Reversible Reaction Systems

Thesis for the Degree of Doctor of Philosophy (PhD)

by Attila Bagossy
Supervisor: Dr. György Vaszil

University of Debrecen
Doctoral Council of Natural Sciences and Information Technology
Doctoral School of Informatics
Debrecen, 2024

Hereby I declare that I prepared this thesis within the Doctoral Council of Natural Sciences and Information Technology, Doctoral School of Informatics, University of Debrecen in order to obtain a PhD Degree in Informatics at Debrecen University. The results published in the thesis are not reported in any other PhD theses.

Debrecen, 2024. 03. 13.

. . . . . .
Attila Bagossy

Hereby I confirm that Attila Bagossy candidate conducted his studies with my supervision within the Theoretical Computer Science, Data Security and Cryptography Programme of the Doctoral School of Informatics between 2023 and 2024. The independent studies and research work of the candidate significantly contributed to the results published in the thesis. I also declare that the results published in the thesis are not reported in any other theses. I support the acceptance of the thesis.

Debrecen, 2024. 03. 13.

. . . . . .
Dr. György Vaszil

Reversible Reaction Systems

Dissertation submitted in partial fulfilment of the requirements for the doctoral (PhD) degree in Informatics

Written by Attila Bagossy certified Computer Science MSc.

Prepared in the framework of
the Doctoral School of Informatics of University of Debrecen
Theoretical Computer Science, Data Security and Cryptography programme

Supervisor: Dr. György Vaszil

The official opponents of the dissertation:

Dr. .....
Dr. .....

The evaluation committee:

chairperson:
members:

Dr. .....
Dr. .....
Dr. .....
Dr. .....
Dr. .....

The date of the dissertation defence: 2024. .....

Contents

1. Introduction

Based on our hands-on experience with real-life phenomena, we traditionally think of computation as an unidirectional flow of actions. Regardless of potential conditional branches and loops, in our imagination, this flow is linear and strictly forward-only. This simplification is what we call a mental model: one we build to comprehend better concepts that involve many layers of complexity. In the case of computation, this model of a forward-only flow seems to align well enough with our experience. Thus, while intriguing, the idea of making this flow bidirectional (to go backward) seems just as impossible as reversing the turn of the time.

The above chain of thoughts led us to choose reversibility as our research topic. However, instead of studying reversibility in its abstract sense, we looked for an appropriate vehicle, a model of computation with which we can experiment. Because of our experience with a similar model, P Systems, its recent nature, and the prevalence of nature-inspired computational models, we chose Reaction Systems.

In this dissertation, we present our research on the reversibility of Reaction Systems. This research started with a straightforward question about defining reversibility for the ordinary, unmodified model of Reaction Systems. Then, facing the limitations of the resulting computational devices, we made sufficient modifications to increase their intuitive computational power while still working with a sequential, synchronous model and a matching paradigm of reversibility. As our initial motivation was rooted in exploring the reversibility of computation, we wished to experiment with different paradigms within the same model. Therefore, we examined another modified version of Reaction Systems, which organizes several interconnected components into a virtual graph. By removing synchronization between the components and enabling concurrent actions, we could implement a paradigm of reversibility suited for concurrent and distributed models.

In what follows, after a brief introduction to natural computing and reversibility, we provide an overview of our contribution in more formal terms. Then, the rest of the dissertation presents these contributions in detail, dedicating a self-contained section—one that comprises the necessary preliminaries as well as prior work—to each step in our research.

1.1. Natural Computing

Natural computing is a research area concerned with investigating computing inspired by nature and, dually, computing taking place in nature. The former consists of human-designed computational models and techniques, drawing inspiration from natural structures and processes. At the same time, the latter is an attempt to apply the terms of information processing to phenomena occurring in nature.

While they all belong under the umbrella term of natural computing, nature-inspired computational models are just as diverse as nature itself. Cellular automata, inspired by self-replication in biological organisms, offer a regular grid of cells, each with its own state. Then, the computation proceeds by applying previously determined rules, resulting in a new (possibly changed) state for each cell. A different take on modeling the inner workings of cells is that of membrane computing. A widespread model in this strand of research, P Systems comprise a hierarchical membrane system, compartmentalizing processes, and entities like cellular membranes. Above the level of cells, neural networks are loose models of the human brain and nervous system. Forming a spectrum, we find attempts to create artificial brains on one end and the prevalent machine learning methods, including deep neural networks and Bayesian methods, on the other. Moving away from models inspired by biological entities, quantum computing replaces the classical definitions of information processing with those inspired by the physical laws governing the level of quantum particles.

