward processes by the fact that in the former each prefix $(a, \lambda)$ can be decorated with a communication key $i$ thus becoming $(a, \lambda)[i]$. A process of the form $(a, \lambda)[i].R$ expresses that in the past the process synchronized with the environment and this synchronization was identified by key $i$. Keys are thus attached only to already executed actions.

Let $A$ be a set of actions (ranged over by $a, b$), $\mathcal{R} = \mathbb{R}_{>0}$ be a set of rates (ranged over by $\lambda, \mu$), and $\mathcal{K}$ be a set of keys (ranged over by $i, j$). Let $\mathcal{L} = A \times \mathcal{R} \times \mathcal{K}$ be a set.
of labels each formed by an action, a rate, and a communication key. We let \( \ell \) and its decorated version range over \( \mathcal{L} \). Moreover, given a forward label \( \ell = (a, \lambda)[i] \), we denote by \( \overline{\ell} = (a, \lambda)[i] \) the corresponding backward label. Finally, \( \mathcal{P} \) is the set of processes generated by the production for \( R \) in Table 1.

**Definition 4.1** (standard process). Process \( R \in \mathcal{P} \) is standard, written \( \text{std}(R) \), iff it can be derived from the production for \( P \) in Table 1.

**Definition 4.2** (process key set). The set of keys of process \( R \in \mathcal{P} \), written \( \text{key}(R) \), is inductively defined as follows:

\[
\text{key}(P) = \emptyset \\
\text{key}((a, \lambda)[i].R) = \{i\} \cup \text{key}(R) \\
\text{key}(R + S) = \text{key}(R) \cup \text{key}(S) \\
\text{key}(R \parallel L.S) = \text{key}(R) \cup \text{key}(S)
\]

The semantics for RMPC is defined as a labeled transition system \((\mathcal{P}, \mathcal{L}, \rightarrow)\). Like in [39], the transition relation \( \rightarrow \subseteq \mathcal{P} \times \mathcal{L} \times \mathcal{P} \) is given by \( \rightarrow \cup \sim \), where the forward transition relation \( \rightarrow \) and the backward transition relation \( \sim \) are the least relations respectively induced by the forward rules in the left part of Table 2 and the backward rules in the right part of the same table.

Rule Act1 deals with prefixed processes of the form \((a, \lambda).P\), with \( P \) written as \( R \) subject to \( \text{std}(R) \). In addition to transforming the action prefix into a transition label, it generates a key \( i \) that is bound to the action \((a, \lambda)\) thus yielding the label \((a, \lambda)[i]\). As can be noted, the prefix is not discarded by the application of the rule, instead it becomes a key-storing decoration in the target process. Rule Act1* reverts the action \((a, \lambda)[i]\) of
the process \((a, \lambda)[i].R\) provided that \(R\) is a standard process, which ensures that \((a, \lambda)[i]\) is the only past action that is left to undo. One of the main design choices of the entire framework is how the rate \(\bar{\lambda}\) of the backward action is calculated. For the time being, we leave it unspecified in \(\text{Act1}^*\) as the value of this rate is not necessary to prove the causal consistency part of reversibility; we will see later on that it is important in the proof of time reversibility.

The presence of rules \(\text{Act2}^*\) is motivated by the fact that rule \(\text{Act1}\) does not discard the executed prefix from the process it generates. In particular, rule \(\text{Act2}\) allows a prefixed process \((a, \lambda)[i].R\) to execute if \(R\) can itself execute provided that the action performed by \(R\) picks a key \(j\) different from \(i\), so that all action prefixes in a sequence are decorated with distinct keys. Rule \(\text{Act2}^*\) simply propagates the execution of backward actions from inner subprocesses that are not standard as long as key uniqueness is preserved, so that past actions are overall undone from the most recent one to the least recent one.

Unlike the classical rules of the choice operator \([36]\), rule \(\text{Cho}\) does not discard the context, i.e., the part of the process that has not contributed to the action. More in detail, if the process \(R\) does an action, say \((a, \lambda)[i]\), and becomes \(R'\), then the entire process \(R + S\) becomes \(R' + S\). In this way, the information about \(+ S\) is preserved. Furthermore, since \(S\) is a standard process because of the premise \(\text{std}(S)\), it will never execute even if it is present in the process \(R' + S\). Hence, the \(+ S\) can be seen as a decoration, or a dead context, of process \(R\). Note that, in order to apply rule \(\text{Cho}\), at least one of the two processes has to be standard, meaning that it is impossible for two non-standard processes to execute if they are composed by a choice operator. Rule \(\text{Cho}^*\) has precisely the same structure as rule \(\text{Cho}\), but uses the backward transition relation. For both rules, we omit their symmetric variants in which it is \(S\) to move.

The semantics of parallel composition is inspired by \([18]\). Rule \(\text{Par}\) allows process \(R\) within \(R \parallel_L S\) to individually perform an action \((a, \lambda)[i]\) provided that \(a \notin L\). It is also checked that the executing action is bound to a fresh key, thus ensuring the uniqueness of communication keys across parallel composition too. Rule \(\text{Coo}\) allows \(R\) and \(S\) to synchronize through any action in the set \(L\), provided that the communication key is the same on both sides. For the sake of simplicity, the rate of the cooperation action is assumed to be the product of the rates of the two involved actions \([16]\). Rules \(\text{Par}^*\) and \(\text{Coo}^*\) respectively have the same structure as \(\text{Par}\) and \(\text{Coo}\); the symmetric variants of \(\text{Par}\) and \(\text{Par}^*\) are omitted.

Not all the processes generated by the grammar in Table 1 are meaningful, i.e., respectful of communication key uniqueness enforced by the rules in Table 2. Therefore, in the rest of the paper we only consider processes that are initial or reachable in the following sense.

**Definition 4.3** (initial process). Process \(R \in \mathcal{P}\) is initial iff \(\text{std}(R)\) holds. ■

**Definition 4.4** (reachable process). Process \(R \in \mathcal{P}\) is reachable iff it is initial or can be derived from an initial one via finitely many applications of the rules for \(\rightarrow\) in Table 2. ■

### 4.2. Preliminary Reversibility Properties.

A basic property to satisfy in order for RMPC to be reversible is the so called loop lemma \([8, 39]\), which will be exploited to establish both causal consistent reversibility and time reversibility. This property states that each transition of a reachable process can be undone, as formalized below.
Lemma 4.5 (loop lemma). Let \( R \in \mathcal{P} \) be a reachable process. Then \( R \xrightarrow{(a, \lambda)[i]} S \) iff \( S \xrightarrow{\rho} R \).

Proof. It is easily proved by induction on the depth of the derivation of \( R \xrightarrow{(a, \lambda)[i]} S \) (resp., \( S \xrightarrow{\rho} R \)), by noting that for each forward (resp., backward) rule there exists a corresponding backward (resp., forward) one.

Given a sequence \( \sigma \) of \( n \in \mathbb{N}_{>0} \) labels \( \ell_1, \ldots, \ell_n \), let \( R \xrightarrow{\sigma} S \) be the corresponding forward sequence of transitions \( R \overset{\ell_1}{\rightarrow} R_1 \overset{\ell_2}{\rightarrow} \cdots \overset{\ell_n}{\rightarrow} S \) and \( \bar{\sigma} \) be the corresponding backward sequence such that, for each \( \ell_i \) occurring in \( \sigma \), it holds that \( R_{i-1} \overset{\ell_i}{\rightarrow} R_i \) iff \( R_i \overset{\ell_i}{\rightarrow} R_{i-1} \). The loop lemma generalizes as follows.

