Theoretical Computer Science 926 (2022) 3–20

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Controlled reversibility in communicating reaction systems ☆

Attila Bagossy, György Vaszil *

Department of Computer Science, Faculty of Informatics, University of Debrecen, Kassai út 26, 4028, Debrecen, Hungary

ARTICLE INFO

Article history:
Received 3 July 2021
Received in revised form 27 May 2022
Accepted 30 May 2022
Available online 2 June 2022

Keywords:
Reaction systems
Reversible computing
Networks of reaction systems

ABSTRACT

We study the reversibility of communicating reaction systems, variants of networks of reaction systems communicating by sending reaction products to specific target components. We first consider the possibility of "backtracking" their computations, then define distributed communicating reaction systems, an "unsynchronized" variant of the model in order to show how reversibility can be defined in a causally consistent manner.

1. Introduction

Reaction systems were introduced by Ehrenfeucht and Rozenberg in [1] with the aim of creating a computational model motivated by the biochemical processes occurring inside the cells of living organisms. The idea is to think of these processes as computations and to describe with simple mathematical tools how the interplay between facilitation and inhibition is performed (these mechanisms determine which reactions may take place), that is, how the corresponding computations proceed. A reaction system is essentially a set of reactions, each reaction is represented by a triple of finite sets: the sets of reactants, inhibitors, and results. In each step, the system produces the results of those reactions which are not inhibited and have all their reactants present as the result of the previous computational step. See [2] for more details.

An interesting direction in the study of reaction systems is the theory of networks of reaction systems [3] consisting of several reaction systems working and cooperating according to different strategies for distributing their products. Basically, the sets of products of every reaction system provide the context for the reaction systems at neighboring nodes. A variant of this concept, called communicating reaction systems communicating by products, was introduced by Csuhaj-Varjú and Sethy in [4] where the component systems use so called extended reactions, reactions where each product entity is communicated to specific target components of the system.

In this paper we study reversibility in the framework of communicating reaction systems. Reversible computation is a paradigm aiming to extend the traditional "forward only" notion of computation with the possibility of being executed also in the reverse direction. For more information, see the monographs [5,6] or the recent collection [7]. In deterministic computational models, reversibility essentially means backward determinism and is closely related to information preservation. Motivated by arguments of Landauer in [8], namely that operations with information loss necessarily result in heat dissipation, Bennett constructed a reversible universal Turing machine in [9], showing that irreversibility is not an inherent property of computation.

☆ This work was supported by the National Research, Development and Innovation Fund of Hungary through project no. K 120558, financed under the K 16 funding scheme.

* Corresponding author.
E-mail addresses: attila.bagossy@chree.io (A. Bagossy), vaszil.gyorgy@inf.unideb.hu (G. Vaszil).

https://doi.org/10.1016/j.tcs.2022.05.030

From a more practical point of view, however, the reversing of not necessarily deterministic processes is also of interest, see [10] for example, for the case of reversibility of nondeterministic finite automata. Reversibility in this case implies the possibility of exploration and experimentation by undoing certain parts of computations and redoing them by choosing possibly different computational paths. Following [5], we might call this the do-undo-redo paradigm or "backtracking", as it involves undoing and then redoing parts of computations, where by undoing we mean the execution of the last computational steps in the exact reverse order.

While backward determinism relies on the exact sequential ordering of computational steps, causal-consistent reversibility, introduced by Danos and Krivine [11], relates causality with reversibility. The need for such a paradigm arises from the fact that computations of distributed and concurrent systems usually lack the notion of the "last step", so ideas that rely on the total order of computational steps are not applicable. Nonetheless, some ordering is still imperative to decide which step to undo next. Causal-consistency uses causality to determine the order of reversal. Essentially, it says that we can only reverse a computational step, once we reversed all of its consequences. Thus, to construct models according to this paradigm, the concept of consequence is fundamental. It had been used in many different computational models since its inception, primarily in the realm of process calculi [12], but relevant work has also been done in other areas, such as Petri nets [13], Erlang [14], or event structures [15].

In the subsequent part of the paper we will study the reversibility of communicating reaction systems. First, we look at the possibility of the backtracking of computations, then, we study causal consistent reversibility in communicating reaction systems consisting of reversible components.

1.1. Related work

Since Danos and Krivine introduced the notion of causal-consistent reversibility in their seminal paper [11], it has become the standard notion for defining reversibility in the concurrent setting. While it was originally defined on top of Milner's calculus of communicating systems (CCS) [16], a type of process calculus, since then this notion has seen wide use in quite different models as well, such as Petri Nets [13], the Erlang programming language [14] or term rewriting systems [17].

Our contribution of introducing a controlled form of causal-consistency in the framework of reaction systems uses the original notion of consequence and causal-consistency as defined in [11] and is inspired by the explicit rollback facilities of [18,19]. As an instantiation of causal-consistency in the framework of a specific computational model, our work is also connected to a generalized view of causal-consistency, as written in [20]. However, in this work, we focus on the original notion of consequence and causal-consistency, we do not consider causal safety and causal liveness. The way our construction tracks the communication between the distributed components is also related to the stack-like solutions of reversible process calculi as collected in [12].

The reversibility of reaction systems has been considered by Aman and Ciobanu in multiple works (see [21] and [22]). Our contribution differs by allowing external input from the environment (originally examined in [23] and [24]) and considering causal-consistent reversibility in the setting of communicating reaction systems that contain multiple simultaneously executing components.

2. Preliminaries

First we recall the basic notions concerning reaction systems, refer to [1] for more details. Let S denote a finite set of entities. Reactions are defined over S as triplets $a = (R_a, I_a, P_a)$ with $R_a, I_a, P_a \subseteq S$. The three sets R_a, I_a and P_a contain the *reactants*, *inhibitors* and *products* of the reaction, satisfying the constraints $R_a \cap I_a = \emptyset$, $R_a \neq \emptyset$ and $P_a \neq \emptyset$. (It is usually also assumed that I_a, the set of inhibitors is non-empty, but for the sake of generality we will drop this additional assumption here.) The set of all reactions over S is denoted by $\mathrm{rac}(S)$.

Given a set S and a reaction $a \in \mathrm{rac}(S)$, a is *enabled by* $W \subseteq S$ if $R_a \subseteq W$ and $I_a \cap W = \emptyset$. The *result of* a *on* W, denoted by $\mathrm{res}_a(W)$, is defined as $\mathrm{res}_a(W) = P_a$ if a is enabled by W, or $\mathrm{res}_a(W) = \emptyset$ if a is not enabled by W. If A is a finite set of reactions over S, then $\mathrm{en}_A(W)$ denotes the *set of all reactions in* A *enabled by* W, thus $\mathrm{en}_A(W) = \{a \in A \mid a \text{ is enabled by } W\}$, and the *result of* A *on* W, denoted by $\mathrm{res}_A(W)$, is defined as $\mathrm{res}_A(W) = \bigcup_{a \in A} \mathrm{res}_a(W)$ (or alternatively, as $\mathrm{res}_A(W) = \bigcup_{a \in \mathrm{en}_A(W)} P_a$).

A *reaction system* is a pair $\mathcal{A} = (S, A)$, where the background set S is a finite set of entities while $A \subseteq \mathrm{rac}(S)$ is the set of reactions.

An *interactive process* in a reaction system $\mathcal{A} = (S, A)$ is a pair $\pi = (\gamma, \delta)$ of finite sequences, such that γ is the *context sequence* of π, defined as $\gamma = C_0, C_1, \ldots C_m$, where $C_i \subseteq S$ for all $0 \leq i \leq m$, and δ is the *result sequence* of π, defined as $\delta = D_0, D_1, \ldots D_m$, where $D_0 = \emptyset$ and $D_i = \mathrm{res}_A(D_{i-1} \cup C_{i-1})$ for all $1 \leq i \leq m$. The process is *non-restarting*, if $D_j \neq \emptyset$ for all $1 \leq j \leq m-1$. We also define $\mathrm{sts}(\pi)$ as the *state sequence* of π by $\mathrm{sts}(\pi) = W_0, W_1, \ldots W_m$, where $W_i = C_i \cup D_i$ for $0 \leq i \leq m$.

Some other notions that we need are as follows. A state $W \subseteq S$ is *reachable* in the reaction system $\mathcal{A} = (S, A)$, if there exists an interactive process π with state sequence $\mathrm{sts}(\pi) = W_0, W_1, \ldots W_j$, such that $W = W_j$ for some $j \geq 0$. Note that if $C_0 \subseteq S$ is allowed to be arbitrary, then all possible states (subsets of S) are reachable, but if we impose restrictions on the initial state (as it will be the case later), then the existence of unreachable states may follow.

For $W, W' \subseteq S$, let W' be a *direct predecessor* of W in the reaction system $\mathcal{A} = (S, A)$ if $\mathrm{res}_A(W') = W$, and let the set of direct predecessors of W be denoted by $\mathrm{pred}(W)$. We say that W has a *unique predecessor* if the set of its direct predecessors is a singleton, that is, if $|\mathrm{pred}(W)| = 1$. Otherwise, if $|\mathrm{pred}(W)| > 1$, then W has *multiple predecessors*.

Reversibility of reaction systems was also studied by Aman and Ciobanu in [21,22]. Essentially, their approach is based on constructing reversible reaction systems by adding the reversed version of reactions which are defined by exchanging the products and the reactants. Our notion of reversible reaction systems, on the other hand, is defined by the reversibility of the interactive processes.

An interactive *process* π with $\mathrm{sts}(\pi) = W_0, W_1, \ldots W_m$ is *reversible* if W_j has a unique predecessor for each $j,\ 1 \leq j \leq m$. A *reaction system* $\mathcal{A}$ is *reversible*, if every non-restarting interactive process in $\mathcal{A}$ is reversible.

In [23] we gave sufficient and necessary conditions for the reversibility of a reaction system in this sense. We followed the idea of so called context restricted reaction systems introduced and studied in [25] to reformulate the notion of an interactive process. The background set is "refined" as $S = \Sigma_c \cup \Sigma_p$, the union of (not necessarily disjoint) alphabets of symbols where $\Sigma_c \subseteq S$ contains the context entities, those entities which are allowed to appear as environmental input in the context sets, while $\Sigma_p \subseteq S$ contains those which can appear as products of reactions.

Besides the separation of the alphabet into contexts and products, we also need the following notations. If A is a finite set of reactions, we denote by R_A and P_A the union of the reactant sets and product sets, that is, $R_A = \cup_{a \in A} R_a$ and $P_A = \cup_{a \in A} P_a$.

Also, if S is a finite set such that $A \subseteq \mathrm{rac}(S)$, then $\mathrm{EN}_A(S)$ contains the sets of reactions where the members of each set can be applied together for some subset of S. Formally $\mathrm{EN}_A(S) = \{E \subseteq A \mid \text{there exists } S' \subseteq S, \text{ such that } \mathrm{en}_A(S') = E\}$, and further, we denote by $\mathrm{RES}_A(S)$ the set that contains the results of applying every set of reactions in $\mathrm{EN}_A(S)$ to the appropriate subsets of entities, that is, $\mathrm{RES}_A(S) = \{\mathrm{res}_E(S') \mid S' \subseteq S, E \subseteq A, \text{ such that } \mathrm{en}_A(S') = E\}$ (or alternatively, $\mathrm{RES}_A(S) = \mathrm{res}_{EN_A(S)}(S)$).

Now, we can recall the characterization of reversible systems.

Theorem 1 *([23], Theorem 1). A reaction system $\mathcal{A} = (S, A)$ is reversible, if and only if the following conditions hold.*

(1) For all $E_1, E_2 \in EN_A(S)$, $E_1 \neq E_2$ implies $P_{E_1} \neq P_{E_2}$.
(2) For all $W_1, W_2 \subseteq S$, $\mathrm{en}_A(W_1) \neq \emptyset$, $W_1 \neq W_2$ implies $\mathrm{en}_A(W_1) \neq \mathrm{en}_A(W_2)$.
(3) For all $R_1, R_2 \in \mathrm{RES}_A(S)$ such that $R_1 = \mathrm{res}_A(W)$ for some state $W \subseteq S$ which is reachable in $\mathcal{A}$, $R_1 \neq R_2$ implies $R_1 \setminus \Sigma_c \neq R_2 \setminus \Sigma_c$.