As shown by the above examples, natural computing spans an immense variety of computational models, each with its own inspiration. For a more extended introduction, refer to [30], while [53] is a comprehensive study of the field.

Our model of interest, Reaction Systems, also belongs to the group of natural computing models. Introduced by Ehrenfeucht and Rozenberg in 2007, Reaction Systems aim to capture biochemical processes inside living cells [22]. A step in such a process is modeled by applying reactions to a finite set of entities called the current state. Through the interplay of facilitation (reactions require some entities) and inhibition (other entities forbid particular reactions), a new state is produced in each computational step. Then, a series of such steps comprise an interactive process, extending the system's state with arbitrary input from the environment (hence its interactive nature). After this brief introduction to Reaction Systems, we present them in formal detail in Section 2.

1.2. Reversibility

Reversibility allows the flow of computation to be bidirectional, extending the traditional way of proceeding forward with the ability to go backward. As strange as this idea might seem, reversibility is profoundly related to information processing fundamentals.

1.2.1. Motivation and Applications

Research interest in reversibility originates from Landauer's work on information processing and heat dissipation [32]. Landauer argued that any logical operation with information loss necessarily results in heat dissipated by the system operating. Such information loss causes the computation to be irreversible. Likewise, Landauer claimed that there are computations where heat generation is inevitable, making irreversibility an inherent property of computation. Inspired by these claims, Bennett created a universal reversible Turing machine, proving that we cannot equate irreversibility and computation [11]. This result spawned extensive research interest in the field of reversible computing since it proved that reversibility is the tool to overcome the performance constraints of traditional irreversible systems [23]. A more recent and rigorous treatment of reversibility concerning Turing machine computations can also be found in [9].

Circumventing heat constraints and low-energy computing are not the only strands of research in reversibility. While we generally think of computation as a linear flow (referring back to the mental model of the introduction), producing some output from the input, in reality, it comprises branches, loops, error cases, and failures. Reversible operations can allow the system to backtrack branches in the hope of finding more promising ones, for example, in the case of state-space exploration. A similar use case is reversible debugging, in which the debugger allows forward and backward steps, permitting experimentation (for example, as in [37]). Finally, reversing the computational flow can also be a tool for system reliability, letting the system automatically recover from errors and failures. A widespread construct using this model is the idea of checkpoints and rollbacks (in the case of database software, for example).

We can also relate reversibility with natural computing, as numerous biochemical and physical phenomena share this property. In the case of biochemical reactions, environmental factors, such as temperature and pressure, can affect the direction of computation (as considered in [5]). Reversibility might even allow these reactions to revert to unvisited states, resulting in

otherwise unreachable products [31]. Quantum computing models mimic computation occurring at the quantum scale, which is naturally reversible. An interesting example in this field is that of quantum finite automata [54].

Overall, reversible computation is a field attracting interest from the points of view of several possible applications, and much work is also devoted to establishing its solid theoretical foundations. For more information on applications of reversibility, see the monograph [49], for a state-of-the-art survey of the area, see the recent collection [55].

1.2.2. Paradigms and Categorization

After discussing the motivation behind the reversibility of computation, we now briefly present how computational models might implement reversibility.

The most straightforward paradigm of reversibility, called backtracking, is best suited for sequential models such as Turing machines or most types of conventional automata. Backtracking involves recursively undoing previous actions until the desired state is recovered, building on the "last performed action" concept. How this action is determined is specific to the model but usually comprises a record of the computation history.

While backtracking is ideal for sequential models, it cannot be applied to those with concurrent computational semantics. Concurrent processes do not have a definite notion of the "last action", unlike sequential processes with a clear order of the executed computational steps. Indeed, if we attempted to establish a "global" order between the actions, we can only do so via synchronization, eliminating concurrent behavior. As some form of order is still necessary to determine which actions to undo, Danos and Krivine introduced the concept of causal consistent reversibility, which intuitively says that any action can be undone provided that all its consequences, if any, are undone already. For a survey of causal consistent reversibility, see [35].

While backtracking and causal consistency align well with our real-world experience of cause and effect, one can also define out-of-causal-order reversibility. This paradigm makes it possible to reach states that cannot be reached by forward execution alone [50].

Orthogonal to the above paradigms is the controlled versus uncontrolled nature of reversibility. An uncontrolled system might perform a forward or a backward action in any computational step. Conversely, controlled systems comprise a mechanism that decides the direction of the computation. Refer to [34] for a comprehensive study of such control mechanisms.

The ways of implementing reversibility are just as diverse as the computational models themselves. Recognizing this diversity and the need for relating and comparing the different approaches, [25] attempts to construct a comprehensive taxonomy.