Corollary 4.6. Let \( R \in \mathcal{P} \) be a reachable process. Then \( R \xrightarrow{\sigma} S \) iff \( S \xrightarrow{\bar{\sigma}} R \). 

4.3. Causal Consistent Reversibility for RMPC. In order to prove the causal consistent reversibility of RMPC, we borrow some machinery from [8] that needs to be adapted as the reversing method of [39] we are using is different from the one of [8]. In particular, we import the notion of concurrent transitions.

Given a transition \( \theta : R \overset{\ell}{\rightarrow} S \) with \( R, S \in \mathcal{P} \) reachable processes, we call \( R \) the source of \( \theta \) and \( S \) its target. If \( \theta \) is a forward transition, i.e., \( \theta : R \overset{\ell}{\rightarrow} S \), we denote the corresponding backward transition \( S \overset{\ell}{\rightarrow} R \) as \( \bar{\theta} \). Two transitions are said to be coinital if they have the same source, and cofinal if they have the same target. A sequence of pairwise composable transitions is called a computation, where composable means that the target of any transition in the sequence is the source of the next transition. We let \( \omega \) and its decorated version range over computations, with \( |\omega| \) denoting the length of \( \omega \) expressed as the number of transitions constituting it. The notions of source, target, and compositibility extend naturally to computations. We indicate with \( \epsilon \) the empty computation and with \( \omega_1 \omega_2 \) the composition of the two computations \( \omega_1 \) and \( \omega_2 \) when they are composable.

Before specifying when two transitions are concurrent, we need to define the set of causes – identified by keys – that lead to a given communication key along with the notion of process context.

Definition 4.7 (causal set). Let \( R \in \mathcal{P} \) be a reachable process and \( i \in \text{key}(R) \). The causal set \( \text{cau}(R, i) \) is inductively defined as follows for \( j \neq i \):

\[
\begin{align*}
\text{cau}((a, \lambda)[i], R, i) &= \emptyset \\
\text{cau}((a, \lambda)[j], R, i) &= \{j\} \cup \text{cau}(R, i) \\
\text{cau}(R + S, i) &= \text{cau}'(R, i) \cup \text{cau}'(S, i) \\
\text{cau}(R \parallel_L S, i) &= \text{cau}'(R, i) \cup \text{cau}'(S, i)
\end{align*}
\]

where \( \text{cau}'(R, i) = \text{cau}(R, i) \) if \( i \in \text{key}(R) \) and \( \text{cau}'(R, i) = \emptyset \) otherwise.

If \( i \in \text{key}(R) \), then \( \text{cau}(R, i) \) represents the set of keys in \( R \) that caused \( i \), with \( \text{cau}(R, i) \subseteq \text{key}(R) \) because \( i \notin \text{cau}(R, i) \) and keys not causally related to \( i \) are not considered. A key \( j \) causes \( i \) if it appears syntactically before \( i \) in \( R \) or, said otherwise, \( i \) is inside the scope of \( j \).
Definition 4.8 (process context). A process context \( \mathcal{C} \) is a process with a hole \( \bullet \) generated by the grammar \( \mathcal{C} ::= \bullet \mid (\alpha, \lambda)[i] \mathcal{C} \mid R + \mathcal{C} \mid C + R \mid R \parallel L \mathcal{C} \mid \mathcal{C} \parallel L R. \)

We are now in a position to define what we mean by concurrent transitions.

Definition 4.9 (concurrent transitions). Two coinitial transitions \( \theta_1 \) and \( \theta_2 \) from a reachable process \( R \in \mathcal{P} \) are in conflict iff one of the following two conditions holds:

1. \( \theta_1 : R \xrightarrow{(\alpha, \lambda)[i]} S_1 \) and \( \theta_2 : R \xrightarrow{(b, \mu)[j]} S_2 \) with \( j \in \text{cau}(S_1, i) \).
2. \( R = C[P_1 + P_2] \) with \( \theta_k \) deriving from \( P_k \xrightarrow{(a_k, \lambda_k)[i_k]} S_k \) for \( k = 1, 2 \).

Two coinitial transitions are concurrent when they are not in conflict.

Remark 4.10. It is worth noting that in a stochastic process calculus like RMPC the semantic treatment of the choice operator is problematic [17] because a process of the form \( (\alpha, \lambda).P + (a, \lambda).P \) should produce either a single \( a \)-transition whose rate is \( \lambda + \lambda \), or two \( a \)-transitions each having rate \( \lambda \) that do not collapse into a single one. In our reversible framework, two distinct transitions are generated thanks to the fact that the key associated with the executed action is stored into the derivative process too, as shown in the bottom part of Figure 1.

Concurrent transitions can commute, as formally stated below by the diamond property, while conflicting ones cannot.

Lemma 4.11 (diamond property). Let \( \theta_1 : R \xrightarrow{\ell_1} S_1 \) and \( \theta_2 : R \xrightarrow{\ell_2} S_2 \) be two coinitial transitions from a reachable process \( R \in \mathcal{P} \). If \( \theta_1 \) and \( \theta_2 \) are concurrent, then there exist two cofinal transitions \( \theta_2/\theta_1 : S_1 \xrightarrow{\ell_2} S \) and \( \theta_1/\theta_2 : S_2 \xrightarrow{\ell_1} S \).
Proof. The proof is by case analysis on the direction of $\theta_1$ and $\theta_2$. We distinguish three cases according to whether the two transitions are both forward, both backward, or one forward and the other backward. Suppose that $\ell_1 = (a, \lambda)[i]$ and $\ell_2 = (b, \mu)[j]$, with $i \neq j$ otherwise $\theta_1$ and $\theta_2$ would be generated by the two subprocesses of a choice operator and hence could not be concurrent:

- Suppose that $\theta_1$ and $\theta_2$ are both forward. Since $\theta_1$ and $\theta_2$ are concurrent, by virtue of Definition 4.9 the two transitions cannot originate from a choice operator. They must thus be generated by a parallel composition, but not through the CoO rule because $\theta_1$ and $\theta_2$ have different keys and hence cannot synchronize. Without loss of generality, we can assume that $R = R_1 \parallel_L R_2$ with $R_1 \xrightarrow{(a, \lambda)[i]} S_1$ and $R_2 \xrightarrow{(b, \mu)[j]} S_2$ and $a, b \notin L$. By applying the PAR rule, we have that $R_1 \parallel_L R_2 \xrightarrow{(a, \lambda)[i]} S_1 \parallel_L R_2 \xrightarrow{(b, \mu)[j]} S_1 \parallel_L S_2$ as well as $R_1 \parallel_L R_2 \xrightarrow{(b, \mu)[j]} R_1 \parallel_L S_2 \xrightarrow{(a, \lambda)[i]} S_1 \parallel_L S_2$.

- The case in which $\theta_1$ and $\theta_2$ are both backward is similar to the previous one.

- Suppose that $\theta_1$ is forward and $\theta_2$ is backward. Since $\theta_1$ and $\theta_2$ are concurrent, by virtue of Definition 4.9 the backward transition cannot remove a cause of the forward one. Since either subprocess of a choice operator 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 the choice operator only one of the two subprocesses can perform transitions, without loss of generality we can assume that $R = R_1 \parallel_L R_2$ with $R_1 \xrightarrow{(a, \lambda)[i]} S_1$ and $R_2 \xrightarrow{(b, \mu)[j]} S_2$ as well as $a, b \notin L$ so as to preserve causes. By applying the PAR rule, we have that $R_1 \parallel_L R_2 \xrightarrow{(a, \lambda)[i]} S_1 \parallel_L R_2 \xrightarrow{(b, \mu)[j]} S_1 \parallel_L S_2$ as well as $R_1 \parallel_L R_2 \xrightarrow{(b, \mu)[j]} R_1 \parallel_L S_2 \xrightarrow{(a, \lambda)[i]} S_1 \parallel_L S_2$.