Condition (1) formulates the requirement that we cannot have different sets of reactions producing the same result. Clearly, if we had two different sets of reactions that produce the same entities when applied, then the state formed by the resulting entities would have multiple predecessors. Condition (2) describes the requirement that the sets of reactions enabled by any two distinct subsets of S must be different as well. This is also clear, since if two subsets would enable the same set of reactions, then the result set of these reactions would be a state having at least two predecessors, the two subsets that enable them. Condition (3) excludes the case when two result sets differ only in entities which can appear in the context sets. This is necessary, since $R_1 = \mathrm{res}_{E_1}(W_1)$ and $R_2 = \mathrm{res}_{E_2}(W_2)$ with $R = R_1 \setminus \Sigma_c = R_2 \setminus \Sigma_c$ and $C = (R_1 \cup R_2) \cap \Sigma_c$ implies that $R \cup C$ has multiple predecessors, namely W_1 and W_2.

Now we recall the most important notions and definitions concerning communicating reaction systems, see [4] for more details.

An *extended reaction* over a set of entities S is a triplet $a = (R_a, I_a, P_a)$, where $P_a \subseteq S \times \mathbb{N}^+$ is the non-empty set of products with targets. A *product with target* is a $\rho = (p, t)$ pair, where $p \in S$ is the actual product, while $t \in \mathbb{N}^+$ is the index of the targeted component. The pair (p, t) means that the product p is communicated to the component with index t. The set of all extended reactions over S is denoted by $\mathrm{erac}(S)$.

Given a set of entities, $T \subseteq S$, an extended reaction $a \in \mathrm{erac}(S)$ is *enabled by* T if $R_a \subseteq T$ and $I_a \cap T = \emptyset$. The *result of a on T*, denoted by $\mathrm{res}_a(T)$, is defined as $\mathrm{res}_a(T) = \{p \mid (p, t) \in P_a\}$ if a is enabled by T, or $\mathrm{res}_a(T) = \emptyset$ if a is not enabled by T. For a set of extended reactions $A \subseteq \mathrm{erac}(S)$, the set of enabled reactions, $\mathrm{en}_A(T)$ and the set of results, $\mathrm{res}_A(T)$ are defined as in the previous section.

A *communicating reaction system* (cdcR system, in short) formed by $n \geq 1$ reaction systems (the components) is an $n+1$ tuple $\Delta = (S, \mathcal{A}_1, \ldots, \mathcal{A}_n)$, where S is a finite set of entities, called the background set of Δ, and $\mathcal{A}_i = (S, A_i)$ is the ith component of Δ, a reaction system over the background set S, with a finite set of extended reactions $A_i \subseteq \mathrm{erac}(S)$. For each extended reaction $a \in A_i$, the indices of the targeted components are from the set $\{1, 2, \ldots, n\}$.

Remark 1. Communicating reaction systems are introduced in [4] with the full name *communicating reaction systems with direct communication* (hence the "*cd*" in *cdcR systems*). The variant we consider are called cdcR systems communicating by product, cdcR(p) systems in short. Since we only consider this type of communication, we use the abbreviation of the more general term, cdcR system, to refer to this variant.

Let $\Delta = (S, \mathcal{A}_1, \ldots, \mathcal{A}_n)$ be a cdcR system as above. A *global state* of Δ is an n-tuple $W = (W^1, \ldots W^n)$ where $W^i \subseteq S$, for $1 \leq i \leq n$. A global state $W_2 = (W_2^1, \ldots, W_2^n)$ is a *direct successor* of the global state $W_1 = (W_1^1, \ldots, W_1^n)$, denoted as

$W_2 = \text{succ}(W_1)$ if $W_2^i = \bigcup_{1\le k\le n}\{p \mid (p,i) \in \text{res}_{A_k}(W_1^k)\}$. If $W_2^i = \text{succ}(W_1^i)$, then W_1^i is one of the *direct predecessors* of W_2^i, denoted as $W_1^i \in \text{pred}(W_2^i)$.

An (m-step) *global state sequence* of the cdcR system is $W_0, W_1, \ldots, W_m$, a sequence of global states, such that $W_{j+1} = \text{succ}(W_j)$ for all $0 \le j \le m-1$. This can also be written as $((W^1)_m, \ldots, (W^n)_m)$ where each $(W^i)_m$ is a finite sequence $(W^i)_m = W_0^i, W_1^i, \ldots, W_m^i$, $1 \le i \le n$, and $(W_j^1, \ldots, W_j^n) = W_j$, $0 \le j \le m$, $(W_0^1, \ldots, W_0^n) = W_0$ is called the *initial state*.

We can also describe the functioning of the cdcR system as an (m-step) *interactive communicating process*, that is, as an n-tuple $\Pi = (\pi^1, \ldots, \pi^n)$ where each $\pi^i = ((\gamma^i)_m, (\delta^i)_m)$ is an *interactive process at component* $\mathcal{A}_i$, that is, a pair of finite sequences, such that $(\gamma^i)_m$ is the context sequence of π^i, defined as the sequence $(\gamma^i)_m = C_0^i, C_1^i, \ldots, C_m^i$ of sets of elements communicated to the component $\mathcal{A}_i$ by the other components, that is, $C_j^i = \bigcup_{1\le k\le n, k\ne i}\{p \mid (p,i) \in \text{res}_{A_k}(W_{j-1}^k)\}$ for $j \ge 1$, and $(\delta^i)_m$ is the result sequence of π^i, defined as $(\delta^i)_m = D_0^i, D_1^i, \ldots, D_m^i$, the sequence of sets of elements produced by reactions of component $\mathcal{A}_i$ which are not communicated to other components, that is, $D_j^i = \{p \mid (p,i) \in \text{res}_{A_i}(W_{j-1}^i)\}$ for $j \ge 1$.

We call the sequence $\text{sts}(\pi^i) = (W^i)_m = W_0^i, W_1^i, \ldots, W_m^i$ the *state sequence of* π^i. Note, that $W_j^i = C_j^i \cup D_j^i$ for $j \ge 0$, and we also assume that $W_0^i = C_0^i$ and $D_0^i = \emptyset$.

In Section 5 we will make extensive use of stacks to track the communication between the components of a system. A stack is an ordered sequence of elements accessible from the most recently pushed item. We denote the empty stack as []. Pushing an element e on top of some stack K is written as $e \bullet K$. The topmost element of K can be retrieved using the $top(K)$ function. Thus, $e = top(e \bullet K)$. One can remove the top of K by writing $pop(K)$. This function will return the new, shortened stack, hence $K = pop(e \bullet K)$.

Following the approach of Lanese, Philips and Ulidowski [20], we will build our notion of reversibility on top of labeled transition systems (LTSs). While we introduce the necessary concepts as they appear, we refer the reader to [20] for more details.

3. Backtracking in communicating reaction systems

In this section we study the possibility of backtracking computations. To this aim, based on [23] we define the reversibility of cdcR systems as follows. The idea is to have a system which is "backward deterministic".

Definition 1. Let $\Delta = (S, \mathcal{A}_1, \ldots, \mathcal{A}_n)$ be a cdcR system, and let $W = (W^1, \ldots, W^n)$, $W^i \subseteq S$, $1 \le i \le n$, be a global state of Δ. We say that W has a *unique predecessor* if the set of its direct predecessors is a singleton, that is, if $|\text{pred}(W)| = 1$. Otherwise, if $|\text{pred}(W)| > 1$, then W has *multiple predecessors*.

Let $\Pi = (\pi^1, \ldots, \pi^n)$ be an interactive communicating process in Δ with $\text{sts}(\pi^i) = (W^i)_m = W_0^i, W_1^i, \ldots, W_m^i$, $1 \le i \le n$. The interactive *process* Π *is reversible*, if every global state $W_j = (W_j^1, \ldots, W_j^n)$, $1 \le j \le m$, has a unique predecessor, and an interactive process Π is *non-restarting*, if $W_j^i \ne \emptyset$ for all $1 \le i \le n$, $1 \le j \le m$.

The cdcR *system* Δ *is reversible*, if every non-restarting interactive communicating process in Δ is reversible.

Note that we defined the reversibility of cdcR systems through the reversibility on non-restarting interactive communicating processes. We did this for the sake of unity with the corresponding definition for individual reaction systems: Reversing a restarting process in an individual reaction system would mean that we produce "something from nothing" which we would like to avoid.

Now we are interested in the relationship of "global" and "local" reversibility of cdcR systems, that is, the reversibility of the system and the reversibility of its components. But what do we mean by the reversibility of a component? Our goal is to call a component reversible, if it would be reversible as an individual reaction system. As cdcR systems usually start their functioning with an initial state, it seems natural to look at the individual components as *initialized context restricted* reaction systems, a notion introduced in [25] to describe a case when only entities from a subset of the background set are allowed to appear in the context sets (with the exception of the initial state which can be arbitrary). In our case, "initialization" is allowed only with entities that are not part of the product alphabet. We need this restriction, as shown in [24], in order to avoid the appearance of circles in the computational paths of the systems used as components. We will exploit this property when constructing reversible systems. As a consequence of the lack of circles, a result set may appear at most once in a given forward computation. Thus, we can use result sets to determine when certain communications happened even without a notion of time or step counting.

Definition 2. Let $\Delta = (S, \mathcal{A}_1, \ldots, \mathcal{A}_n)$ be a cdcR system, and let $\mathcal{A}_i = (S, A_i)$ be one of its components, $1 \le i \le n$, with a set of extended reactions $A_i \subseteq \text{erac}(S)$.

Consider the reaction system $\mathcal{A}_i' = (S, A_i')$ where the background set S contains the subsets $\Sigma_c^i, \Sigma_{localp}^i \subseteq S$ with $\Sigma_c^i = \bigcup_{1\le j\le n, j\ne i}\{p \mid (p,i) \in P_a \text{ for some } a \in A_j\}$, $\Sigma_{localp}^i = \{p \mid (p,i) \in P_a \text{ for some } a \in A_i\}$, and the set of (non-extended) reactions is $A_i' \subseteq \text{rac}(S)$ with $A_i' = \{(R, I, \{p \mid (p,i) \in P\}) \mid (R, I, P) \in A_i\}$.

We call the ith component of the cdcR system Δ *reversible*, if the reaction system $\mathcal{A}'_i$ constructed above is reversible, that is, if every non-restarting interactive process π^i in $\mathcal{A}'_i$ with state sequence $\mathrm{sts}(\pi^i) = C^i_0, C^i_1 \cup D^i_1, \ldots, C^i_m \cup D^i_m$ where $C^i_0 \subseteq (S \setminus \Sigma^i_{localp})$, $C^i_j \subseteq \Sigma^i_c$, $1 \leq j \leq m$, is reversible.

A cdcR system is *locally reversible*, if all its components are reversible.

It is clear, that local reversibility (the reversibility of the components) implies the reversibility of the cdcR system, as can be seen by the following proposition.

Proposition 1. *If a cdcR system Δ is locally reversible, then Δ is also reversible.*

Proof. Let $\Delta = (S, \mathcal{A}_1, \ldots, \mathcal{A}_n)$ be a cdcR system, and let $\mathcal{A}'_i = (S, A')$ (constructed as above according to Definition 2) be reversible for all $1 \leq i \leq n$.

Consider a non-restarting interactive communicating process Π and an arbitrary global state W which is reachable through this process, that is, $\Pi = (\pi^1, \ldots, \pi^n)$ with $\mathrm{sts}(\pi^i) = W^i_0, W^i_1, \ldots, W^i_j$, $1 \leq i \leq n$, and $W = (W^1_j, \ldots, W^n_j)$ for some $j \geq 0$.