1.3. Structure and Contribution

In what follows, we briefly cover the contributions of this dissertation with pointers to the appropriate sections.

Our initial motivation was to explore different paradigms of reversibility in Reaction Systems; thus, in Section 3, we first define the necessary and sufficient conditions for such systems to be reversible based on the backtracking paradigm of reversibility. Contrary to prior work, our construct also considers environmental input. Using this definition as a stepping stone, in Section 4, we extend reversible Reaction Systems with an external control mechanism. Section 5 presents an interlude on the computational power of reversible Reaction Systems. Using transition graphs as the vehicle for our research, we present the limits of what we can achieve with such a system and describe some variants to extend their computational possibilities. Following our goal of experimenting with different paradigms, Section 6 introduces Communicating Reaction Systems, a variant that comprises an arbitrary number of individual systems organized into a virtual graph. By definition, the components of such a graph perform computation synchronously; hence, we apply the backtracking paradigm. In Section 7, we introduce a variant of Communicating Reaction Systems, called Distributed Communicating Reaction Systems, that allows concurrent behavior in the model. Building on the concurrent computational semantics, we finish the dissertation by implementing causal consistent reversibility based on the axiomatic framework of [38].

2. Reaction Systems Preliminaries

This section introduces the fundamental ideas behind Reaction Systems, followed by the corresponding formal definitions. With the base model defined, we demonstrate its flexibility by presenting a few select variants, extending the fundamental definitions. Then, we conclude the section with a brief overview of prior work related to Reaction Systems reversibility.

2.1. Overview

Reaction Systems is a natural computational model by Ehrenfeucht and Rozenberg [22] aiming to provide a formal framework for investigating the biochemical reactions inside living cells. Computation in this model goes forward by applying reactions to a set of entities (called a state), creating a new set (a new state). Reaction application happens through an interplay between facilitation and inhibition. Each reaction has a set of reactants facilitating its application and a disjoint set of inhibitors blocking the reaction from occurring. Successfully applying a reaction results in its set of products.

As one may have noted, the model of Reaction Systems is entirely set-based, taking a qualitative (rather than a quantitative, multiset-based) approach to resources. The so-called *threshold assumption* states that an entity (which might represent some chemical or biological resource) is either present in a sufficient amount to participate in any number of reactions or is missing altogether. Consequently, reactions are non-conflicting, both on their consuming and producing ends. Regarding facilitation, reactions can be enabled even if their reactant sets overlap, allowing each to produce results. Then, these results are also *non-conflicting*, which means that even if two or more reactions produce the same entity (because of intersections in their product sets), there will be no "multiple occurrences" in the result set.

The subsequent application of reactions to the resulting products is called an interactive process. It is interactive because, apart from the results of the previously applied reactions, each state may also incorporate arbitrary entities as input. This input aims to capture the idea that living cells do not act in isolation but always operate in some environment that may influence their behavior (and thus, the computation). Such an interactive process is a finite sequence of steps in which new states are created rather than transformed from the previous states. The creation of new states is governed by the *no permanency* concept: if no reaction in the current step produces

a given entity and is not present in the following input, it will disappear. The intuition behind this idea comes from biochemistry, where entities are produced and sustained by active, energy-consuming processes (not to be confused with our formal interactive processes). Thus, if there is no such process to sustain a given entity, it will vanish.

2.2. Definitions

In what follows, we will briefly introduce the most important notions and notations concerning reaction systems. Our presentation is based on [14] extended with some of our own notation.

Let S denote a finite set of entities called the *background set*. A *reaction* a is a triplet of three finite sets $a = (R_a, I_a, P_a)$ where each of these sets is a subset of S. The sets R_a, I_a and P_a contain the *reactants*, *inhibitors* and *products* of the reaction, respectively, and they satisfy the following constraints: First, the set of reactants and the set of inhibitors are disjoint ($R_a \cap I_a = \emptyset$), otherwise the reaction would never be applicable, as we will see later. Second, the set of reactants and the set of products are non-empty ($R_a \neq \emptyset$ and $P_a \neq \emptyset$). It is usually also assumed that the set of inhibitors is non-empty, but for the sake of being as general as possible, we will drop this additional assumption here. The set of all reactions over S is denoted by $\mathrm{rac}(S)$.

Remark 2.1. In what follows, if a is a reaction, then we will denote its components as R_a, I_a and P_a without explicitly writing out the complete triplet form $a = (R_a, I_a, P_a)$.

Based on the core idea of the model, a reaction is applicable (or enabled) if all of its reactants and none of its inhibitors are present. Applying a reaction creates its products. These intuitions are formalized as follows.

Given a background 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.