We finally show that reversibility is causally consistent in our concurrent framework. This can be done in two ways: either by adapting the original proof of [8], as we did in [5], or by using the general technique provided by [30]. We opt for the latter, according to which causal consistency stems from the diamond property, backward transition independence – which generalizes the concept of backward determinism used for reversible sequential languages [47] – and past well foundedness – which ensures that reachable processes have a finite past. Following [33], we first define a notion of causal equivalence over computations, which abstracts from the order of concurrent transitions, so that computations obtained by swapping the order of their concurrent transitions are identified with each other and the composition of a computation with its inverse is identified with the empty computation.

**Definition 4.12 (causal equivalence).** Causal equivalence is the smallest equivalence $\asymp$ on computations closed under composition and satisfying:

1. If $\theta_1 : R \xrightarrow{\ell_1} R_1$, $\theta_2 : R \xrightarrow{\ell_2} R_2$ are concurrent and $\theta_2' : R_1 \xrightarrow{\ell_2} S$, $\theta_1' : R_2 \xrightarrow{\ell_1} S$, then $\theta_1 \theta_2' \asymp \theta_2 \theta_1'$.

2. $\emptyset \emptyset \asymp \epsilon$ and $\emptyset \emptyset \asymp \epsilon$.

**Lemma 4.13 (backward transition independence).** Given a reachable process $R \in \mathcal{P}$, any two coinitial backward transitions $\theta_1 : R \xrightarrow{\sim_{\lambda}[i]} S_1$ and $\theta_2 : R \xrightarrow{\sim_{\mu}[j]} S_2$ are concurrent.

**Proof.** Since by Definition 4.9 there is no case in which two backward transitions are conflicting, the property trivially holds.
**Proposition 4.14** (past well foundedness). Let \( R_0 \in \mathcal{P} \) be a reachable process. Then there is no infinite sequence such that \( R_i \stackrel{\ell_i}{\rightarrow} R_{i+1} \) for all \( i \in \mathbb{N} \).

**Proof.** It can be easily proved by induction on \( |key(R_0)| \) by observing that a backward transition decreases by one the total number of keys of \( R_0 \), with this number being finite. □

The further property below, called parabolic lemma in [30], says that if two computations \( \omega_1 \) and \( \omega_2 \) are coinitial and cofinal and \( \omega_2 \) is made of forward transitions only, then in \( \omega_1 \) there are some transitions that are later undone. This computation is thus causally equivalent to a forward one in which the undone transitions do not take place at all. Another way to see this lemma is that backward transitions do not add new states.

**Lemma 4.15** (parabolic lemma). For any computation \( \omega \), there exist two forward-only computations \( \omega_1, \omega_2 \) such that \( \omega \equiv \overline{\omega}_1 \omega_2 \) and \( |\omega_1| + |\omega_2| \leq |\omega| \).

**Proof.** It follows from the diamond property and backward transition independence thanks to [30]. □

**Theorem 4.16** (causal consistency). Let \( \omega_1 \) and \( \omega_2 \) be coinitial computations. Then \( \omega_1 \equiv \omega_2 \) iff \( \omega_1 \) and \( \omega_2 \) are cofinal too.

**Proof.** It follows from past well foundedness and the parabolic lemma thanks to [30]. □

**4.4. Time Reversibility for RMPC.** The rules in Table 2 associate with any initial process \( R \in \mathcal{P} \) a labeled transition system \( \llbracket R \rrbracket = (\mathcal{P}, L, \rightarrow, R) \). To investigate time reversibility, we have to derive from \( \llbracket R \rrbracket \) the CTMC \( \mathcal{M} \llbracket R \rrbracket \) underlying \( R \) and we have to specify how each backward rate \( \lambda \) is obtained from the corresponding forward rate \( \ell \).

First of all, we observe that every non-terminal state of \( \llbracket R \rrbracket \) has infinitely many outgoing transitions. The reason is that rules Act1 and Act2 generate a transition for each possible admissible key, with the key being part of both the label and the derivative process term. On the one hand, this is useful for avoiding the generation of a single \((a, \lambda)\) transition in the case of a process like \((a, \lambda).P + (a, \lambda).P\) whose overall exit rate is \( \lambda + \lambda \); even if the key is the same, two different states \((a, \lambda)[i].P + (a, \lambda).P\) and \((a, \lambda).P + (a, \lambda)[i].P\) are reached. On the other hand, it requires considering transition bundles to build \( \mathcal{M} \llbracket R \rrbracket \), where a transition bundle is a set of transitions departing from the same state and labeled with the same action/rate but different keys, whose target states are syntactically identical up to keys.

**Definition 4.17** (underlying CTMC). Let \( \equiv_{\mathcal{K}} \) be the least equivalence relation over \( \mathcal{P} \) induced by \((a, \lambda)[i].S \equiv_{\mathcal{K}} (a, \lambda)[j].S\). The CTMC underlying an initial process \( R \in \mathcal{P} \) is defined as the labeled transition system \( \mathcal{M} \llbracket R \rrbracket = (\mathcal{P}/\equiv_{\mathcal{K}}, \lambda \times \mathcal{R}, \rightarrow_{\mathcal{K}}, \llbracket R \rrbracket \equiv_{\mathcal{K}}) \) where:

- \( \mathcal{P}/\equiv_{\mathcal{K}} \) is the quotient set of \( \equiv_{\mathcal{K}} \) over \( \mathcal{P} \), i.e., the set of classes of processes that are equivalent to each other according to \( \equiv_{\mathcal{K}} \), representing the set of states.
- \( \llbracket R \rrbracket \equiv_{\mathcal{K}} \) is the equivalence class of \( R \) with respect to \( \equiv_{\mathcal{K}} \), which simply is the singleton set \( \{R\} \) as \( R \) is initial and hence contains no keys, representing the initial state.
\(\rightarrow_K \subseteq \mathcal{P} / \equiv_K \times (\mathcal{A} \times \mathcal{R}) \times \mathcal{P} / \equiv_K \) is the transition relation given by \(\rightarrow_K \cup \sim_K\) such that:

- \([S] \equiv_K \xrightarrow{(a, \lambda)}_K [S'] \equiv_K\) whenever \(S \xrightarrow{(a, \lambda)[i]} S'\) for some \(i \in \mathcal{K}\).
- \([S] \equiv_K \xrightarrow{\underbrace{\cdots}}_K [S'] \equiv_K\) whenever \(S \xrightarrow{\underbrace{\cdots}} S'\) for some \(i \in \mathcal{K}\).

When moving from \([R]\) to \(\mathcal{M}[R]\), individual states are thus replaced by classes of states that are syntactically identical up to keys in the same positions; moreover, keys are removed from transition labels. As a consequence, every state of \(\mathcal{M}[R]\) turns out to have finitely many outgoing transitions. We also note that \(\mathcal{M}[R]\) is an action-labeled CTMC, as each of its transitions is labeled with both a rate and an action.

As a preliminary step towards time reversibility, we have to show that \(\mathcal{M}[R]\) is stationary. It holds that \(\mathcal{M}[R]\) is even ergodic thanks to the loop lemma.

**Lemma 4.18.** Let \(R \in \mathcal{P}\) be an initial process. Then \(\mathcal{M}[R]\) is time homogeneous and ergodic.

**Proof.** The time homogeneity of \(\mathcal{M}[R]\) is a straightforward consequence of the fact that its rates simply are positive real numbers, not time-dependent functions. The ergodicity of \(\mathcal{M}[R]\) stems from the fact that the graph representation of \(\mathcal{M}[R]\) is a finite, strongly connected component due to Corollary 4.6.