Since Π is non-restarting, $W^i_k \neq \emptyset$ for all $1 \leq i \leq n$, $0 \leq k \leq j$, so every π^i is also a non-restarting interactive process. Because every non-restarting interactive process in $\mathcal{A}'_i$ is reversible, π^i must also be reversible, thus, all W^i_k, $1 \leq k \leq j$, in the state sequences $\mathrm{sts}(\pi^i)$, $1 \leq i \leq n$, have unique predecessors. This means that the state $(W^1_j, \ldots, W^n_j)$ must also have a unique predecessor, and since $(W^1_j, \ldots, W^n_j)$ was chosen arbitrarily, it also means that Π is reversible. Since any non-restarting interactive communicating process in Δ is reversible, Δ is also reversible. □

On the other hand, a cdcR system can be reversible also in the case when its component systems are not.

Proposition 2. *There are reversible cdcR systems which are not locally reversible.*

Proof. Consider the cdcR system $\Delta = (S, \mathcal{A}_1, \mathcal{A}_2)$ where $S = \{a, b, c, d, e\}$ and $\mathcal{A}_i = (S, A_i)$, $1 \leq i \leq 2$, with sets of extended reactions

$$\begin{aligned} A_1 = \{&a^1_1 = (\{a\}, \{b, c, d, e\}, \{(b, 1)\}),\ a^1_2 = (\{b\}, \{a, c, d, e\}, \{(d, 1)\}),\\ &a^1_3 = (\{b, e\}, \{a, c, d\}, \{(c, 1)\})\},\\ A_2 = \{&a^2_1 = (\{a\}, \{b, c, d, e\}, \{(b, 2), (e, 1)\}),\ a^2_2 = (\{d\}, \{a, b, c, e\}, \{(b, 2)\}),\\ &a^2_3 = (\{b\}, \{a, c, d, e\}, \{(c, 2)\})\}. \end{aligned}$$

All non-restarting interactive communicating processes in Δ are reversible, so Δ is reversible. To see this, we can consider each non-restarting interactive communicating process. In order to have at least two states in the state sequence (to have at least an enabled reaction in both components), the initial global states (W^1_0, W^2_0) can be such, that $W^1_0 \in \{\{a\}, \{b\}, \{b, e\}\}$ and $W^2_0 \in \{\{a\}, \{b\}, \{d\}\}$ which gives us nine combinations with nine possible state sequences. The reader might check that all global states in these sequences have unique predecessors, we look at the case when $W^1_0 = W^2_0 = \{a\}$ as an example. This sequence consists of three states $W_0 = (\{a\}, \{a\})$, $W_1 = (\{b, e\}, \{b\})$, $W_2 = (\{c\}, \{c\})$.

Let us check that each of them has a unique predecessor. The state $(\{c\}, \{c\})$ can only be the result of applying reactions a^1_3 and a^2_3, and the inhibitors only allow these to be applied to the state $(\{b, e\}, \{b\})$, which is W^i_1, the previous state in the sequence. Looking at $(\{b, e\}, \{b\})$, we see that it can only be obtained by the products of a^1_1 and a^2_1, and these can only be applied to $(\{a\}, \{a\})$ which is W_0, the initial state of the sequence.

Let us check now, that at least one of the components of Δ is not reversible by constructing $\mathcal{A}'_2 = (S, A'_2)$, the initialized reaction system corresponding to the component $\mathcal{A}_2$, as described in Definition 2.

We have $\Sigma^2_c = \emptyset$, $\Sigma^2_{localp} = \{b, c, d\}$, $S \setminus (\Sigma^2_c \cup \Sigma^2_{localp}) = \{a, e\}$, and

$$\begin{aligned} A'_2 = \{&a^{2\prime}_1 : (\{a\}, \{b, c, d, e\}, \{b\}),\ a^{2\prime}_2 : (\{d\}, \{a, b, c, e\}, \{b\}),\\ &a^{2\prime}_3 : (\{b\}, \{a, c, d, e\}, \{c\})\}. \end{aligned}$$

Consider now the state sequence of an interactive process π^2 with $\mathrm{sts}(\pi^2) = \{a\}, \{b\}, \{c\}$, and notice that $\{b\}$ has multiple predecessors: it can either be obtained from a state $\{a\}$ as the product of reaction $a^{2\prime}_1$ or from a state $\{d\}$ by the reaction $a^{2\prime}_2$. Since a state in the state sequence has multiple predecessors, the non-restarting interactive process π^2 is not reversible, so $\mathcal{A}'_2$ is not reversible either. □

Note the reason behind the fact that components of a reversible system are not necessarily reversible: The interaction of the components through communication can produce situations when a global state of the system has a unique prede-

cessor even if the "local states" of the individual components have multiple predecessors. (As, for example, the global state $(\{b, e\}, \{b\})$ of the system Δ and the local state $\{b\}$ of the component $\mathcal{A}_2$ in the proof of the proposition above.)

4. Distributed communicating reaction systems

A common trait of both ordinary reaction systems and cdcR systems is the use of a global clock, which drives the computational steps of interactive processes. Every time this clock ticks, each component of the communicating system performs a single computational step, resulting in a new global state. Consequently, as shown in [4], we can construct a single reaction system that computes the same states as its communicating counterpart. In the previous section, building on this property and our previous result regarding the backward determinism of individual systems, we could construct backward deterministic cdcR systems.

Backward determinism is best suited for computational models in which the concept of the last action is well-defined. We can consider simple sequential models, such as ordinary reaction systems, as examples. Once we introduce concurrency, however, the definition of the last action is not that clear anymore. As many computations might have occurred in the system concurrently, imposing a single, exact ordering on backward actions would be overly restricting. This insight led to the development of more suitable paradigms of reversibility, such as causal-consistency, introduced by Danos and Krivine [11].

Having examined backward determinism, we now turn our attention to causal-consistency as a paradigm better fitting distributed or networked models, such as cdcR systems. As noted previously, however, thanks to being synchronized with a global clock, cdcR systems still enjoy the property of the last action, despite their networked nature. Since causal-consistency relates reversibility and concurrency [12], we would like to consider modifications to the cdcR systems model, allowing concurrent computations.

First, we remove the synchronization between component interactive processes by allowing external control to decide which components should compute. Here "compute" refers to applying reactions and consuming input. Thus, individual components may step forward independently from others. Idle components retain their contained symbols while accepting context and communication. While this might seem like a significant diversion from the no permanency rule of reaction systems, no permanency still applies when performing computation. Hence, idle components, in turn, can be thought of as sponges collecting molecules and releasing them once reactions are possible.

Then, we also introduce the notion of blocking, which refers to a component's inability to step forward. If a component cannot apply any reactions, then we say that it is blocked and cannot carry on with its computation. Blocking is related to our previous notion of non-restarting interactive processes, as it prevents a component from computing the empty result set. In this case, we use the concept of blocking to represent the state of being unable to make progress.

We call this modified model distributed cdcR system or d-cdcR system for short. Here, we would like to note that Męski, Koutny and Penczek also defined their model of distributed reaction systems in [25]. However, their solution of handling communication between the components differs significantly from the way cdcR systems work.

In what follows, we formally introduce d-cdcR systems and their computational behavior.

Definition 3. An *interactive process with delays* in a reaction system $\mathcal{A} = (S, A)$ is a pair $\bar{\pi} = (\gamma, \delta)$ of finite sequences, such that γ is the *context sequence* of $\bar{\pi}$, defined as $\gamma = C_0, C_1, \ldots C_m$, where $C_j \subseteq S$ for all $0 \leq j \leq m$, and δ is the *result sequence* of $\bar{\pi}$, defined as $\delta = D_0, D_1, \ldots D_m$, where $D_0 = \emptyset$, and for all $1 \leq j \leq m$,

1. if $\mathrm{res}_A(D_{j-1} \cup C_{j-1}) = \emptyset$, then $D_j = D_{j-1}$, otherwise,
2. if $\mathrm{res}_A(D_{j-1} \cup C_{j-1}) \neq \emptyset$, then

$$D_j = \begin{cases} \mathrm{res}_A(D_{j-1} \cup C_{j-1}), & \text{or} \\ D_{j-1}. \end{cases}$$

We also define $\mathrm{sts}(\bar{\pi})$ as the *state sequence* of $\bar{\pi}$ by $\mathrm{sts}(\bar{\pi}) = W_0, W_1, \ldots W_m$, where $W_j = C_j \cup D_j$ for $0 \leq j \leq m$, and we say that W_{j-1} is an *active state* in $\bar{\pi}$, if $D_j = \mathrm{res}_A(D_{j-1} \cup C_{j-1})$.

Definition 4. Let $\Delta = (S, \mathcal{A}_1, \ldots, \mathcal{A}_n)$ be a cdcR system where $\mathcal{A}_i = (S, A_i)$ is the ith component of Δ, a reaction system over the background set S, with a finite set of extended reactions $A_i \subseteq \mathrm{erac}(S)$.

An m-step *distributed interactive process* is an n-tuple $\bar{\Pi} = (\bar{\pi}^1, \ldots, \bar{\pi}^n)$ where each $\bar{\pi}^i = ((\gamma^i)_m, (\delta^i)_m)$ is an *interactive process with delays at component* $\mathcal{A}_i$ with $(\gamma^i)_m = C_0^i, C_1^i, \ldots, C_m^i$ and $(\delta^i)_m = D_0^i, D_1^i, \ldots, D_m^i$, where the following conditions hold.

In each step, there is a set $Step_j \subseteq \{1, \ldots, n\}$, $0 \leq j \leq m-1$, and

$$Active_j = \{i \mid i \in Step_j \text{ and } i \notin Block_j,\ 1 \leq i \leq n\},$$

where $Block_j = \{i \mid \mathrm{res}_{A_i}(W_j^i) = \emptyset\}$.

For each component $\mathcal{A}_i$, the context sets C^i_j in $(\gamma^i)_m$, $1 \leq j \leq m$, are as follows:

$$ComTo^i_j = \bigcup_{\substack{1\leq k\leq n,\ k\neq i \\ k\in Active_{j-1}}} \{p \mid (p,i) \in \mathrm{res}_{A_k}(W^k_{j-1})\} \cup C^{env,i}_j,$$

$$C^i_j = \begin{cases} ComTo^i_j, \text{ if } i \in Active_{j-1}, \\ ComTo^i_j \cup C^i_{j-1}, \text{ otherwise,} \end{cases}$$

where $C^{env,i}_j$ is the input received by $\mathcal{A}_i$ from the environment in the jth step of the interactive process.

Then, the result sets D^i_j of $\mathcal{A}_i$ in $(\delta^i)_m$, $1 \leq j \leq m$, are as follows:

$$D^i_j = \begin{cases} \{p \mid (p,i) \in \mathrm{res}_{A_i}(W^i_{j-1})\}, \text{ if } i \in Active_{j-1}, \\ D^i_{j-1}, \text{ otherwise.} \end{cases}$$

Let us further detail the above definition. In the case of distributed interactive processes, the environment can control which components will proceed with their computations in each step j using the $Step_j$ sets. Then, the $Active_j$ sets track the effectively computing components, considering whether they are blocked or not. Given a step j, components in the $Active_j$ set consume their context C_j and apply their enabled reactions resulting in a new result set D_{j+1}. Inactive components, on the other hand, accumulate context received in the form of communication or external input (denoted as $ComTo^i_j$ and $C^{env,i}_j$, respectively) and retain their result sets.

In the following, we will consider cdcR systems with distributed interactive processes, and we will call them *distributed communicating reaction systems* (d-cdcR systems in short). Thus, the structure of d-cdcR systems is the same as that of cdcR systems. The difference between the two models lies in how interactive processes perform computations.