In the previous definitions, we only considered a single reaction. It is rare, however, that a single reaction can capture the behavior of a complex process. Consequently, the concepts above should be generalized to multiple reactions.

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 $\mathrm{res}_A(W) = \bigcup_{a \in \mathrm{en}_A(W)} P_a$).

With these definitions in mind, we can see how the characteristics mentioned in Section 2.1 are implemented. Reactions with overlapping reactant sets do not interfere, as each can create its products if enabled. This non-interfering nature also applies to the products. Even if multiple reactions produce the same entity, there will still be a single occurrence in the result set, as reaction systems are defined over sets instead of multisets.

Prior to defining reaction systems and how they perform computation, we introduce further shorthand notations to ease our work with reactants and products of finite sets of reactions.

Notation 2.1. Let A be a finite set of reactions. Then, we denote by R_A and P_A the union of the reactant sets and product sets, respectively: $R_A = \cup_{a \in A} R_a$ and $P_A = \cup_{a \in A} P_a$.

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

$$\mathrm{EN}_A(S) = \{E \subseteq A \mid \text{there exists } S' \subseteq S, \text{ such that } \mathrm{en}_A(S') = E\}.$$

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, or formally

$$\mathrm{RES}_A(S) = \{\mathrm{res}_E(S') \mid S' \subseteq S, E \subseteq A, \text{ such that } \mathrm{en}_A(S') = E\}.$$

Example 2.1. Let us consider the set of reactions $A = \{a, b, c\}$ over $S = \{1, 2, 3\}$, where

$$a = (\{1\}, \emptyset, \{2\}),\ b = (\{2\}, \emptyset, \{3\}),\ c = (\{3\}, \{1\}, \{1\}).$$

Here, we have $\mathrm{EN}_A(S) = \{\, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\} \,\}$, since there is no set of elements such that a and c are enabled together (as $R_a \cap I_c \neq \emptyset$), that is, $\mathrm{en}_A(\{1\}) = \mathrm{en}_A(\{1, 3\}) = \{a\}$, $\mathrm{en}_A(\{1, 2\}) = \mathrm{en}_A(\{1, 2, 3\}) = \{a, b\}$, and we also have $\mathrm{en}_A(\{2\}) = \{b\}$, $\mathrm{en}_A(\{3\}) = \{c\}$, $\mathrm{en}_A(\{2, 3\}) = \{b, c\}$ in addition.

The elements of $\mathrm{RES}_A(S)$ are the product sets produced by the reactions in the sets of $\mathrm{EN}_A(S)$ applied to appropriate subsets of S

$$\mathrm{RES}_A(S) = \{\, \{2\}, \{3\}, \{1\}, \{2, 3\}, \{1, 3\} \,\}.$$

With the essential notions defined for reactions, we now recall the definition of a *reaction system*, which is an ordered pair $\mathcal{A} = (S, A)$. The background set S is a finite set of entities while $A \subseteq \mathrm{rac}(S)$ is the set of reactions.

Remark 2.2. Given a reaction system $\mathcal{A} = (S, A)$, we assume, without loss of generality, that A does not contain different reactions having the same set of reactants and the same set of inhibitors. (If $a, a' \in A$ are reactions with $a = (R, I, P_1)$ and $a' = (R, I, P_2)$, then they are always enabled and applied simultaneously so that they can be replaced by a single reaction $(R, I, P_1 \cup P_2)$ having the same effect.)

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 $1 \leq i \leq m$. 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 $0 \leq i \leq m$.

The above notion of an interactive process is visualized in Figure 2.1. Note that even though the figure shows the context set C_i and the result set D_i of the same state as non-overlapping rectangles, they do not have to be disjoint.

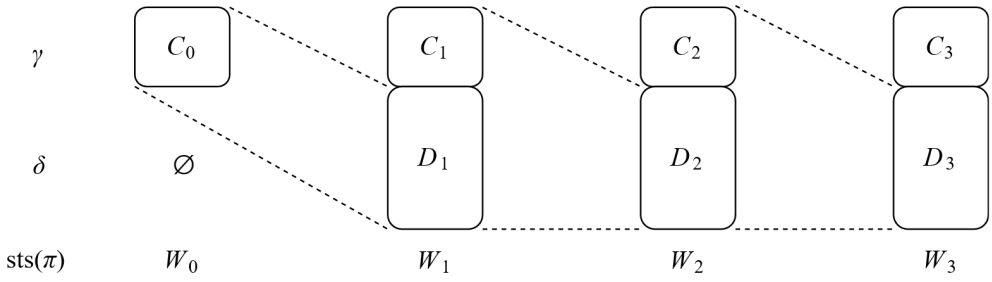

Figure 2.1: An interactive process π in a reaction system.