We exploit once more the loop lemma to derive that, under \(\overline{\lambda} = \lambda\), the steady-state probability distribution of \(\mathcal{M}[R]\) is the uniform distribution, from which time reversibility will immediately follow.

**Lemma 4.19.** Let \(R \in \mathcal{P}\) be an initial process, \(\mathcal{S}\) be the set of states of \(\mathcal{M}[R]\), and \(n = |\mathcal{S}|\). If every backward rate is equal to the corresponding forward rate, then the steady-state probability distribution \(\pi\) of \(\mathcal{M}[R]\) satisfies \(\pi(s) = 1/n\) for all \(s \in \mathcal{S}\).

**Proof.** If \(n = 1\), i.e., \(R\) is equal to \(0\) or to the parallel composition of several processes whose initial actions have to synchronize but are different from each other, then trivially \(\pi(s) = 1/n = 1\) for the only state \(s \in \mathcal{S}\).

Suppose now that \(n \geq 2\). From Lemma 4.18, it follows that \(\mathcal{M}[R]\) has a unique steady-state probability distribution \(\pi\). Due to Lemma 4.5, the global balance equation for an arbitrary \(s \in \mathcal{S}\) is as follows:

\[
\pi(s) \cdot \sum_{s' \xrightarrow{(a, \lambda)}_K s} \lambda = \sum_{s' \xrightarrow{(a, \lambda)}_K s} \pi(s') \cdot \overline{\lambda}
\]

Since every backward rate is equal to the corresponding forward rate, the global balance equation for \(s\) boils down to:

\[
\pi(s) \cdot \sum_{s' \xrightarrow{(a, \lambda)}_K s'} \lambda = \sum_{s' \xrightarrow{(a, \lambda)}_K s'} \pi(s') \cdot \lambda
\]

Since the two summations have the same number of summands, the equation above is satisfied when \(\pi(s) = \pi(s')\) for all \(s' \in \mathcal{S}\) reached by a transition from \(s\). All global balance equations are thus satisfied when \(\pi(s) = 1/n\) for all \(s \in \mathcal{S}\).

**Theorem 4.20** (time reversibility). Let \(R \in \mathcal{P}\) be an initial process. If every backward rate is equal to the corresponding forward rate, then \(\mathcal{M}[R]\) is time reversible.

**Proof.** Let \(\mathcal{S}\) be the set of states of \(\mathcal{M}[R]\) and \(n = |\mathcal{S}|\). From Lemma 4.18, it follows that \(\mathcal{M}[R]\) has a unique steady-state probability distribution \(\pi\). To avoid trivial cases, suppose \(n \geq 2\) and consider \(s, s' \in \mathcal{S}\) with \(s \neq s'\) connected by a transition. The proof resembles the
one of Lemma 4.19, but focuses on partial balance equations rather than on global ones. Due to Lemma 4.5, the partial balance equation for $s$ and $s'$ is as follows:

$$\pi(s) \cdot \sum_{s \xrightarrow{(a,\lambda)} Ks'} \lambda = \pi(s') \cdot \sum_{s' \xrightarrow{(a,\lambda)} Ks} \lambda$$

Since every backward rate is equal to the corresponding forward rate, the partial balance equation for $s$ and $s'$ boils down to:

$$\pi(s) \cdot \sum_{s \xrightarrow{(a,\lambda)} Ks'} \lambda = \pi(s') \cdot \sum_{s' \xrightarrow{(a,\lambda)} Ks} \lambda$$

Since the two summations have the same number of summands and $\pi(s) = \pi(s') = \frac{1}{n}$ due to Lemma 4.19, the equation above is satisfied. The result then follows from the fact that $s$ and $s'$ are two arbitrary distinct states connected by transitions.

The main difference between our approach to time reversibility and previous ones in the field of formal methods, in particular those of [14, 35] is twofold. Firstly, our approach is part of a more general framework in which also causal consistent reversibility is addressed. Secondly, our approach is inspired by the idea of [39] of developing a formalism in which it is possible to express models whose reversibility is guaranteed by construction, instead of building a posteriori the time-reversed version of a certain model like in [14] or verifying a posteriori whether a given model is time reversible or not like in [35].

It is worth noting that these methodological differences do not prevent us from adapting to our setting some results from [14, 35], although a few preliminary observations about notational differences are necessary.

Both [14] and [35] make a distinction between active actions, each of which is given a rate, and passive actions, each of which is given a weight, with the constraint that, in case of synchronization, the rate of the active action is multiplied by the weight of the corresponding passive action. In RMPC there is no such distinction, however the same operation, i.e., multiplication, is applied to the rates of two synchronizing actions. A passive action can thus be rendered as an action with rate 1, while a set of alternative passive actions can be rendered as a set of actions whose rates sum up to 1. Moreover, in [35] synchronization is enforced between any active-passive pair of identical actions, whereas in RMPC the parallel composition operator is enriched with an explicit synchronization set $L$, which yields as a special case the aforementioned synchronization discipline when $L$ is equal to the set $A$ of all the actions. We can therefore conclude that our parallel composition operator is a generalization of those used in [14, 35], hence the recalled notational differences do not hamper result transferral.

In [14] the compositionality of a CTMC-based stochastic process calculus is exploited to prove RCAT, the reversed compound agent theorem, which establishes the conditions under which the time-reversed version of the cooperation of two processes is equal to the cooperation of the time-reversed versions of those two processes. The application of RCAT leads to product-form solution results from stochastic process algebraic models, including a new different proof of Jackson theorem for product-form queueing networks [19].

In [35] the notion of $\rho$-reversibility is introduced for stochastic automata, which are essentially action-labeled CTMCs. Function $\rho$ is a state permutation that ensures (i) for each action the equality of the total exit rate of any state $s$ and $\rho(s)$ and (ii) the conservation of action-related rate products across cycles when considering states in the forward direction and their $\rho$-counterparts in the backward direction. For any ergodic $\rho$-reversible automaton, it turns out that $\pi(s) = \pi(\rho(s))$ for every state $s$. Moreover, the synchronization inspired
by \cite{42} of two \(\rho\)-reversible stochastic automata is still \(\rho\)-reversible and, in case of ergodicity, under certain conditions the steady-state probability of any compound state is the product of the steady-state probabilities of the two constituent states.

Our time reversibility result for RMPC can be rephrased in the setting of \cite{35} in terms of \(\rho\)-reversibility with \(\rho\) being the identity function over states. As a consequence, the following two results stem from Theorem 4.20 of the present paper and, respectively, Theorems 2 and 3 of \cite{35}.

**Corollary 4.21 (time reversibility closure).** Let \(R_1, R_2 \in \mathcal{P}\) be two initial processes and \(L \subseteq A\). If every backward rate is equal to the corresponding forward rate, then \(\mathcal{M}[R_1 \| R_2]\) is time reversible too.

**Corollary 4.22 (product form).** Let \(R_1, R_2 \in \mathcal{P}\) be two initial processes and \(L \subseteq A\). If every backward rate is equal to the corresponding forward rate and the set of states \(S\) of \(\mathcal{M}[R_1 \| R_2]\) is equal to \(S_{R_1} \times S_{R_2}\) where \(S_{R_i}\) is the set of states of \(\mathcal{M}[R_i]\) and \(S_{R_2}\) is the set of states of \(\mathcal{M}[R_2]\), then \(\pi(s_1, s_2) = \pi_R(1) \cdot \pi_R(2)\) for all \((s_1, s_2) \in S_{R_1} \times S_{R_2}\).