Example 1. Consider $\Delta = (S, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, a d-cdcR system with $S = \{a, b, c, d, e, f\}$ and $\mathcal{A}_i = (S, A_i)$, $1 \leq i \leq 3$, with sets of extended reactions

$$\begin{aligned} A_1 = \{\ & a^1_1 = (\{a\}, \{b\}, \{(c,1)\}), \\ & a^1_2 = (\{a,c\}, \{d\}, \{(f,1),(e,2),(e,3)\})\ \}, \\ A_2 = \{\ & a^2_1 = (\{a\}, \{b\}, \{(c,1),(c,2)\}), \\ & a^2_2 = (\{a\}, \{c\}, \{(d,3)\})\ \}, \\ A_3 = \{\ & a^3_1 = (\{c,d\}, \{a\}, \{(b,2),(d,3)\}), \\ & a^3_2 = (\{a\}, \{c\}, \{(a,3)\})\ \}. \end{aligned}$$

The following is a 2-step distributed interactive process in Δ. Let the initial context and result sets

$$((C^1_0, D^1_0), (C^2_0, D^2_0)(C^3_0, D^3_0))$$

be

$$((\{a,b\}, \emptyset), (\{a\}, \emptyset)(\{c,d\}, \emptyset)).$$

Component $\mathcal{A}_1$ is blocked, but we can apply the reactions from A_2 and a^3_1 from A_3, so we might obtain

$$(C^1_1, D^1_1), (C^2_1, D^2_1)(C^3_1, D^3_1)) = ((\{a,b,c\}, \emptyset), (\{a,b\}, \{c\})(\{c,d\}, \{d\})).$$

$\mathcal{A}_1$ is blocked, so $D^1_1 = D^1_0$, and $C^1_1 = C^1_0 \cup \{c\}$ because the product c is received from $\mathcal{A}_2$ due to the application of $a^2_1 \in A_2$. The product set of the second component is $D^2_1 = \{c\}$ (because of the application of $a^2_1 \in A_2$), and $C^2_1 = \{a, b\}$ where b is obtained from $\mathcal{A}_3$ due to the reaction a^3_1, while a is added as input from the environment. The set of products at the third component is $D^3_1 = \{d\}$, because $a^3_1 \in A_3$ was applied, and $C^3_1 = \{c, d\}$, since d is received from the second component and c is added from the environment.

The states now are

$$(W^1_1, W^2_1, W^3_1) = (\{a,b,c\}, \{a,b,c\}, \{c,d\}),$$

so the second component is blocked, and we might choose not to apply anything from A_3, so we can proceed by applying the reaction a^2_1 in the first component (as a^1_1 is inhibited by b, so it cannot be applied) and adding inputs a and c to the first and second components, respectively. This way we get

$$(C_2^1, D_2^1), (C_2^2, D_2^2)(C_2^3, D_2^3)) = (((\{a\}, \{f\}), (\{a, b, c, e\}, \{c\})(\{c, d, e\}, \{d\})),$$

as the symbol f is produced by a_1^2 remains in D_2^1, while e is sent to the second and to the third.

According to the notation of Definition 4, $Step_0 = \{2, 3\}$, $Block_0 = \{1\}$, so $Active_0 = \{2, 3\}$. Further, $Block_1 = \{2\}$ and $Step_1 = Active_1 = \{1\}$.

5. Reversible distributed communicating reaction systems

By enabling concurrent computation to take place in cdcR systems, we can now focus on our approach to reversibility. As discussed previously, causal-consistent reversibility offers a paradigm suitable for concurrent models. On a high level, the definition of causal-consistency says that we can reverse any action with no consequences left. Even without the precise notion of consequence, we can imagine the following scenario: if component $\mathcal{A}$ communicates with component $\mathcal{B}$, which, in turn, proceeds computing, then one cannot reverse the communication from component $\mathcal{A}$ immediately. Before that, one must undo the subsequent steps of $\mathcal{B}$, computed using the symbols received from $\mathcal{A}$. On the other hand, components that do not communicate with each other can be reversed in any order. As we can see, causal-consistent reversibility connects three concepts: reversibility, concurrency, and causality.

Thanks to its desirable features, causal-consistent reversibility has been considered for many computational models. A comprehensive survey on such models and the approaches taken can be found in [12]. These models, however, use their terms to define causal-consistency and related properties, which makes it harder to compare the approaches and establish connections between the different concepts. To resolve these issues and help develop new causal-consistent models, Lanese, Philips, and Ulidowski introduced a common framework based on labeled transition systems [20]. In this framework, they provide a set of axioms and then derive more complex properties from them. The idea is that, given a computational model, it is sufficient to verify these individual axioms rather than to provide elaborate proofs for properties, such as causal-consistency, directly.

The axioms and properties in the framework of [20] are defined for labeled transition systems (LTSs). Thus, a straightforward approach to use them is to construct LTSs corresponding to individual distributed cdcR systems. One can define a labeled transition system by specifying its processes ($Proc$), labels (Lab), and transitions (Tr).

Definition 5. A labeled transition system (LTS) is a triplet $(Proc, Lab, Tr)$, where $Proc$ is the set of processes, Lab is the set of labels, and $Tr \subseteq Proc \times Lab \times Proc$ is a transition relation. A transition t from P to Q, labeled with a, is denoted as $t : P \xrightarrow{a} Q$.

In our first attempt, let us start with the processes of the LTS. Since computation in d-cdcR systems takes place in the form of distributed interactive processes, it is straightforward to turn the global states into LTS processes, as follows. Given an n-component d-cdcR system, processes are of the form

$$P = (W^1, \ldots, W^n),$$

where W^i is the state of component $\mathcal{A}^i$. For the labels of the transitions, we may use the $Step$ sets and the communications between the components.

While the above approach seems sufficient, it is required to satisfy the following properties.

- Every transition is reversible. If there is a forward transition from process P to process Q, there is an inverse transition from Q to P.
- There is an irreflexive, symmetric, binary independence relation defined on transitions. One can relate the independence of transitions with the concurrent behavior of the underlying model.

Constructing an appropriate independence relation is possible, however, reverting transitions is much trickier. First, given any result set D^i, we must be able to determine the predecessor state W^i for which $D = \mathrm{res}(W^i)$ holds. Second, we must ensure that backward transitions respect causality, which, in our model, is related to communication. Unfortunately, we have to extend our current definitions to resolve these issues.

In the case of predecessors, we can choose from two approaches. We can assign a computational history to each component storing its previous steps, or we can require that each state has a uniquely determined predecessor. As stated in our past works, we believe that the latter approach is better suited for reaction systems. Therefore, we will use uniquely determined predecessors in this case as well. To define the reversibility of interactive processes with delays, we need to relax the requirement that all states have unique predecessors. It will be sufficient to consider the steps of the process where actual computations (reaction applications) happen. Regarding the second issue, we must record the flow of communication between the components of the system. Thus, in what follows, we modify our definitions accordingly.

Definition 6. An interactive *process with delays* $\bar{\pi}$ with $\mathrm{sts}(\bar{\pi}) = W_0, W_1, \ldots W_m$ is *reversible*, if for each j, $0 \leq j \leq m-1$, whenever W_j^i is an active state of $\bar{\pi}$, then it is the unique predecessor of W_{j+1}.

A *reaction system with delayed interactive processes* $\mathcal{A}$ is *reversible*, if every interactive process with delays in $\mathcal{A}$ is reversible.

The notion of local reversibility can be based on a similar construction as given in Definition 2, if we also add to Σ_c^i, $1 \leq i \leq n$, the entities that we would like to be able to input to the context sets of component $\mathcal{A}_i$ from the environment.

Definition 7. A *reversible distributed communicating reaction system* formed by $n \geq 1$ reaction systems (called its components) is an $n+1$ tuple $\Delta = (S, \mathcal{A}_1, \ldots, \mathcal{A}_n)$, where S is a finite set of entities, called the background set of Δ, and $\mathcal{A}_i = (S, A_i)$ is the ith component of Δ, a reversible reaction system with delayed interactive processes over the background set S and the set of extended reactions $A_i \subseteq \mathrm{erac}(S)$.

By requiring that each component of a reversible d-cdcR systems is locally reversible (as in the case of the backtracking systems of Section 3), we ensure a uniquely determined predecessor for each result set that might occur. What remains is the record of communication flow between the components.

Definition 8. An m-step *reversible distributed interactive process* is an n-tuple $\bar{\Pi} = (\bar{\pi}^1, \ldots, \bar{\pi}^n)$ where each $\bar{\pi}^i = ((\gamma^i)_m, (\delta^i)_m, (send^i)_m, (recv^i)_m)$ is an *interactive process at component* $\mathcal{A}_i$ with $(\gamma^i)_m$ and (δ^i) defined identically as in Definition 4.

The symbols sent from component $\mathcal{A}_i$ to $\mathcal{A}_k$ in a computational step are defined for $0 \leq j \leq m-1$, as

$$Sent_j^{i,k} = \begin{cases} \{p \mid (p,k) \in \mathrm{res}_{A_i}(W_j^i)\}, & \text{if } i \in Active_j, \\ \emptyset, & \text{otherwise,} \end{cases}$$

and the sets $Active_j$, $Step_j$ and $Block_j$ are defined the same way as in Definition 4.

For each component $\mathcal{A}_i$, $(send^i)_m$ is a series of stack-families in which each stack is defined as follows:

$$send_j^{i,k} = \begin{cases} D_j^i \bullet send_{j-1}^{i,k}, & \text{if } Sent_{j-1}^{i,k} \neq \emptyset, \\ send_{j-1}^{i,k}, & \text{otherwise,} \end{cases}$$

where $1 \leq k \leq n, k \neq i$ is the index of the recipient and $1 \leq j \leq m$ is the index of a computational step.

Then, the stacks in the series $(recv^i)_m$ are defined as follows:

$$recv_j^{k,i} = \begin{cases} (D_j^i, Sent_{j-1}^{k,i}) \bullet recv_{j-1}^{k,i}, & \text{if } Sent_{j-1}^{k,i} \neq \emptyset, \\ recv_{j-1}^{k,i}, & \text{otherwise,} \end{cases}$$

$$recv_j^{env,i} = \begin{cases} (D_j^i, C_j^{env,i}) \bullet recv_{j-1}^{env,i}, & \text{if } C_j^{env,i} \neq \emptyset, \\ recv_{j-1}^{env,i}, & \text{otherwise,} \end{cases}$$

where $1 \leq k \leq n, k \neq i$ is the index of the sender and $1 \leq j \leq m$ is the index of a computational step.

In a reversible distributed interactive process, we use the *send* and *recv* stacks to keep track of the communication between the components comprising the system. Given a step j, if the component $\mathcal{A}_i$ applies reactions that result in products targeting some other component $\mathcal{A}_k$, then, in the next step $j+1$, the new result set of $\mathcal{A}_i$, D_{j+1}^i is placed onto its *sent-to-k* stack, $send_{j+1}^{i,k}$. Since our definition of reversibility and, in particular, unique predecessors forbids computational cycles ([24]), the result set placed onto a *send* stack precisely determines when the component sent the symbols. To be exact, it determines the step in which the target received but not yet used the communicated symbols. Therefore, using the *send* stacks, we can keep track of the targets and times of outgoing communications. The *recv* stacks work in a similar manner with an important exception: they also record the received symbols. Hence, each entry in $recv_j^{k,i}$ contains the result set D_j^i in which component $\mathcal{A}_i$ received the communication, and the actual symbols sent by the origin, $\mathcal{A}_k$.