As a consequence of the above definition, every interactive process is finite with a predetermined length. Therefore, an interactive process can be considered a finite sequence of states (the union of contexts and results). In such a sequence, the idea of environmental input is formalized by the context sets. A process with input is called *context-dependendent* while those in which every context set is empty are said to be *context-independent*.

2.3. Variants

While the preceding definitions might seem relatively simple, they merely act as a foundation for building additional layers as necessary. This flexibility is best demonstrated by the vast array of variants developed since the inception of Reaction Systems. In what follows, we briefly introduce a few of these variants.

Interactive processes capture the idea of environmental input through the sequence of context sets. These sets, by definition, may contain any element of the background set. This property, however, might be excessively permissive when modeling various phenomena. To overcome the permissive nature of the context sets, Męski, Penczek, and Rozenberg introduced the concept of *Context Restricted Reaction Systems* [41]. Such a system is a triplet $\mathcal{A} = (S, A, \varepsilon)$, where S is the background set, A is the set of reactions, as usual, and ε is the set of context entities. Then, given an n-step interactive process $\pi = (\gamma, \delta)$ in A, we have that $C_0, \ldots, C_{n-1} \subseteq \varepsilon$. Since D_0 is empty by definition, restricting context sets like this leads to unreachable states: interactive processes can only compute starting from a subset of ε. As this might be an undesired side-effect of context restriction, [41] extends the model further through *Initialized Context Restricted Reaction Systems*. In such a system, given an n-step interactive process, D_0 is still empty; however, $C_0 \subseteq S$, allowing to start from an arbitrary state.

Another direction regarding environmental input is introducing two-way communication between the system and its environment. Such interactions are commonplace in biological systems, meaning even the system can influence its enclosing environment. [13] implements this idea in *System-Environment Reaction Systems*. In such a system, the context sets are no longer arbitrary but are determined by the result sets of the process. Formally, given a background set S, we have a so-called environment function $\psi : \mathscr{P}(S) \to \mathscr{P}(S)$ (where $\mathscr{P}$ denotes power sets) that produces the subsequent context sets from the appropriate products. This interaction between the product entities and the context can happen immediately (the current result set determines the current context set) or after a possibly entity-specific delay (entities in the current result set determine later context sets).

A fundamental aspect of Reaction Systems is the rule of no permanency. If there is no reaction to produce a given entity, it disappears in the next step of the containing interactive process. However, as highlighted in [15], entities in organic chemistry have a so-called decay time. Such entities vanish only after a given period, even in the lack of a sustaining biochemical process.

Reaction Systems with Duration, introduced in [15], model this decay time through a duration function d, which assigns a "lifetime" to each entity in the background set. A given entity x is sustained for $d(x) \geq 1$ steps from the one where it was created (including the creator state itself). Thus, $d(x) = 1$ for every entity will yield the original definition of Reaction Systems. Suppose a reaction produces an entity while it is still "alive," then the lifetime of the entity is reset.

2.4. Reversibility

In the previous section, we briefly studied a few selected extensions to Reaction Systems. Now, we turn our attention to reversibility, giving a concise overview of prior work regarding the reversibility of Reaction Systems. Then, we present these in detail in the appropriate sections.

Aman and Ciobanu extensively studied the topic in [5] and [6]. In [5], they developed an extension to Reaction Systems that implements controlled backtracking with environmental control over the direction of computation. This model, however, forbids the use of context entities apart from a designated rollback symbol. In [6], they introduce another variant to the base model, Restricted Reaction Systems, that operate with mutually exclusive reactions. When both reactions are enabled in such a mutually exclusive pair, the model makes a non-deterministic choice of which one to apply. Then, they relate Restricted Reaction Systems to the base model and present a study of uncontrolled reversibility for both.

Holzer and Rauch approached reversible Reaction Systems from a different direction, studying the computational complexity of different problems (such as reachability) [28]. Their definition of a reversible system states that it is reversible if the underlying state transition function is bijective.

Another, not necessarily reversible, but connected branch of research is the study of Quantum Reaction Systems, introduced by Hirvensalo [26]. In the quantum model, reactions produce quantum sets (a generalization of fuzzy sets) instead of traditional sets of entities. However, interactive processes are only defined for systems with a single reaction, as combining the results of multiple reactions is problematic.

3. Uncontrolled Backtracking in Reaction Systems

Our journey toward distributed reversibility starts with a much simpler yet foundational step: defining reversible Reaction Systems in terms of uncontrolled backtracking. In this section, we first give an overview of the backtracking paradigm, followed by a description of uncontrolled reversibility. Then, we introduce necessary and sufficient conditions regarding the reversibility of Reaction Systems. We conclude the section with an overview (and comparison) of prior work.