The product form result above avoids the calculation of the global balance equations over \(\mathcal{M}[R_1 \| R_2]\), as \(\pi(s_1, s_2)\) can simply be obtained by multiplying \(\pi_R(1)\) with \(\pi_R(2)\). However, the condition \(S = S_{R_1} \times S_{R_2}\) requires to check that every state in the full Cartesian product is reachable from \(R_1 \| R_2\). This means that no compound state is such that its constituent states enable some action but none of the enabled actions can be executed due to the constraints imposed by the synchronization set \(L\). The condition \(S = S_{R_1} \times S_{R_2}\) implies that \(\mathcal{M}[R_1 \| R_2]\) is ergodic over the full Cartesian product of the two original state spaces, which is the condition used in \cite{35}. Although implicit in the statement of the corollary, the time reversibility of \(\mathcal{M}[R_1 \| R_2]\) is essential for the product form result.

5. **Forward and Backward Markovian Bisimilarity**

Behavioral equivalences provide a means to identify structurally different systems that expose the same observable behavior. A central notion for stochastic process calculi is Markovian bisimilarity \cite{17}. It equates systems capable of stepwise mimicking each other’s functional and performance behavior and enjoys nice properties in terms of compositional reasoning and equational and logical characterizations. For the sake of uniformity, in this section we work at the state space level, rather than at the linguistic level, by considering an action-labeled CTMC \((S, A \times R, \rightarrow)\), where \(A\) is a set of actions while \(R = \mathbb{R}_{>0}\) is a set of rates, which in the case of RMPC is obtained by collecting transitions into bundles as formalized in Definition 4.17.

**Definition 5.1 (Markovian bisimilarity).** An equivalence relation \(\mathcal{B}\) over the set of states \(S\) is a Markovian bisimulation iff, whenever \((s_1, s_2) \in \mathcal{B}\), then for all actions \(a \in A\) and equivalence classes \(C \in S/\mathcal{B}\):

\[
\text{rate}(s_1, a, C) = \text{rate}(s_2, a, C)
\]

where \(\text{rate}(s, a, C) = \sum\{ \lambda \in R | \exists s' \in C, s \xrightarrow{(a, \lambda)} s' \}\). Two states \(s_1, s_2 \in S\) are Markovian bisimilar, written \(s_1 \sim_{\mathcal{B}} s_2\), iff there exists a Markovian bisimulation \(\mathcal{B}\) such that \((s_1, s_2) \in \mathcal{B}\).

Following \cite{39}, we may adapt Markovian bisimilarity to our reversible calculus by means of two conditions like the one in the definition above, with the former referring to forward
transitions and the latter referring to backward transitions. By so doing, we would end up with a very restrictive equivalence, as for instance \((a, \lambda) \cdot 0 \parallel_0 (b, \mu) \cdot 0\) and \((a, \lambda) \cdot 0 + (b, \mu) \cdot (a, \lambda) \cdot 0\) would be told apart. The reason is that in the former process from \((a, \lambda)[i] \cdot 0 \parallel_0 (b, \mu)[j] \cdot 0\) both a backward \(a\)-transition and a backward \(b\)-transition are enabled, whilst in the latter process from \((a, \lambda)[i] \cdot 0 + (b, \mu) \cdot (a, \lambda)[j] \cdot 0\) only a backward \(b\)-transition is enabled as well as from \((a, \lambda) \cdot 0 + (b, \mu)[i] \cdot (a, \lambda)[j] \cdot 0\) only a backward \(a\)-transition is enabled. In other words, the so-called expansion law [36], which transforms a parallel composition into a choice among all of its possible interleaved computations, would not hold, because the two transitions in the former process are concurrent and hence when going backward there is no obligation to follow the path traversed in the forward direction, whereas this is not the case with the latter process.

As recognized in [10], in order to set up a more useful equivalence, it is necessary to enforce that, when going backward, a process can only move along the path representing the history that brought the process to the current state. To accomplish this, bisimilarity has to be defined as a relation over paths, not over states, hence we start by adapting the notation of the nondeterministic setting of [10] to our stochastic setting. In particular, we suitably extend the notion of transition.

**Definition 5.2 (path).** A sequence \(\xi = (s_0, (a_1, \lambda_1), s_1)(s_1, (a_2, \lambda_2), s_2) \ldots (s_{n-1}, (a_n, \lambda_n), s_n) \in \rightarrow^*\) is called a path from state \(s_0\) of length \(n\). We let \(\text{first}(\xi) = s_0\) and \(\text{last}(\xi) = s_n\); the empty path is indicated with \(\epsilon\). We denote by \(\text{path}(s)\) the set of paths from state \(s\).

**Definition 5.3 (run).** A pair \(\rho = (s, \xi)\) is called a run from state \(s\) iff \(\xi \in \text{path}(s)\), in which case we let \(\text{path}(\rho) = \xi\), \(\text{first}(\rho) = \text{first}(\xi)\), \(\text{last}(\rho) = \text{last}(\xi)\), with \(\text{first}(\rho) = \text{last}(\rho) = s\) when \(\xi = \epsilon\). We denote by \(\text{run}(s)\) the set of runs from state \(s\). Given \(\rho = (s, \xi) \in \text{run}(s)\) and \(\rho' = (s', \xi') \in \text{run}(s')\), their composition \(\rho \rho' = (s, \xi \xi') \in \text{run}(s)\) is defined iff \(\text{last}(\rho) = \text{first}(\rho')\). We write \(\rho \xrightarrow{(a, \lambda)} \rho'\) iff there exists \(\rho'' = (s, (s, (a, \lambda), s'))\) with \(s = \text{last}(\rho)\) such that \(\rho' = \rho \rho''\).

In the following, for the action-labeled CTMC \((\mathcal{S}, \mathcal{A} \times \mathcal{R}, \rightarrow)\) we consider the set \(\mathcal{U}\) of its runs in lieu of \(\mathcal{S}\). Furthermore, we view the transition relation \(\rightarrow\) as being symmetric with respect to source and target states, so that every transition can be traversed in both directions. In the setting of RMPC, this amounts to considering only the forward transition relation thanks to Lemma 4.5. Given a run \(\rho\), an action \(a\), and a bisimulation equivalence class \(C\), based on [10] we distinguish between the total rate of outgoing and incoming run transitions, respectively, when moving between \(\rho\) and \(C\) via \(a\). Forward and backward Markovian bisimilarity thus relies on checking both outgoing rate equality and incoming rate equality.

**Definition 5.4 (forward and backward Markovian bisimilarity).** An equivalence relation \(\mathcal{B}\) over the set of runs \(\mathcal{U}\) is a forward and backward Markovian bisimulation iff, whenever \((\rho_1, \rho_2) \in \mathcal{B}\), then for all actions \(a \in \mathcal{A}\) and equivalence classes \(C \in \mathcal{U}/\mathcal{B}\):

\[
\begin{align*}
\text{rate}_0(\rho_1, a, C) &= \text{rate}_0(\rho_2, a, C) \\
\text{rate}_1(\rho_1, a, C) &= \text{rate}_1(\rho_2, a, C)
\end{align*}
\]

where \(\text{rate}_0(\rho, a, C) = \sum \{\lambda \in \mathcal{R} \mid \exists \rho' \in C. \rho \xrightarrow{(a, \lambda)} \rho'\}\) and \(\text{rate}_1(\rho, a, C) = \sum \{\lambda \in \mathcal{R} \mid \exists \rho' \in C. \rho' \xrightarrow{(a, \lambda)} \rho\}\). Two states \(s_1, s_2 \in \mathcal{S}\) are forward and backward Markovian bisimilar, written \(s_1 \sim_{\text{FBMB}} s_2\), iff there exists a forward and backward Markovian bisimulation \(\mathcal{B}\) such that \(((s_1, \epsilon), (s_2, \epsilon)) \in \mathcal{B}\).
Theorem 5.5. Let $s_1, s_2 \in S$. Then $s_1 \sim_{\text{FBMB}} s_2 \implies s_1 \sim_{\text{MB}} s_2$.