Example 2. Consider a variant of the system in Example 1, a d-cdcR system $\Delta' = (S, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ with reversible components, where $S = \{a, b, c, d, e, f\}$ and $\mathcal{A}_i = (S, A_i)$, $1 \leq i \leq 3$, with sets of extended reactions

$$\begin{aligned} A_1 = \{\, & a_1^1 = (\{a\}, \{b, c, d, e, f\}, \{(c, 1)\}), \\ & a_2^1 = (\{a, c\}, \{d, e, f\}, \{(f, 1), (e, 2), (e, 3)\}) \,\}, \\ A_2 = \{\, & a_1^2 = (\{a\}, \{b, d, e, f\}, \{(c, 1), (c, 2)\}), \\ & a_2^2 = (\{a\}, \{c, d, e, f\}, \{(d, 3), (f, 2)\}) \,\}, \\ A_3 = \{\, & a_1^3 = (\{c, d\}, \{a, b, e\}, \{(b, 2), (d, 3), (f, 3)\}), \\ & a_2^3 = (\{a\}, \{b, c, d, e, f\}, \{(a, 3), (b, 3)\}) \,\}. \end{aligned}$$

The following is a 2-step reversible distributed interactive process in Δ'. Let the initial context sets, result sets, and stacks

$$((C_0^1, D_0^1, send_0^1, recv_0^1), (C_0^2, D_0^2, send_0^2, recv_0^2)(C_0^3, D_0^3, send_0^3, recv_0^3))$$

be $((\{a, b\}, \emptyset, send_0^1, recv_0^1), (\{a\}, \emptyset, send_0^2, recv_0^2)(\{c, d\}, \emptyset, send_0^3, recv_0^3))$ where the stack sequences are

$$send_0^1 = (send_0^{1,2}, send_0^{1,3}), \quad send_0^2 = (send_0^{2,1}, send_0^{2,3}), \quad send_0^3 = (send_0^{3,1}, send_0^{3,2}),$$

and

$$recv_0^1 = (recv_0^{2,1}, recv_0^{3,1}, recv_0^{env,1}), \quad recv_0^2 = (recv_0^{1,2}, recv_0^{3,2}, recv_0^{env,2}),$$
$$recv_0^3 = (recv_0^{1,3}, recv_0^{2,3}, recv_0^{env,3}),$$

each stack being empty initially.

Component $\mathcal{A}_1$ is blocked, but we can apply both reactions from A_2 and a_1^3 from A_3, so we might obtain

$$((\{a, b, c\}, \emptyset, send_1^1, recv_1^1), (\{a, b\}, \{c, f\}, send_1^2, recv_1^2), (\{c, d\}, \{d, f\}, send_1^3, recv_1^3)),$$

where both stacks in $send_1^1$ are still empty (since the first component is blocked), but $recv_1^{2,1} = (\emptyset, \{c\})$. As the second component sent products to both of the others, we have $send_1^{2,1} = \{c\} = send_1^{2,3} = \{c, f\}$, but it also received the product of the third component together with input from the environment, thus $recv_1^{3,2} = (\{c, f\}, \{b\})$, $recv_1^{env,2} = (\{c, f\}, \{a\})$. Similarly, $send_1^{3,2} = \{d, f\}$, while $send_1^{3,1} = send_0^{3,1} = \emptyset$, and $recv_1^{2,3} = (\{d, f\}, \{d\})$, $recv_1^{env,3} = (\{d, f\}, \{c\})$.

The states now are

$$(W_1^1, W_1^2, W_1^3) = (\{a, b, c\}, \{a, b, c, f\}, \{c, d, f\}),$$

so the second component is blocked. Let us choose not to apply any reaction from A_3, and proceed by applying the reaction a_1^2 and adding environmental inputs a and d to the first and third components, respectively. This way we get

$$((\{a\}, \{f\}, send_2^1, recv_2^1), (\{a, b, e\}, \{c, f\}, send_2^2, recv_2^2), (\{c, d, e\}, \{d, f\}, send_2^3, recv_2^3)).$$

The contents of the stacks are as follows. At the first component, $send_2^{1,2} = send_2^{1,3} = \{f\}$, $recv_2^{env,1} = (\{f\}, \{a\})$. The second and third components were not active, so their *send*-stacks remain unchanged, while $recv_2^{1,2} = (\{c, f\}, \{e\})$, $recv_2^{1,3} = (\{d, f\}, \{e\})$, and $recv_2^{env,3} = (\{d, f\}, \{d\}) \bullet (\{d, f\}, \{c\})$.

Having established the necessary machinery to reverse individual states and track communications, we can now continue by defining the labeled transition system corresponding to our model.

Definition 9. Let Δ be a reversible d-cdcR system, formed by $n \geq 1$ components. Then, a process $P \in Proc$ in the corresponding labeled transition system (LTS) is as follows:

$$P = ((C^1, D^1, send^1, recv^1), \ldots, (C^n, D^n, send^n, recv^n)),$$

where, for each $1 \leq i \leq n$, $W^i = C^i \cup D^i$ is the state of component $\mathcal{A}_i$ and $send^i$ and $recv^i$ denote the $send^{i,k}$, $recv^{k,i}$, $recv^{env,i}$ stacks respectively, $1 \leq k \leq n$, $k \neq i$.

Using the above definition, we can readily represent the steps of any interactive process as LTS processes. Given an n-component reversible d-cdcR system Δ and some m-step reversible distributed interactive process $\bar{\Pi} = (\bar{\pi}^1, \ldots, \bar{\pi}^n)$, we have that

$$P_j = ((W_j^1, send_j^1, recv_j^1), \ldots, (W_j^n, send_j^n, recv_j^n)),$$

such that $P_j \in Proc$, for any step $0 \leq j \leq m$.

Notice that in P_j we have denoted the context and product sets, C_j^i and D_i^i by the state sets W_j^i (which is their union), but this is just a notational convenience, W_j^i stands for the pair (C_j^i, D_i^i), $1 \leq i \leq n$, $0 \leq j \leq m$.

What is left is to connect these processes via labels and transitions.

Definition 10. Let $\Delta = (S, \mathcal{A}^1, \ldots, \mathcal{A}^n)$ be a reversible d-cdcR system, formed by $n \geq 1$ components. Then, a label $a \in Lab$ in the corresponding LTS is as follows:

$$a = active; comm,$$

where $active \subseteq \{1, \ldots, n\}$ contains the indices of active components and $comm \subseteq \{1, \ldots, n, env\} \times S \times \{1, \ldots, n\}$ is the set of communicated symbols in the form of $(sender, symbol, recipient)$ triplets, such that $sender \neq recipient$.

Definition 11. Let $P = ((W_P^1, send_P^1, recv_P^1), \ldots, (W_P^n, send_P^n, recv_P^n)) \in Proc$ be a process in the LTS corresponding to an n-component reversible d-cdcR system. The label $a = active; comm \in Lab$ is valid in state P, if the following hold.

(1) $active \neq \emptyset$,
(2) there is no i, such that $i \in active$ and $i \in Block_P$,
(3) $(i, p, k) \in comm$ if and only if $i \in active$ and $(p, k) \in \mathrm{res}_{A_i}(W_P^i)$, or if $i = env$,

where $Block_P$ is as defined in Definition 4.

Then, the result of the transition,

$$Q = ((W_Q^1, send_Q^1, recv_Q^1), \ldots, (W_Q^n, send_Q^n, recv_Q^n)) \in Proc$$

is as follows. If $i \notin active$, then $W_Q^i = C_Q^i \cup D_Q^i$ for $D_Q^i = D_P^i$ and

$$C_Q^i = C_P^i \cup \{p \mid (k, p, i) \in comm, 1 \leq k \leq n\},$$

$$recv_Q^{k,i} = \begin{cases} (D_Q^i, \{p \mid (k, p, i) \in comm\}) \bullet recv_P^{k,i}, & \text{if } \{p \mid (k, p, i) \in comm\} \neq \emptyset, \\ recv_P^{k,i}, & \text{otherwise.} \end{cases}$$

For all $i \in active$, we have $W_Q^i = C_Q^i \cup D_Q^i$ for $D_Q^i = \mathrm{res}_{A_i}(W_P^i)$ and

$$C_Q^i = \cup\{p \mid (k, p, i) \in comm, 1 \leq k \leq n\},$$

$$send_Q^{i,k} = \begin{cases} D_Q^i \bullet send_P^{i,k}, & \text{if there is some } (i, p, k) \in comm, \\ send_P^{i,k}, & \text{otherwise,} \end{cases}$$

and $recv_Q^{i,k}$ is the same as in the case of $i \notin active$.

The transition is denoted as $P \xrightarrow{a} Q$.

Before going forward, let us further detail the above definition. Condition (1) formulates the requirement that there must be at least one active component computing a new result set, which corresponds to the idea that the underlying system must make progress. Condition (2) restricts the elements of the *active* set, such that blocked components cannot appear. Thus, the *active* set of labels directly corresponds to the *Active* set of Definition 8. Condition (3) establishes a connection between the elements of the *comm* set and the reactions applied in the process P. It states that *comm* contains precisely the same communicated symbols computed by the reactions, optionally with additional input from the environment. The result of the transition, Q, is constructed parallel to the subsequent interactive process steps of Definition 8. Therefore, it is clear that given a reversible distributed interactive process, we can formulate each of the individual steps as LTS processes, and then connect them with transitions of the above form.

Example 3. Consider the system $\Delta' = (S, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ from Example 2, and the state

$$P = ((\{a, b, c\}, \emptyset, send_1^1, recv_1^1), (\{a, b\}, \{c, f\}, send_1^2, recv_1^2), (\{c, d\}, \{d, f\}, send_1^3, recv_1^3)),$$

after the first step of computation in the example. If

$$a = \{1\}; \{(1, e, 2), (1, e, 3), (env, a, 1), (env, d, 3)\},$$

and

$$Q = ((\{a\}, \{f\}, send_2^1, recv_2^1), (\{a, b, e\}, \{c, f\}, send_2^2, recv_2^2), (\{c, d, e\}, \{d, f\}, send_2^3, recv_2^3)),$$

that is, if Q is the next state of the computation from Example 2, then $P \xrightarrow{a} Q$ according to Definition 11 above.

Central to the notion of causal-consistency is the connection between reversibility, causality, and concurrency. In what follows, we establish a link between causality and concurrency in the form of the so-called independence relation. Our approach is similar to that of [26].

Definition 12. Let $t : P \xrightarrow{a} Q$ and $u : Q \xrightarrow{b} S$ be two consecutive forward transitions in an LTS corresponding to some reversible d-cdcR system, where $a = active_a; comm_a$ and $b = active_b; comm_b$. We say that t *causes* u if $active_a \cap active_b \neq \emptyset$ or there exists $k \in active_b$ such that $(i, p, k) \in comm_a$ for some i, k.

Definition 13. Let $t : P \xrightarrow{a} Q$ and $u : P \xrightarrow{b} S$ be two coinitial forward transitions in an LTS corresponding to some reversible d-cdcR system, where $a = active_a; comm_a$ and $b = active_b; comm_b$. We say that t *is in conflict with* u if $active_a \cap active_b \neq \emptyset$ or $\{i \mid (i, p, k) \in comm_a\} \cap active_b \neq \emptyset$ or $\{i \mid (i, p, k) \in comm_b\} \cap active_a \neq \emptyset$.

Remark 2. If t is in conflict with u then u is in conflict with t.

Definition 14. Let t and u be two consecutive transitions. If t does not cause u then they are *independent*. Furthermore, let t' and u' be two coinitial transitions. If t' is not in conflict with u' then they are *independent*.

We write $t \iota u$ to denote that transitions t and u are independent.

Remark 3. Independence is irreflexive, as t is always in conflict with itself. It is also symmetric, given t, u $(t \neq u)$ both $t \iota u$ and $u \iota t$ hold.

In reversible d-cdcR systems, components can influence the computational steps of each other by communicating targeted products. These symbols will appear in the context set of the recipient, potentially impacting the set of enabled reactions in the preceding step. Consequently, we can say that the received communication acted as a cause for the subsequent computation. Of course, we should also consider the straightforward case when a component "communicates" with itself, computing a new result set from its previous state. The above definition of independence translates this idea to LTS transitions. When considering two consecutive transitions, if the first contains computation (either in the form of communication or by producing a new result set) that the second one uses, they are in a causal relationship. When such causality is not present, they are independent. The idea is also translated to coinitial transitions, stating that they cannot be independent if they communicate with and act on the same components.