The results of this section were initially published in [J2].

3.1. Paradigms of Reversibility: Backtracking

Biochemically inspired computational models, even when abstract and very much simplified, naturally include some concurrency and parallelism between their different possible computational processes. Suitable examples are membrane systems [47, 48] which deal with multisets of symbols processed in the compartments of hierarchical structure of membranes according to some multiset rewriting rules: some of the symbols are changed in parallel according to the rules associated with their containing regions, while the others remain unchanged (and can be used in the subsequent steps) or get moved to other regions of the membrane structure.

The concurrent computation mode, however, poses particular questions concerning reversibility. Opposed to sequential processes (like the computations of Turing machines or most types of conventional automata) where the order of the execution of the computational steps can easily be reversed by undoing the last action, the definition of the backward execution of a collection of concurrently executing distributed processes is not straightforward at all, since there is no definite notion of the "last action" which should be undone first. For example, in the case of membrane systems, [1, 2] defines reversibility as a form of duality, while in [7] the reversibility of biochemical reactions in parallel rewriting systems are investigated (which can easily represent classes of models such as membrane systems, or Petri nets). In a more recent paper [52], membrane system configurations are enriched with a memory recording the information necessary for reversing steps.

The situation in the case of Reaction Systems, however, is different. Although they are motivated by (and, in a certain sense, can also be used

to model) biochemical processes, they represent a qualitative model. As opposed to systems being able to "count" by dealing with entities having multiplicities (as in multisets, for example), Reaction Systems deal with sets, which distinguishes them from the models mentioned above by (i) a *threshold assumption*: if a resource is present, it is always present in a sufficient amount necessary for any reaction; (ii) *no permanency assumption*: if an entity is not produced at a particular step, it will not be available for use at the next step [22].

Due to the qualitative nature of their definition, Reaction Systems offer parallelism on the level of individual computational steps. Since reactions are non-conflicting, regarding both their reagents and products, we can apply them in parallel (independent of each other) to construct the next state of the system. Nonetheless, we find that the model is sequential when we examine its computational properties on the level of interactive processes. Such a process is a finite series of states, computing one after the other, very close to how finite transition systems (or finite automata) operate. Therefore, the overall model is sequential despite the low-level parallelism of applying reactions.

The concept of reversibility in the context of these types of computational models is relatively straightforward. A model is reversible if it is "backward deterministic", that is, if each of its computational configurations (or states) has at most one predecessor, or in other words, no state is accessible from two distinct states[1]. In such a model, we can reverse computation by backtracking our way through the predecessor states to the desired state.

This definition of reversibility is closely related to the notion of state determinism. Nagy introduces state determinism in the context of $5' \to 3'$ Watson-Crick automata, saying that an automaton is state-deterministic if every transition from a particular state leads to the same state [44]. Consequently, backtracking is state determinism over the backward transitions.

As simple as this definition of backtracking is, it gives rise to different implementational paradigms among the actual models, some of which we will shortly review in the following. Our presentation is based mainly on [43, 49].

As noted in Section 2.4, the origins of reversible models of computation can be traced back to Bennett's universal reversible Turing machine, which proved that irreversibility is not an inherent property of computation [11].

[1] A similar approach is also possible in the case of membrane systems by considering deterministic variants, as done in [3, 29], or in [46] where reversible register machines are simulated.

When constructing his model, he developed the so-called Compute-Copy-Uncompute paradigm, which comprises the following steps: The machine performs a reversible forward computation, resulting in the desired output. Then, it creates a copy from this output. Finally, the reverse execution of the forward steps (also reversible) cleans up the effects of the forward computation, leaving the original input and the copy of the corresponding output on the tapes. (See [49] for a more detailed and systematic treatment of this paradigm.)

While the Compute-Copy-Uncompute paradigm fits the power consumption-related study of reversibility well, it might be too static for others since the outcome of the computation is of the most importance, as opposed to the actual process of the computation itself. If we focus on the processes, however, we can discover another significant implication of reversibility: it allows for exploration and experimentation. Since every configuration has at most one predecessor, we can freely undo any previous computation and proceed by choosing a different computational route.