Proof. Suppose that $s_1 \sim_{\text{FBMB}} s_2$, with $\mathcal{B}$ being a forward and backward Markovian bisimulation on $\mathcal{U}$ such that $((s_1, \epsilon), (s_2, \epsilon)) \in \mathcal{B}$. We show that $\mathcal{B}' = \{(\text{last}(p_1), \text{last}(p_2)) \mid (p_1, p_2) \in \mathcal{B}\}$ is a Markovian bisimulation on $\mathcal{S}$, from which $s_1 \sim_{\text{MB}} s_2$ will follow.

Consider $((\text{last}(p_1), \text{last}(p_2)) \in \mathcal{B}'$. By definition of $\mathcal{B}'$, we have that $(p_1, p_2) \in \mathcal{B}$. Since $\mathcal{B}$ is a forward and backward Markovian bisimulation, for all $a \in \mathcal{A}$ and $C \in \mathcal{U}/\mathcal{B}$ it holds in particular that $\text{rate}_0(p_1, a, C) = \text{rate}_0(p_2, a, C)$. Since $\rho_k \xrightarrow{(a, \lambda)} \rho'_k$ iff $\text{last}(\rho_k) \xrightarrow{(a, \lambda)} \text{last}(\rho'_k)$ for $k \in \{1, 2\}$ and provided that function $\text{last}$ is lifted from runs to sets of runs – any equivalence class $C' \in \mathcal{S}/\mathcal{B}'$ is of the form $\{\text{last}(\rho)\}_{\mathcal{B}'} = \{\text{last}(\rho') \in \mathcal{S} \mid (\text{last}(\rho), \text{last}(\rho')) \in \mathcal{B}'\} = \text{last}([\rho]_\mathcal{B})$, i.e., $C' = \text{last}(C)$ for some equivalence class $C \in \mathcal{U}/\mathcal{B}$, it follows that for all $a \in \mathcal{A}$ and $C' \in \mathcal{S}/\mathcal{B}$ such that $(p_1, p_2) \in \mathcal{B}$ and $d(\text{last}(p_1), a, C') = \text{rate}_0(p_1, a, C) = \text{rate}_0(p_2, a, C) = \text{rate}_0(p_1, a, C')$.

The behavioral equivalence $\sim_{\text{FBMB}}$ is strictly finer than $\sim_{\text{MB}}$. Indeed, it turns out to be exceedingly discriminating. For example, a CTMC with the only two transitions $s_1 \xrightarrow{(a, \lambda)} s'_1$ and $s_1 \xrightarrow{(a, \mu)} s''_1$ would be distinguished from a CTMC having only transition $s_2 \xrightarrow{(a, \lambda+\mu)} s''_2$.

Observing that in terms of runs the considered transitions are reformulated as $p_1 \xrightarrow{(a, \lambda)} p'_1$, $p_2 \xrightarrow{(a, \lambda+\mu)} p'_2$ where $p_1 = (s_1, \epsilon)$ and $p_2 = (s_2, \epsilon)$, the reflexive, symmetric, and transitive closure of the relation $\{(p_1, p_2), (p'_1, p'_2), (p'_1, p'_2)\}$ would work well when moving forward, as for instance $\text{rate}_0(p_1, a, C) = \lambda + \mu = \text{rate}_0(p_2, a, C)$ for $C = \{p'_1, p'_2\}$, whereas this would not be the case when moving backward, as for instance $\text{rate}_1(p'_1, a, C) = \lambda$, $\text{rate}_1(p'_1, a, C) = \mu$, $\text{rate}_1(p'_2, a, C) = \lambda + \mu$ for $C = \{p_1, p_2\}$.

This example suggests that rate-based quantitative aspects may be neglected when going backward. Indeed, summing up rates when considering transitions departing from the same state is consistent with the fact that the sojourn time in that state is exponentially distributed with rate given by the sum of the rates of the outgoing transitions. Operationally, this can be interpreted as if there were a race among those transitions to decide which one will be executed, with each transition having a winning probability proportional to its rate. This race interpretation no longer applies in the backward direction, as we sum up rates of transitions that may depart from different states. We therefore consider a time-abstract variant of $\sim_{\text{FBMB}}$ when going backward. It is worth noting that, if it abstracted from time also when going forward, then we would precisely obtain the strong back and forth bisimilarity of [10].

Definition 5.6 (forward and time-abstract backward Markovian bisimilarity). An equivalence relation $\mathcal{B}$ over the set of runs $\mathcal{U}$ is a forward and time-abstract backward Markovian bisimulation iff, whenever $(p_1, p_2) \in \mathcal{B}$, then for all actions $a \in \mathcal{A}$ and equivalence classes $C \in \mathcal{U}/\mathcal{B}$:

$$\text{rate}_0(p_1, a, C) = \text{rate}_0(p_2, a, C)$$

$$\text{trans}_1(p_1, a, C) = \text{trans}_1(p_2, a, C)$$

where $\text{trans}_1(p, a, C) = 1$ if there exist $\rho' \in C$ and $\lambda \in \mathcal{R}$ such that $\rho' \xrightarrow{(a, \lambda)} \rho$, $\text{trans}_1(p, a, C) = 0$ otherwise. Two states $s_1, s_2 \in S$ are forward and time-abstract backward Markovian bisimilar, written $s_1 \sim_{\text{FBAB}} s_2$, iff there exists a forward and time-abstract backward Markovian bisimulation $\mathcal{B}$ such that $((s_1, \epsilon), (s_2, \epsilon)) \in \mathcal{B}$. □
We conclude by showing that $\sim_{\text{FTABMB}}$ – which is defined on runs, compares for any action both outgoing total rates and the existence of incoming transitions, and preserves history when going backward even in the presence of concurrent transitions – coincides with the standard $\sim_{\text{MB}}$ – which is defined on states and compares for any action only outgoing total rates – thus generalizing the first result of [10] in our stochastic setting. As a consequence, the former equivalence inherits the compositionality properties and the equational and logical characterizations of the latter, including in particular the expansion law as well as the stochastic variant of idempotency according to which $(a, \lambda). P + (a, \mu). P$ is identified with $(a, \lambda + \mu). P$.

**Theorem 5.7.** Let $s_1, s_2 \in S$. Then $s_1 \sim_{\text{FTABMB}} s_2 \iff s_1 \sim_{\text{MB}} s_2$.

**Proof.** The proof is divided into two parts:

- The proof of $s_1 \sim_{\text{FTABMB}} s_2 \implies s_1 \sim_{\text{MB}} s_2$ is identical to the proof of Theorem 5.5 as only outgoing total rates are considered.