With the introduction of reversible d-cdcR systems, our goal was to enable uncontrolled reversibility. Thus, to allow computation in interactive processes to flow backward as easily as forward. To this end, the *send-recv* stacks of Definition 8 record communication history, while the unique predecessor property ensures that given any result set, we can determine its predecessor state. Building on top of these, we now define backward labels and transitions for the corresponding LTSs.

Definition 15. Let Δ be a reversible d-cdcR system, formed by $n \geq 1$ components. Then, a *backward label* $\underline{a} \in \underline{Lab}$ in the corresponding LTS is as follows:

$$\underline{a} \subseteq \{1, \ldots, n\} \cup \{env_1, \ldots, env_n\},$$

where $i \in \underline{a}$ corresponds to the reversal of computation in component $\mathcal{A}_i$, and $env_i \in \underline{a}$ represents the removal of the most recently received environmental input from component $\mathcal{A}_i$.

Definition 16. Let $Q = ((W_Q^1, send_Q^1, recv_Q^1), \ldots, (W_Q^n, send_Q^n, recv_Q^n)) \in Proc$ be a process in the LTS corresponding to an n-component reversible d-cdcR system. The backward label $\underline{a} \in \underline{Lab}$ is valid in state Q, if $\underline{a} \cap \{1, \ldots, n\} \neq \emptyset$ and the following hold for each $i, env_i \in \underline{a}$.

(1) $D_Q^i \neq \emptyset$.
(2) If there exists $1 \leq k \leq n$ such that $top(recv_Q^{k,i}) = (D_Q^i, C)$ for some C, then $k \in \underline{a}$ and $top(send_Q^{k,i}) = D_Q^k$.
(3) If there exists $1 \leq k \leq n$ such that $top(send_Q^{i,k}) = D_Q^i$, then $top(recv_Q^{i,k}) = (D_Q^k, C)$ for some C.
(4) If $env_i \in \underline{a}$ then $top(recv_Q^{env,i}) = (D_Q^i, C)$ for some C.
(5) For all $k \in \{1, \ldots, n, env\}$ if $(D, C) = top(pop(recv_Q^{k,i}))$ then $D \neq D_Q^i$.

Then, the result of the transition,

$$P = ((W_P^1, send_P^1, recv_P^1), \ldots, (W_P^n, send_P^n, recv_P^n)) \in Proc$$

is as follows. If $i \notin \underline{a}$:

$$recv_P^{k,i} = \begin{cases} pop(recv_Q^{k,i}) \text{ if } k \in \underline{a}, \\ recv_Q^{k,i}, \text{ otherwise,} \end{cases}$$

$$recv_P^{env,i} = \begin{cases} pop(recv_Q^{env,i}) \text{ if } env_i \in \underline{a} \text{ or } i \in \underline{a}, \\ recv_Q^{env,i}, \text{ otherwise,} \end{cases}$$

and the reversed state is computed as

$$\text{EffectiveContext}^i_P = \bigcup_{k\in\{1,\dots,n,env\}} \text{TopConsecutiveContext}(recv^{k,i}_P),$$

$$D^i_P = D^i_Q \text{ and } C^i_P = \text{EffectiveContext}^i_P.$$

Here, given some $recv$ stack, TopConsecutiveContext($recv$) takes every consecutive entry on top of $recv$ sharing the same result set and computes the union of the accompanying context sets.

Otherwise, if $i \in \underline{a}$ then we have that

$$send^{i,k}_P = \begin{cases} pop(send^{i,k}_Q) \text{ if } top(send^{i,k}_Q) = D^i_Q, \\ send^{i,k}_Q, \text{ otherwise,} \end{cases}$$

$$W^i_P = \text{UniquePredecessor}_{A_i}(D^i_Q),$$

and $recv^{k,i}_P$ is the same as in the case of $i \notin \underline{a}$. Here, $\text{UniquePredecessor}_{A_i}(D)$ refers to the set T such that $\text{res}_{A_i}(T) = D$. Note that $C^i_P = \text{EffectiveContext}^i_P$ also in this case, and because the components are reversible themselves, W^i_P also determines the product set D^i_P

The transition is denoted as $Q \xrightarrow{\underline{a}} P$.

In the above definition, condition (1) restricts the processes from which we can perform backward transitions. Reversible distributed interactive processes, by definition, start with an empty result set. Therefore, by requiring a non-empty result set for every reversed component, condition (1) forbids the reversal of initial states. Then, condition (2) states that we cannot undo a state in which the component received communication. Again, by our definition of reversibility, there are no cycles in the computation, thus, by performing forward transitions, we can only compute any result set at most once. Then, we can use the result sets pushed on the appropriate $recv$ stacks to detect whenever attempting to reverse a step with incoming symbols. We can only undo such steps if the origin also reverses the state in which it sent the received communication. Condition (3) is the sending pair of the previous requirement: a step with outgoing communication may only be reversed if it has no causes. Condition (4) states that we may only undo environmental input in the same state it was received. While blocked or inactive, components may receive communication from the same peer (or the environment) in multiple subsequent computational steps. Thus, the same result set is repeatedly pushed onto the appropriate receive stack. Reversing multiple received communications from the same peer in a single step is not possible because the reversal of communication is synchronized (as required by condition (2)). Condition (5) formulates the requirement that a reversed component had not received multiple communications from the same sender in its current state. The condition ensures that the current result set occurs at most once on the top of the stacks.

Once the above conditions are satisfied, the actual backward transition is relatively straightforward. When undoing a communication, we pop the corresponding entries from the top of participating components' *send-recv* stacks. Then, for each component we wish to reverse, we compute a new state by taking the uniquely determined predecessor of the current state. The predecessor includes both the prior result and context sets. Otherwise, in the case of components not in the transition label, we merely exclude any reversed communication.

Definition 17. With the guidance of our end goal, the Square Property (Definition 3.1, [20]), we can extend the notion of *conflicting* coinitial transitions to include backward ones as well.

(1) Two coinitial backward transitions are never in conflict.

(2) Let $t : P \xrightarrow{a} Q$ and $\underline{u} : P \xrightarrow{\underline{b}} S$ be two coinitial transitions in an LTS corresponding to some reversible d-cdcR system, where $a = active_a; comm_a$. We say that t is in conflict with $\underline{u}$ if any of the following holds true:
   (a) There is $(i, p, k) \in comm_a$ such that $k \in \underline{b}$ but $i \notin \underline{b}$.
   (b) There is $i \in \underline{b}$ and $1 \le k \le n$ such that $top(send^{i,k}_P) = D^i_P$, $top(recv^{i,k}_P) = (D^k_P, C)$, $k \notin \underline{b}$, and $k \in active_a$.
   (c) There is $(env, p, k) \in comm_a$ such that $k \in \underline{b}$.
   (d) There is $1 \le i \le n$ such that $env_i \in \underline{b}$ and $i \in active_a$.

Let us elaborate on the above cases further. Item (1), the case of coinitial backward transitions is a consequence of the validity criteria of Definition 16 and our notion of causality. If multiple backward labels are valid in the same process, then the components affected by the labels do not contain communications in causal or conflicting relationships. Then, item (2) enumerates the cases when transitions of opposite directions conflict. When determining such cases, we used the criteria of the Square Property to assess whether placing the labels on subsequent transitions would yield the same result. The first two subcases are related to the entities communicated between components. In subcase (2)a, if we first compute using the forward label a, the backward label becomes invalid. The same occurs in subcase (2)b because of the mismatch between the communication stacks. Then, the latter two subcases are concerned with the environmental input. As captured by subcase (2)c, reversing and receiving entities from the environment yields a different result than receiving the same

entities followed by the reversal. Subcase (2)d presents an invalid backward label: the forward computation, followed by the removal of environmental input, would render the backward label invalid.

At this point, the transition systems we constructed have transitions both in the forward and backward directions. What is lacking, however, is a connection between these two. The question naturally arises whether it is possible to reverse every transition and if it is, then how. First, we define the inverse of a forward transition.

Definition 18. Let Δ be a reversible d-cdcR system formed by $n \geq 1$ components and $t : P \xrightarrow{a} Q$ be a forward transition in the corresponding LTS with label $a = active; comm$.

The *inverse transition label* of a is denoted by $\underline{a}$ and defined as

$$\underline{a} = active \cup \{env_i \mid \text{there is some } (env, p, i) \in comm\}.$$

The above construction is quite straightforward. If a component performed a computational step in a forward transition (thus, it was part of the *active* set), it is included in the corresponding inverse label. What is left is to remove any received environmental input, which is done by examining the communicated entities of the forward transition. For this end, if component $\mathcal{A}_i$ receives symbols from the environment, a corresponding env_i is placed in the label.

Lemma 1. *Let $t : P \xrightarrow{a} Q$ be a forward transition in an LTS corresponding to some reversible d-cdcR system. Then, there exists a backward transition $\underline{t} : Q \xrightarrow{\underline{a}} P$.*

Proof. Let Δ be a reversible d-cdcR system formed by $n \geq 1$ components and $t : P \xrightarrow{a} Q$ be a forward transition in the corresponding LTS with label $a = active; comm$. Then, let us construct the inverse transition, $\underline{t} : Q \xrightarrow{\underline{a}} S$ according to Definition 18. In what follows, we prove that $\underline{t}$ is indeed the inverse of t, thus $S = P$.

First, let us examine whether $\underline{a}$ is a valid backward label for Q by considering the conditions of Definition 16.

(1) For each $i \in active$, we have that $W^i_Q = \mathrm{res}(W^i_P) \cup \{p \mid (k, p, i) \in comm, 1 \leq k \leq n\}$. Since $\underline{a} \cap \{1, \ldots, n\} = active$, this means that $\mathrm{res}(W^i_P)$ is non-empty for each $i \in \underline{a}$. Thus, D^i_Q must be non-empty as well.

(2) If a component $\mathcal{A}_i$ received communication from some other component $\mathcal{A}_k$ in transition t, then $top(recv^{k,i}_Q) = (D^i_Q, C)$ for some C. This is only possible if $k \in active$. Now, since $\underline{a} \cap \{1, \ldots, n\} = active$ we know that $k \in \underline{a}$ as well. Furthermore, because k was the sender of the received communication, $top(send^{k,i}_Q) = D^k_Q$ holds.

(3) For each $i \in active$ that sent communication to some other component k in transition t, we have that $top(send^{i,k}_Q) = D^i_Q$. However, because of the definition of forward transitions, on the receiving end, $top(recv^{i,k}_Q) = (D^k_Q, C)$ holds (where C denotes the communicated symbols).

(4) $env_i \in \underline{a}$ if there is some symbol p such that $(env, p, i) \in comm$. In that case, because of the definition of forward transitions, we have that $top(recv^{env,i}_Q) = (D^i_Q, C)$, where C is the received environmental input.

(5) For each $i \in active$, the forward transition computed a new state for component $\mathcal{A}_i$ in Q. Therefore $\mathcal{A}_i$ could not be blocked or inactive in W^i_Q to receive multiple communications from the same sender. Consequently, as $i \in \underline{a}$ only if $i \in active$, reversed components cannot have their current result set D^i_Q on top of any *recv* stack.

Additionally, since $active \neq \emptyset$, for every forward label, we have that $\underline{a} \cap \{1, \ldots, n\} \neq \emptyset$. Hence, $\underline{a}$ is a valid label for Q.