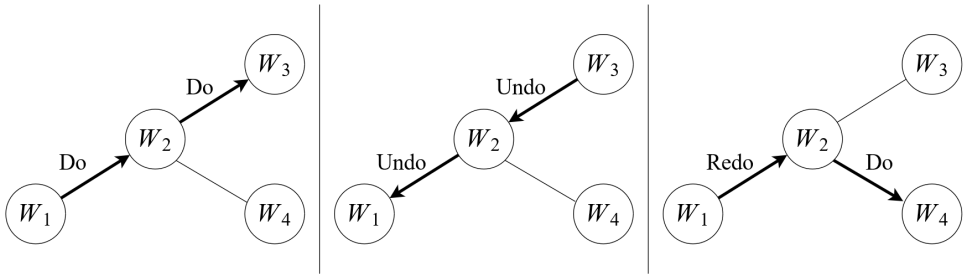

Figure 3.1: Altering the flow of computation in the Undo-Redo-Do paradigm.

This idea is the basis of the Undo-Redo-Do paradigm, depicted in Figure 3.1. Below, we briefly describe the computation flow in this paradigm, as discussed in [49].

- The *Do* operation corresponds to normal forward computation.
- At any stage, we can choose to *Undo* our previous step, essentially reversing the execution, taking us back to the single predecessor of the current state.
- Later, if we wish to recover our prior computation (thus, visiting the same states as before), we can perform a *Redo*.

- Instead of recovering previous actions, if we want to experiment by taking a different route, we can dismiss any undone steps yet to be redone and continue with a *Do* operation.

A similar approach was taken in [27] where reversible nondeterministic automata were investigated, and although in a different context, the idea of exploration and experimentation was also discussed in [21].

When considering reversibility in the case of (sequential) models with an emphasis on interaction with some external environment (such as reaction systems), the implementation of a paradigm like Undo-Redo-Do seems to be more suitable than, for example, Compute-Copy-Uncompute since it is well-aligned with the dynamic and exploratory characteristics of these models. Moreover, as we already mentioned, the process-focused nature of the paradigm (in contrast to the result-oriented focus of Compute-Copy-Uncompute) also motivates its use in the following investigations.

3.2. Reversibility without Control

While the underlying paradigm and implementation strategy defines *how* computational steps should be reversed, another critical question is deciding *when* reversal should occur. Without an explicitly defined control mechanism, the model might perform a forward or a backward computational step at any given point, possibly diverging virtually any computation.

Nevertheless, the above approach of lack of control called uncontrolled reversibility, is still valuable. When constructing the reversible counterpart of an existing, forward-only model, a widely used strategy first enables proper backward steps. Then, once the desired properties hold for the reverse mode of operation, one can follow up with another extension that adds a suitable control mechanism. For example, such a two-step approach is employed in [33] using a rollback symbol as the control mechanism of choice or in [20] with irreversible actions.

When defining Reversible Reaction Systems, we will follow the same path: first, we define uncontrolled reversibility in the next section, extended by control over the direction of computation in Section 4.

3.3. Uncontrolled Backtracking with Environmental Input

Based on the discussions in Section 3.1, we aim to interpret the notion of backward determinism in the framework of Reaction Systems without any foundational modifications or extensions. In this model, state sequences correspond to interactive processes, so we start by defining how unique predecessors can be applied to the states of interactive processes.

Definition 3.1. Let $\mathcal{A} = (S, A)$ be a reaction system and $\pi = (\gamma, \delta)$ be an interactive process in $\mathcal{A}$, such that $\gamma = C_0, C_1, \ldots C_n$ and $\text{sts}(\pi) = W_0, W_1, \ldots W_n$.

A state W_i, $1 \leq i \leq n$, has *multiple predecessors* if there exists $W'_{i-1}, C'_{i-1} \subseteq S$ such that $W'_{i-1} \neq W_{i-1}$, but $\text{res}_A(W'_{i-1}) \cup C'_{i-1} = W_i$. If there is no such W'_{i-1}, then W_i has a *unique predecessor*.

The interactive process π is *reversible*, if every state W_i, $1 \leq i \leq n$, has a unique predecessor.

In the following, we will discuss the conditions necessary for obtaining reversible interactive processes. It is clear that if different reactions produce the same result, then we can arrive at the same state from different predecessor states. Moreover, since a state is the union of a context set and a result set, the property of having a unique predecessor does not only depend on the reactions and the results of the reactions but also on the context sets which are added in each step of the interactive process. Even if every result set was unique, identical states could be created with well-chosen contexts.

In order for an interactive process to be reversible, there must be at most one way to produce each context-result union, so the unique predecessor property of Definition 3.1 depends not only on the reactions of the system but also on the context sets of the process. Since this dependency is rather involved, we will follow a step-by-step approach and introduce lemmas to describe the different requirements related to the reversibility of interactive processes.