- Suppose that $s_1 \sim_{\text{MB}} s_2$. Let $\text{ct}$ be the mapping that associates with each path $\xi$ its colored trace, i.e., the path obtained from $\xi$ by replacing each state with its Markovian bisimulation equivalence class:

  $$\text{ct}((s_0, (a_1, \lambda_1), s_1)(s_1, (a_2, \lambda_2), s_2) \ldots (s_{n-1}, (a_n, \lambda_n), s_n)) =$$

  $$((s_0)_{\sim_{\text{MB}}}, (a_1, \lambda_1), (s_1)_{\sim_{\text{MB}}}, \ldots (s_{n-1})_{\sim_{\text{MB}}, (a_n, \lambda_n), (s_n)_{\sim_{\text{MB}}}})$$

Let $\mathcal{B} = \{(\rho_1, \rho_2) | \rho_1 \in \text{Run}(s_1), \rho_2 \in \text{Run}(s_2), \text{ct}(\text{path}(\rho_1)) = \text{ct}(\text{path}(\rho_2))\}$, which contains in particular $((s_1, \epsilon), (s_2, \epsilon))$. We show that its reflexive, symmetric, and transitive closure $\mathcal{B}'$ is a forward and time-abstract backward Markovian bisimulation. Given $(\rho_1, \rho_2) \in \mathcal{B}$, i.e., $\rho_1 \in \text{Run}(s_1)$ and $\rho_2 \in \text{Run}(s_2)$ such that $\text{ct}(\text{path}(\rho_1)) = \text{ct}(\text{path}(\rho_2))$, with $l$ being the length of $\text{path}(\rho_1)$ and $\text{path}(\rho_2)$, let us examine the forward and backward directions respectively:

- By virtue of $\text{ct}(\text{path}(\rho_1)) = \text{ct}(\text{path}(\rho_2))$, it holds in particular that $\text{last}(\rho_1) \sim_{\text{MB}} \text{last}(\rho_2)$, hence for all $a \in \mathcal{A}$ and $C \in S/\sim_{\text{MB}}$ we have that $\text{rate}(\text{last}(\rho_1), a, C) = \text{rate}(\text{last}(\rho_2), a, C)$. Since $\text{last}(\rho_k) \xrightarrow{(a, \lambda)} \text{last}(\rho_k')$ iff $\rho_k \xrightarrow{(a, \lambda)} \rho_k'$ for $k \in \{1, 2\}$, with $\rho_k'$ still belonging to $\text{Run}(s_k)$, and any equivalence class $C' \in (\text{Run}(s_1) \cup \text{Run}(s_2))/\mathcal{B}'$ is made of runs with paths of the same length that traverse the same sequence of Markovian bisimulation equivalence classes of states and perform the same sequence of exponentially timed actions, it follows that for all $a \in \mathcal{A}$ and $C' \in (\text{Run}(s_1) \cup \text{Run}(s_2))/\mathcal{B}'$:

  - if the length of all the runs of $C'$ is less than $l + 1$, then it trivially holds that $\text{rate}_0(\rho_1, a, C') = 0 = \text{rate}_0(\rho_2, a, C')$;

  - otherwise, assuming that the states reached after $l + 1$ transitions by all of the runs of $C'$ belong to some $C \in S/\sim_{\text{MB}}$, it holds that $\text{rate}_0(\rho_1, a, C') = \text{rate}(\text{last}(\rho_1), a, C) = \text{rate}(\text{last}(\rho_2), a, C) = \text{rate}_0(\rho_2, a, C')$.

- Since $\text{ct}(\text{path}(\rho_1)) = \text{ct}(\text{path}(\rho_2))$, it cannot be the case that $\text{path}(\rho_1) = \epsilon$ and $\text{path}(\rho_2) \neq \epsilon$, or vice versa. If $\text{path}(\rho_1) = \text{path}(\rho_2) = \epsilon$, then for all $a \in \mathcal{A}$ and $C' \in (\text{Run}(s_1) \cup \text{Run}(s_2))/\mathcal{B}'$ it trivially holds that $\text{trans}_{\epsilon}(\rho_1, a, C') = 0 = \text{trans}_{\epsilon}(\rho_2, a, C')$. Suppose that $\text{path}(\rho_1) \neq \epsilon \neq \text{path}(\rho_2)$, with $\rho_k = \rho'_k \rho''_k$ and $\rho''_k = (s'_k, (a, \lambda), s''_k)$, so that $s'_k \xrightarrow{(a, \lambda)} s''_k$ is the last transition in $\text{path}(\rho_k)$ and hence $\rho'_k \xrightarrow{(a, \lambda)} \rho_k$ with $\rho'_k$ still belonging to $\text{Run}(s_k)$, for $k \in \{1, 2\}$. From $\text{ct}(\text{path}(\rho_1)) = \text{ct}(\text{path}(\rho_2))$, it follows in particular that $s'_1 \sim_{\text{MB}} s'_2$. As a consequence, for all $a \in \mathcal{A}$ and $C' \in (\text{Run}(s_1) \cup \text{Run}(s_2))/\mathcal{B}'$: 


* if the length of all the runs of $C'$ is less than $l$, then it trivially holds that $\text{trans}_1(\rho_1, a, C') = 0 = \text{trans}_1(\rho_2, a, C')$;
* otherwise, assuming that the states reached after $l - 1$ transitions by all of the runs of $C'$ belong to some $C \in S_{\sim MB}$, it holds that $\text{trans}_1(\rho_1, a, C') = 1 = \text{trans}_1(\rho_2, a, C')$ or $\text{trans}_1(\rho_1, a, C') = 0 = \text{trans}_1(\rho_2, a, C')$ depending on whether $s'_1, s'_2 \in C$ or $s'_1, s'_2 \notin C$.

**Corollary 5.8.** Let $R_1, R_2 \in \mathcal{P}$ be two reachable processes. Then $R_1 \sim_{\text{FTAMM}} R_2 \iff R_1 \sim_{\text{MB}} R_2$. □

### 6. Conclusions

Since the work of Landauer [23], reversible computing has attracted a growing attention mainly for the possibility of building energy efficient circuits. Nowadays, the interest has spread into many application areas and hence reversible computing requires a deep investigation of its theoretical foundations from a software science perspective. There exist different interpretations of reversibility in the literature. In this paper, we have addressed our research quest towards bridging causal consistent reversibility [8] – developed in concurrency theory – and time reversibility [20] – originated in the field of stochastic processes for efficient analysis purposes.

We have accomplished this by introducing the stochastic process calculus RMPC, whose syntax and semantics follow the approach of [39], with the aim of paving the way to concurrent system models that are causal consistent reversible and time reversible by construction. Causal consistent reversibility has been proved by exploiting the technique of [30] after importing in the setting of [39] some notions coming from [8]. Thanks to time reversibility, we have inherited from [35] a product form result that enables the efficient calculation of performance measures. Finally, we have developed two forward and backward variants of Markovian bisimilarity [17] inspired by [10], with the former preserving the expansion law and the latter additionally preserving a stochastic variant of the idempotency law.

There are several lines of research that we plan to undergo, ranging from the application of our results to case studies modeled with RMPC to the development of further theoretical results. For instance, we would like to investigate further conditions under which time reversibility is achieved, in addition to the one relying on the equality of forward and backward rates, as well as further properties of forward and backward variants of Markovian bisimilarity. As another example, we would like to add recursion to the syntax of RMPC. From the point of view of the ergodicity of the underlying CTMC, the absence of recursion is not a problem because every forward transition has the corresponding backward transition by construction. However, there might be situations in which recursion is necessary to appropriately describe the behavior of a system. Because of the use of communication keys, a simple process of the form $P \triangleq (a, \lambda).P$, whose standard labeled transition system features a single state with a self-looping transition, produces a sequence of infinitely many distinct states even if we resort to transition bundles. Our claim is that the specific cooperation operator that we have considered may require a mechanism lighter than communication keys to keep track of past actions, which may avoid the generation of an infinite state space in the presence of recursion.
ACKNOWLEDGEMENT

This research has been supported by the MUR PRIN project NiRvAna – Noninterference and Reversibility Analysis in Private Blockchains.