Now we continue by examining the actual backward transition $\underline{t} : Q \xrightarrow{\underline{a}} S$. Let us first consider the *recv* stacks, where $\mathcal{A}_i$ is the recipient and $\mathcal{A}_k$ is the sender, $1 \leq i, k \leq n$, $i \neq k$. If $k \notin active$, then there is no p such that $(k, p, i) \in comm$. In that case, $k \notin \underline{a}$ either. Thus, $recv^{k,i}_S = recv^{k,i}_Q = recv^{k,i}_P$. On the other hand, if $k \in active$ and $(k, p, i) \in comm$ for some p, then $recv^{k,i}_Q = (D^i_Q, \{p \mid (k, p, i) \in comm\}) \bullet recv^{k,i}_P$. Now, because $k \in \underline{a}$, we have that $recv^{k,i}_S = pop(recv^{k,i}_Q) = recv^{k,i}_P$.

Continuing with the *send* stacks, for every $i \in \underline{a} \cap \{1, \ldots, n\}$ we know that $i \in active$ holds. In that case, if there is no p, k such that $(i, p, k) \in comm$ then $send^{i,k}_P$ is unchanged, which means that $send^{i,k}_S = send^{i,k}_Q = send^{i,k}_P$. On the other hand, if $(i, p, k) \in comm$ for some p, k then by the definition of forward transitions, $send^{i,k}_Q = D^i_Q \bullet send^{i,k}_P$. As $D^i_Q = top(send^{i,k}_Q)$, we have that $send^{i,k}_S = pop(send^{i,k}_Q)$. Consequently, $send^{i,k}_S = send^{i,k}_P$.

The last step is to consider the states of the processes. If $i \in active$ then

$$W^i_Q = \mathrm{res}_{A_i}(W^i_P) \cup \{p \mid (k, p, i) \in comm, 1 \leq k \leq n\}.$$

In this case $i \in \underline{a}$ as well, thus, $W^i_S = \mathrm{UniquePredecessor}_{A_i}(D^i_Q)$. Because $D^i_Q = \mathrm{res}_{A_i}(W^i_P)$ its unique predecessor is W^i_P. Therefore $W^i_S = W^i_P$.

Finally, if $i \notin active$ then

$$W^i_Q = W^i_P \cup \{p \mid (k, p, i) \in comm, 1 \leq k \leq n\}.$$

For every k such that there exists p such that $(k, p, i) \in comm$, we already showed that $recv_S^{k,i} = recv_P^{k,i}$. Therefore, $\text{EffectiveContext}_S^i = \text{EffectiveContext}_P^i$. Since every received communication is pushed on top of the appropriate $recv$ stack, we also have that $\text{EffectiveContext}_P^i = ComTo_P^i$. As a consequence

$$\begin{aligned} W_S^i &= D_Q^i \cup \text{EffectiveContext}_S^i = D_Q^i \cup \text{EffectiveContext}_P^i \\ &= D_Q^i \cup ComTo_P^i = (W_P^i \setminus ComTo_P^i) \cup ComTo_P^i \\ &= W_P^i, \end{aligned}$$

which concludes the proof. □

In what follows, we continue by examining the other direction: constructing forward transitions from valid backward ones.

Definition 19. Let Δ be a reversible d-cdcR system formed by $n \geq 1$ components and $\underline{t} : P \xrightarrow{\underline{a}} Q$ be a backward transition in the corresponding LTS. Then, the *inverse transition label* of $\underline{a}$ is denoted by $\underline{\underline{a}}$, and defines as

$$\underline{\underline{a}} = active; comm$$

where

$$\begin{aligned} active &= \underline{a} \cap \{1, \ldots, n\}, \\ comm &= \bigcup_{i \in \underline{a}} \text{SentBy}_i \cup \bigcup_{i,\ env_i \in \underline{a}} \text{EnvironmentTo}_i, \end{aligned}$$

and

$$\begin{aligned} \text{SentBy}_i &= \{(i, p, k) \mid (p, k) \in \text{res}_{A_i}(\text{UniquePredecessor}_{A_i}(W_Q^i))\}, \\ \text{EnvironmentTo}_i &= \begin{cases} \{(env, p, i) \mid p \in C\}, \text{ if } top(recv_Q^{env,i}) = (D_Q^i, C), \\ \emptyset, \text{ otherwise.} \end{cases} \end{aligned}$$

Let us break down the above definition into simpler terms. The set of active components is straightforward: those reversed by the inversed backward transition, $\underline{t}$. Since backward labels do not contain the reversed communications, one must determine them indirectly. In doing so, we can choose from two approaches. The first approach involves the unique predecessor of the reversed state. If we apply reactions to the unique predecessor, we can obtain the communicated symbols received in the current state. Another possibility is the use of the *send* and *recv* stacks. Since the *recv* stacks record every incoming communication of a given component, we can restore the received symbols by examining the contents of these stacks. For communication received from other components, we chose the former approach, as seen in the definition of SentBy_i. In the case of environmental input, however, EnvironmentTo_i inspects the stack contents of the reversed state.

Lemma 2. *Let $\underline{t} : Q \xrightarrow{\underline{a}} P$ be a backward transition in an LTS corresponding to some reversible d-cdcR system. Then, there exists a forward transition $\underline{t} : P \xrightarrow{\underline{\underline{a}}} Q$.*

Proof. Let Δ be a reversible d-cdcR system formed by $n \geq 1$ components and $\underline{t} : Q \xrightarrow{\underline{a}} P$ be a backward transition in the corresponding LTS. Then, let us construct the inverse transition $\underline{t} : P \xrightarrow{\underline{\underline{a}}} S$ according to Definition 19. In what follows, we prove that $\underline{t}$ is indeed the inverse of $\underline{t}$, thus $S = Q$.

We do so by first examining whether $\underline{\underline{a}}$ is a valid forward label for P with respect to the conditions of Definition 11.

(1) Since $\underline{a}$ is a valid backward label, $\underline{a} \cap \{1, \ldots, n\} \neq \emptyset$. Consequently $active = \underline{a} \cap \{1, \ldots, n\} \neq \emptyset$.
(2) Given an arbitrary state T, component $\mathcal{A}_i$ is blocked if $\text{res}_{A_i}(T) = \emptyset$. As $\underline{a}$ is a valid backward label, for each $i \in \underline{a} \cap \{1, \ldots, n\}$ we have that $D_Q^i \neq \emptyset$. Therefore, because the backward transition produced the unique predecessor of D_Q^i such that $W_P^i = \text{UniquePredecessor}_{A_i}(D_Q^i)$, component $\mathcal{A}_i$ cannot be blocked as $\text{res}_{A_i}(W_P^i) = D_Q^i \neq \emptyset$.
(3) The *comm* set of $\underline{\underline{a}}$ is comprised of two parts: the symbols communicated by the individual components, and the environmental input. Let us first consider the former. For each $i \in \underline{a} \cap \{1, \ldots, n\}$, the backward transition produces $W_P^i = \text{UniquePredecessor}_{A_i}(D_Q^i)$. SentBy_i, in turn, is defined to contain the communicated symbols of $\text{res}_{A_i}(\text{UniquePredecessor}_{A_i}(W_Q^i))$. Since that can be rewritten as $\text{res}_{A_i}(W_P^i)$, $SentBy_i$ contains precisely the symbols communicated by the enabled reactions of W_P^i. Since $active = \underline{a} \cap \{1, \ldots, n\}$, the union of the $SentBy$ sets is equal to the communicated entities of every stepped component. Thus, other elements of *comm* must be environmental context of the form (env, p, k) for some p, k.

Consequently, $\underline{\underline{a}}$ is a valid forward label for P.

Now we continue by examining the result of the forward transition $\underline{t}$. Let us first consider the *recv* stacks, where $\mathcal{A}_i$ is the recipient and $\mathcal{A}_k$ is the sender, $1 \leq i, k \leq n$, $i \neq k$. In the backward transition, if there is no k such that $recv_Q^{k,i} = (D_Q^i, C)$ for some C, we have that $recv_Q^{k,i} = recv_P^{k,i}$. Now, if $k \in \underline{a}$ then, even though $k \in active$, the forward transition will not produce any communicated symbols not present in Q, as $W_P^k = \text{UniquePredecessor}_{A_k}(D_Q^k)$. Therefore $recv_P^{k,i} = recv_S^{k,i}$ which means that $recv_S^{k,i} = recv_Q^{k,i}$. On the other hand, if there exists k such that $top(recv_Q^{k,i}) = (D_Q^i, C)$ for some C, then, as dictated by condition (2) of Definition 15, we have that $k \in \underline{a}$. Furthermore, $recv_P^{k,i} = pop(recv_Q^{k,i})$. However, since $k \in \underline{a}$, by the definition of $\underline{a}$, in this case $k \in active$ as well. Again, because the backward transition computed the unique predecessor of D_Q^k, the forward transition can only compute the very same result set, producing appropriate communicated entities. As the same is true for the component $\mathcal{A}_i$, we have that $D_S^i = D_Q^i$. Now, because $k \in active$ and $W_P^k = \text{UniquePredecessor}_{A_k}(D_Q^k)$, the component $\mathcal{A}_k$ will communicate the same symbols as in the original forward computation, resulting in $recv_S^{k,i} = (D_S^i, C) \bullet recv_P^{k,i}$ for some C. Since $D_S^i = D_Q^i$, and the new forward transition computes from the unique predecessor of D_Q^k, producing the same communicated symbols, we have that $recv_S^{k,i} = recv_Q^{k,i}$. Considering symbols from the environment, for every component $\mathcal{A}_i$, $top(recv_Q^{env,i})$ is the received environmental input. If $i \in \underline{a}$ or $env_i \in \underline{a}$ then $recv_P^{env,i} = pop(recv_Q^{env,i})$ given $top(recv_Q^{env,i}) = (D_Q^i, C_Q^{env,i})$. Since $EnvironmentTo_i$ is defined to contain exactly the symbols placed on top of $recv_Q^{env,i}$, we have that, $recv_S^{env,i} = (D_S^i, C_Q^{env,i}) \bullet recv_P^{env,i}$. As $D_S^i = D_Q^i$, $recv_S^{env,i} = recv_Q^{env,i}$ follows.

Continuing with the *send* stacks, we only need to consider components in the *active* set. By the definition of backward labels, for each $i \in active$ we have that $i \in \underline{a} \cap \{1, \ldots, n\}$ and vice versa. Given $i \in \underline{a} \cap \{1, \ldots, n\}$, if $top(send_Q^{i,k}) = D_Q^i$ for some $1 \leq k \leq n$, then $send_P^{i,k} = pop(send_Q^{i,k})$. As stated in the case of the *recv* stacks, because the backward transition produces the unique predecessors, when going forward, $\underline{t}$ must compute the reversed results sets along with appropriate communicated symbols. Thus, if $i \in active$ and $top(send_Q^{i,k}) = D_Q^i$ for some $1 \leq k \leq n$, then $send_S^{i,k} = D_S^i \bullet send_P^{i,k}$. As $send_P^{i,k} = pop(send_Q^{i,k})$ and $D_S^i = D_Q^i$, we have that $send_S^{i,k} = send_Q^{i,k}$.

The last step is to consider the states of the processes. If $i \in \underline{a} \cap \{1, \ldots, n\}$ then $W_P^i = \text{UniquePredecessor}_{A_i}(D_Q^i)$. In this case, since $i \in active$, we have that $D_S^i = \text{res}_{A_i}(W_P^i)$. Thus, $D_S^i = \text{res}_{A_i}(\text{UniquePredecessor}_{A_i}(D_Q^i)) = D_Q^i$. On the other hand, given $1 \leq i \leq n$ such that $i \notin \underline{a}$, we have that $W_P^i = D_Q^i \cup \text{EffectiveContext}_P^i$. Therefore, $D_P^i = D_Q^i$, and, because $i \notin active$, $D_S^i = D_P^i = D_Q^i$.