REFERENCES

[1] J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In Proc. of CONCUR 1993, volume 715 of LNCS, pages 477–492. Springer, 1993.
[2] K. Barylska, M. Koutny, L. Mikulski, and M. Piatkowski. Reversible computation vs. reversibility in Petri nets. Science of Computer Programming, 151:48–60, 2018.
[3] C.H. Bennett. Logical reversibility of computations. IBM Journal of Research and Development, 17:525–532, 1973.
[4] C.H. Bennett. Notes on Landauer’s principle, reversible computation, and Maxwell’s demon. Studies in History and Philosophy of Science Part B: Studies in History and Philosophy of Modern Physics, 34:501–510, 2003.
[5] M. Bernardo and C.A. Mezzina. Towards bridging time and causal reversibility. In Proc. of FORTE 2020, volume 12136 of LNCS, pages 22–38. Springer, 2020.
[6] A. Berut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz. Experimental verification of Landauer’s principle linking information and thermodynamics. Nature, 483:187–189, 2012.
[7] I. Cristescu, J. Krivine, and D. Varacca. A compositional semantics for the reversible π-calculus. In Proc. of LICS 2013, pages 388–397. IEEE-CS Press, 2013.
[8] V. Danos and J. Krivine. Reversible communicating systems. In Proc. of CONCUR 2004, volume 3170 of LNCS, pages 292–307. Springer, 2004.
[9] V. Danos and J. Krivine. Transactions in RCCS. In Proc. of CONCUR 2005, volume 3653 of LNCS, pages 398–412. Springer, 2005.
[10] R. De Nicola, U. Montanari, and F.W. Vaandrager. Back and forth bisimulations. In Proc. of CONCUR 1990, volume 458 of LNCS, pages 152–165. Springer, 1990.
[11] E. de Vries, V. Koutavas, and M. Hennessy. Communicating transactions. In Proc. of CONCUR 2010, volume 6269 of LNCS, pages 569–583. Springer, 2010.
[12] M.P. Frank. Physical foundations of Landauer’s principle. In Proc. of RC 2018, volume 11106 of LNCS, pages 3–33. Springer, 2018.
[13] E. Giachino, I. Lanese, and C.A. Mezzina. Causal-consistent reversible debugging. In Proc. of FASE 2014, volume 8411 of LNCS, pages 370–384. Springer, 2014.
[14] P.G. Harrison. Turning back time in Markovian process algebra. Theoretical Computer Science, 290:1947–1986, 2003.
[15] L. Hay-Schmidt, R. Glück, M.H. Cservenka, and T. Haulund. Towards a unified language architecture for reversible object-oriented programming. In Proc. of RC 2021, volume 12805 of LNCS, pages 96–106. Springer, 2021.
[16] J. Hillston. The nature of synchronisation. In Proc. of PAPM 1994, pages 51–70. University of Erlangen, Technical Report 27–4, 1994.
[17] J. Hillston. A Compositional Approach to Performance Modelling. Cambridge University Press, 1996.
[18] C.A.R. Hoare. Communicating Sequential Processes. Prentice Hall, 1985.
[19] J.-R. Jackson. Jobshop-like queueing systems. Management Science, 10:131–142, 1963.
[20] F.P. Kelly. Reversibility and Stochastic Networks. John Wiley & Sons, 1979.
[21] J.G. Kemeny and J.L. Snell. Finite Markov Chains. Van Nostrand, 1960.
[22] L. Kleinrock. Queueing Systems. John Wiley & Sons, 1975.
[23] R. Landauer. Irreversibility and heat generated in the computing process. IBM Journal of Research and Development, 5:183–191, 1961.
[24] I. Lanese, M. Lienhardt, C.A. Mezzina, A. Schmitt, and J.-B. Stefani. Concurrent flexible reversibility. In Proc. of ESOP 2013, volume 7792 of LNCS, pages 370–390. Springer, 2013.
[25] I. Lanese and D. Medić. A general approach to derive uncontrolled reversible semantics. In Proc. of CONCUR 2020, volume 171 of LIPIcs, pages 33:1–33:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
[26] I. Lanese, D. Medić, and C.A. Mezzina. Static versus dynamic reversibility in CCS. *Acta Informatica*, 58:1–34, 2021.
[27] I. Lanese, C.A. Mezzina, and J.-B. Stefani. Reversing higher-order π. In *Proc. of CONCUR 2010*, volume 6269 of LNCS, pages 478–493. Springer, 2010.
[28] I. Lanese, N. Nishida, A. Palacios, and G. Vidal. CauDeR: A causal-consistent reversible debugger for Erlang. In *Proc. of FLOPS 2018*, volume 10818 of LNCS, pages 247–263. Springer, 2018.
[29] I. Lanese, N. Nishida, A. Palacios, and G. Vidal. A theory of reversibility for Erlang. *Journal of Logical and Algebraic Methods in Programming*, 100:71–97, 2018.
[30] I. Lanese, I.C.C. Phillips, and I. Ulidowski. An axiomatic approach to reversible computation. In *Proc. of FOSSACS 2020*, volume 12077, pages 442–461. Springer, 2020.
[31] J.S. Laursen, L.-P. Ellekilde, and U.P. Schultz. Modelling reversible execution of robotic assembly. *Robotica*, 36:625–654, 2018.
[32] G.B. Leeman Jr. A formal approach to undo operations in programming languages. *ACM Transactions on Programming Languages and Systems*, 8:50–87, 1986.
[33] J.-J. Lévy. An algebraic interpretation of the λβK-calculus; and an application of a labelled λ-calculus. *Theoretical Computer Science*, 2:97–114, 1976.
[34] M. Lienhardt, I. Lanese, C.A. Mezzina, and J.-B. Stefani. A reversible abstract machine and its space overhead. In *Proc. of FMOODS/FORTE 2012*, volume 7273 of LNCS, pages 1–17. Springer, 2012.
[35] A. Marin and S. Rossi. Quantitative analysis of concurrent reversible computations. In *Proc. of FORMATS 2015*, volume 9268 of LNCS, pages 206–221. Springer, 2015.
[36] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
[37] A. Palacios and G. Vidal. Concolic execution in functional programming by program instrumentation. In *Proc. of ICFP 2015*, volume 9527 of LNCS, pages 277–292. Springer, 2015.
[38] K.S. Perumalla and A.J. Park. Reverse computation for rollback-based fault tolerance in large parallel systems - Evaluating the potential gains and systems effects. *Cluster Computing*, 17:303–313, 2014.
[39] I.C.C. Phillips and I. Ulidowski. Reversing algebraic process calculi. *Journal of Logic and Algebraic Programming*, 73:70–96, 2007.
[40] I.C.C. Phillips, I. Ulidowski, and S. Yuen. A reversible process calculus and the modelling of the ERK signalling pathway. In *Proc. of RC 2012*, volume 7581 of LNCS, pages 218–232. Springer, 2013.
[41] G.M. Pinna. Reversing steps in membrane systems computations. In *Proc. of CMC 2017*, volume 10725 of LNCS, pages 245–261. Springer, 2017.
[42] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proc. of SIGMETRICS 1985*, pages 147–154. ACM Press, 1985.
[43] M. Schordan, T. Oppelstrup, D.R. Jefferson, and P.D. Barnes Jr. Generation of reversible C++ code for optimistic parallel discrete event simulation. *New Generation Computing*, 36:257–280, 2018.
[44] H. Siljak, K. Psara, and A. Philippou. Distributed antenna selection for massive MIMO using reversing Petri nets. *IEEE Wireless Communication Letters*, 8:1427–1430, 2019.
[45] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.
[46] M. Vassor and J.-B. Stefani. Checkpoint/rollback vs causally-consistent reversibility. In *Proc. of RC 2018*, volume 11106 of LNCS, pages 286–303. Springer, 2018.
[47] T. Yokoyama and R. Glick. A reversible programming language and its invertible self-interpreter. In *Proc. of PÆPEM 2007*, pages 144–153. ACM Press, 2007.