Now it is left to restore the symbols received from other components and the environment. We do this by showing that the forward transition $\underline{t}$ will step the exact components needed to reproduce such symbols. Because $\underline{a}$ is a valid backward label, we know that for each $1 \leq k \leq n$ such that $recv_Q^{k,i} = (D_Q^i, C)$ for some C, $k \in \underline{a}$ holds. Consequently, $k \in active$ holds as well. Since the *recv* stacks record every received communication, by reversing and then stepping forward every such component $\mathcal{A}_k$, $\underline{t}$ will reproduce the exact communication received by each component. This holds for environmental input as well, as shown in the case of $recv^{env,i}$ stacks (where $1 \leq i \leq n$). By restoring the contents of the *recv* stacks, the forward transition reproduces every communication reversed by the backward transition, resulting in $C_S^i = C_Q^i$ for each $1 \leq i \leq n$. Since for each $1 \leq i \leq n$, $D_S^i = D_Q^i$ and $C_S^i = C_Q^i$ hold, we have that $W_S^i = W_Q^i$, which concludes the proof. □

As a consequence of the above results, given any transition, regardless of its direction, we can compute its inverse. Then, by applying these transitions repeatedly, the underlying system will step through the same states again and again. This property, namely, the ability to construct diverging computation from any transition and its inverse, is called the Loop Lemma and is the foundation of the causal-consistent framework of [20].

Lemma 3 *(Loop Lemma, [11], Lemma 6). For any forward transition $t : P \xrightarrow{a} Q$ there exists a backward transition $\underline{t} : Q \xrightarrow{\underline{a}} P$, and conversely, for any backward transition $\underline{t} : Q \xrightarrow{\underline{a}} P$ there exists a forward transition $\underline{\underline{t}} : P \xrightarrow{\underline{\underline{a}}} Q$*

Proof. Follows from Lemma 1 and Lemma 2.

By establishing all the prerequisites defined in [20], the next step is to prove a set of axioms for our transition system, and therefore, the underlying reversible d-cdcR systems. Building on these axioms, our original goal, the property of causal-consistent reversibility will automatically follow.

Lemma 4 *(Square property, [20], Definition 3.1). Let $t : P \xrightarrow{\alpha} Q$ and $u : P \xrightarrow{\beta} R$ be two transitions with $t \iota u$. Then there are transitions u', t' such that $u' : Q \xrightarrow{\beta} T$ and $t' : R \xrightarrow{\alpha} T$.*

Proof. According to Definition 14, distinct coinitial transitions are independent if they are not in conflict. Since we know that t and u are independent, they cannot be conflicting. This presents the same cases as that of Definition 17.

(1) If both transitions are forward, then their labels have distinct active components and the sets of active and targeted components (those that receive communication) do not overlap. Therefore, even if we place these labels on subsequent transitions, they will not result in transitions that cause each other. In this case, if there is no causal relationship between the subsequent transitions, then the resulting process must be the same, regardless of the exact order in which we apply the labels.
(2) If both transitions are backward, they must have valid labels for the current process P. However, as the validity conditions of such labels (stated in Definition 15) revolve around the correct handling of the *send-recv* stacks and the synchronized reversal of communication, distinct labels may only be valid in the same process in two scenarios. The first scenario is if they operate on different sets of components. In such cases, the labels will not result in transitions with causal relationships. In the second scenario, the sets of reversed components overlap. However, because the validity conditions ensure the proper reversal of communications, placing the labels after each other in either order will yield the same result.
(3) If the transitions have opposite directions, the subcases of Definition 17 ensure that the labels will be valid in either order (the communication between components is respected), and the transitions yield the same results (otherwise, the transitions would be conflicting). □

Lemma 5 *(Backward transitions are independent, [20], Definition 3.1). If transitions* $t : P \xrightarrow{a} Q$ *and* $t' : P \xrightarrow{b} Q'$ *are coinitial backward transitions such that* $t \neq t'$*, then* $t \iota t'$ *holds.*

Proof. Follows immediately from the definition of the independence relation, Definition 14, and the extension of the notion of conflict in Definition 17. □

Lemma 6 *(Well-foundedness, [20], Definition 3.1). There is no infinite reverse computation: we do not have* P_i *such that* $P_{i+1} \xrightarrow{a_i} P_i$ *for all* $i \geq 0$*.*

Proof. Follows immediately from the definition of backward labels, Definition 15. □

The so called *Parabolic Lemma* describes a property common among causal-consistent models, stating that one can break down reversible computations into two parts: a backward computation, enabling the broadest range of computational choices possible, and a forward computation, arriving at the specific process. Danos and Krivine called this property parabolic since the backward part resembles a parabola, enabling the computation to draw potential energy from its history [11]. The Parabolic Lemma guarantees that reversibility does not introduce new processes that cannot be reached by pure forward computation.

Definition 20 *(Parabolic Lemma, [20], Definition 3.3).* For any path r in an LTS, there are forward-only paths s, s' such that r is causally equivalent to $\underline{s}s'$ and $|s| + |s'| \leq |r|$.

In the above definition, a *path* r is a finite sequence of subsequent transitions, both forward and backward. $|r|$ denotes the length of the path, which is equal to the count of comprising transitions.

Proposition 3. *Let* Δ *be a reversible d-cdcR system. Then, Parabolic Lemma holds for the corresponding LTS.*

Proof. Since the LTS satisfies the properties of Square Property and Backward Transitions are Independent, according to [20], Parabolic Lemma holds. □

We constructed distributed cdcR systems with the intention of introducing concurrent behavior in the framework of communicating reaction systems. Our goal was to examine how the concept of reaction systems, concurrency, and reversibility fit together. As causal-consistency establishes a connection between reversibility, causality, and concurrency, it stood out as a paradigm of particular interest. It states that computations that end in the same final state (in other words, computations that are *cofinal*) must be causally equivalent. This allows for a relaxed reversible computational order, compared to backward determinism and its concept of the last action.

Definition 21 *(Causal-consistency, [20], Definition 3.5).* Let r and s be two paths in an LTS. If r and s are coinitial and cofinal then they are causally equivalent.

Proposition 4. *Let* Δ *be a reversible d-cdcR system. Then, Causal-Consistency holds for the corresponding LTS.*

Proof. Since the LTS satisfies the properties of Well-Foundedness and Parabolic Lemma, according to [20], Causal-Consistency holds. □

6. Conclusion

We investigated two paradigms of reversibility in the realm of communicating reaction systems. First, we considered backward determinism, a strategy best suited for sequential models. Then, we continued with the concept of causal-consistency, a paradigm relating concurrency, causality, and reversibility. Since the original model lacked concurrent behavior, we defined a slightly modified version which we called distributed cdcR systems. Following the framework of [20], we proved that reversible distributed cdcR systems enjoy causal-consistent behavior.

In the future, it might also be interesting to investigate additional paradigms, such as out-of-causal order reversibility, as well as mixed-directional execution where forward and backward computations are allowed to take place in the same controlled computational step.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] A. Ehrenfeucht, G. Rozenberg, Reaction systems, Fundam. Inform. 75 (1–4) (2007) 263–280.
[2] R. Brijder, A. Ehrenfeucht, M. Main, G. Rozenberg, A tour of reaction systems, Int. J. Found. Comput. Sci. 22 (2011) 1499–1517, https://doi.org/10.1142/S0129054111008842.
[3] P. Bottoni, A. Labella, G. Rozenberg, Networks of reaction systems, Int. J. Found. Comput. Sci. 31 (2020) 53–71, https://doi.org/10.1142/S0129054120400043.
[4] E. Csuhaj-Varjú, P.K. Sethy, Communicating reaction systems with direct communication, in: R. Freund, T.-O. Ishdorj, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), Membrane Computing, Springer International Publishing, Cham, 2021, pp. 17–30.
[5] K.S. Perumalla, Introduction to Reversible Computing, Chapman & Hall/CRC, 2013.
[6] K. Morita, Theory of Reversible Computing, Springer Japan, 2017.
[7] I. Ulidowski, I. Lanese, U.P. Schultz, C. Ferreira (Eds.), Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405, Lecture Notes in Computer Science, vol. 12070, Springer, 2020.
[8] R. Landauer, Irreversibility and heat generation in the computing process, IBM J. Res. Dev. 5 (3) (1961) 183–191.
[9] C.H. Bennett, Logical reversibility of computation, IBM J. Res. Dev. 17 (6) (1973) 525–532.
[10] M. Holzer, M. Kutrib, Reversible nondeterministic finite automata, in: I. Phillips, H. Rahaman (Eds.), Reversible Computation, Springer International Publishing, Cham, 2017, pp. 35–51.
[11] V. Danos, J. Krivine, Reversible communicating systems, in: P. Gardner, N. Yoshida (Eds.), CONCUR 2004 - Concurrency Theory, Springer, Berlin, Heidelberg, 2004, pp. 292–307.
[12] I. Lanese, C. Mezzina, F. Tiezzi, Causal-consistent reversibility, Bull. Eur. Assoc. Theor. Comput. Sci. 114 (2014) 17.
[13] H. Melgratti, C.A. Mezzina, I. Ulidowski, Reversing place transition nets, Log. Methods Comput. Sci. 16 (4) (Oct. 2020), https://doi.org/10.23638/LMCS-16(4:5)2020, https://lmcs.episciences.org/6843.
[14] I. Lanese, N. Nishida, A. Palacios, G. Vidal, A theory of reversibility for Erlang, J. Log. Algebraic Methods Program. 100 (2018) 71–97, https://doi.org/10.1016/j.jlamp.2018.06.004, https://www.sciencedirect.com/science/article/pii/S2352220817301402.
[15] I. Phillips, I. Ulidowski, Reversibility and models for concurrency, Electron. Notes Theor. Comput. Sci. 192 (1) (2007) 93–108, https://doi.org/10.1016/j.entcs.2007.08.018.
[16] R. Milner, Communication and Concurrency, Prentice-Hall, Inc., USA, 1989.
[17] N. Nishida, A. Palacios, G. Vidal, Reversible computation in term rewriting, J. Log. Algebraic Methods Program. 94 (2018) 128–149.
[18] I. Lanese, C.A. Mezzina, A. Schmitt, J. Stefani, Controlling reversibility in higher-order pi, in: J. Katoen, B. König (Eds.), CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011, Proceedings, in: Lecture Notes in Computer Science, vol. 6901, Springer, 2011, pp. 297–311.
[19] E. Giachino, I. Lanese, C.A. Mezzina, F. Tiezzi, Causal-consistent rollback in a tuple-based language, J. Log. Algebraic Methods Program. 88 (2017) 99–120, https://doi.org/10.1016/j.jlamp.2016.09.003, https://hal.inria.fr/hal-01633260.
[20] I. Lanese, I. Phillips, I. Ulidowski, An axiomatic approach to reversible computation, in: J. Goubault-Larrecq, B. König (Eds.), Foundations of Software Science and Computation Structures, Springer International Publishing, Cham, 2020, pp. 442–461.
[21] B. Aman, G. Ciobanu, Controlled reversibility in reaction systems, in: M. Gheorghe, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), Membrane Computing, Springer International Publishing, Cham, 2018, pp. 40–53.
[22] B. Aman, G. Ciobanu, Mutual exclusion and reversibility in reaction systems, J. Membr. Comput. 2 (2020) 171–178, https://doi.org/10.1007/s41965-020-00043-1.
[23] A. Bagossy, G. Vaszil, Simulating reversible computation with reaction systems, J. Membr. Comput. 2 (2020) 1–15, https://doi.org/10.1007/s41965-020-00049-9.
[24] A. Bagossy, G. Vaszil, Transition graphs of reversible reaction systems, in: R. Freund, T.-O. Ishdorj, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), Membrane Computing, Springer International Publishing, Cham, 2021, pp. 1–16.
[25] A. Meski, W. Penczek, G. Rozenberg, Model checking temporal properties of reaction systems, Inf. Sci. (2015), https://doi.org/10.1016/j.ins.2015.03.048.
[26] I. Cristescu, J. Krivine, D. Varacca, A compositional semantics for the reversible pi-calculus, 2013, pp. 388–397.