First, let us consider the following. According to the general notion of reversibility, every state must have a unique predecessor. Nevertheless, when a given state does not enable any reactions, the process continues with an empty result. We can then augment this empty result with a non-empty context, restarting the process. Reversing execution from such an empty or restarting state would be equivalent to obtaining "something from nothing" which comes with a few issues.

When considering the issues of reversing the computation from an empty result set, the first one is a relatively apparent technical roadblock: we need symbols to apply backward reactions. Otherwise, we need to alter the definition of reactions, allowing empty reagent sets. While possible, this approach does not fit the model.

By definition, interactive processes start with the empty result set. If this set has a predecessor, an interactive process might start by immediately computing backward. Consequently, the system would create states that have not happened yet.

Finally, a system with restarting may only be reversible if the empty result set has a unique predecessor. In such a case, however, if we depicted the possible computational paths of the system, we would only get loops (in Section 4, we will explore transition graphs to represent the computation possible in a system concisely).

To overcome the above issues and limitations, in what follows, we will only consider processes in which restarting computation does not occur.

Definition 3.2. Let $\mathcal{A}$ be a reaction system and $\pi = (\gamma, \delta)$ be an interactive process in $\mathcal{A}$ such that $\delta = D_0, D_1, \ldots D_n$. The interactive process π is *non-restarting* if $D_i \neq \emptyset$, $1 \leq i \leq n$. If the opposite holds, π is *restarting*.

Remark 3.1. If $\pi = (\gamma, \delta)$ is an interactive process with $\delta = D_0, D_1, \ldots D_n$ and $\mathrm{sts}(\pi) = W_0, \ldots, W_n$ in some $\mathcal{A} = (S, A)$, then (since the product sets of the reactions are non-empty) $D_i = \emptyset$ is only possible, if $\mathrm{en}_A(W_{i-1}) = \emptyset$ for some $1 \leq i \leq n$. Thus, π is non-restarting, if and only if, $\mathrm{en}_A(W_i) \neq \emptyset$ for all i, $0 \leq i \leq n-1$.

We already noted that reversibility implies the uniqueness of result sets. This implication comes from the fact that regardless of the context, if there are multiple ways to produce the same set of entities, then it is not possible to recover the predecessor.

Example 3.1. Let $S = \{0, 1, 2, 3, 4\}$ be a set of entities and $A = \{a, b, c\}$ be a set of reactions where

$$a = (\{0\}, \emptyset, \{3\}),\ b = (\{0, 1\}, \emptyset, \{4\}),\ c = (\{2\}, \emptyset, \{3, 4\}).$$

If we consider the set $W = \{3, 4\}$, we can see that there are multiple subsets of reactions in A producing W, for example, $\{a, b\}$ and $\{c\}$. As a consequence, just by looking at W, we cannot determine which reactions

produced it. Equivalently, if W were a state in some interactive process, then this process would not be reversible since we could not recover the predecessor of W.

Deriving from the above example, we can impose the following requirement on the set of reactions. If we take every possible subset of reactions in which each reaction is enabled, then no two subsets should produce the same result. As the following lemma states, if two different enabled reaction sets produce the same result set, then a state with multiple predecessors exists.

Lemma 3.1. *Let $\mathcal{A} = (S, A)$ be a reaction system. If there exist $E_1, E_2 \in \mathrm{EN}_A(S)$ with $E_1 \neq E_2$, such that $P_{E_1} = P_{E_2}$, then there exists a state W in some interactive process in $\mathcal{A}$, such that W has multiple predecessors.*

Proof. Let $\mathcal{A} = (S, A)$ be a reaction system. Assume, that there exists $E_1, E_2 \in \mathrm{EN}_A(S)$, $E_1 \neq E_2$, such that $P_{E_1} = P_{E_2}$.

Because $E_1, E_2 \in \mathrm{EN}_A(S)$, we have $T_1, T_2 \subseteq S$ satisfying $\mathrm{en}_A(T_1) = E_1$ and $\mathrm{en}_A(T_2) = E_2$. In addition, since $E_1 \neq E_2$, we also have $T_1 \neq T_2$. This means that there exist $T_1, T_2 \subseteq S$ such that $T_1 \neq T_2$ and $\mathrm{res}_A(T_1) = \mathrm{res}_A(T_2)$, since $P_{E_1} = P_{E_2}$.

Thus, given some context set $C \subseteq S$, if $\mathrm{res}_A(T_1) \cup C = W$, then we also have $\mathrm{res}_A(T_2) \cup C = W$ with $T_1 \neq T_2$. That being so, if W is a state in some interactive process, then W has multiple predecessors.

To see that such an interactive process always exists, consider $\pi = W_0, W_1, \ldots W_n$, $n \geq 0$, with $W_0 = C_0$, where $C_0 = T_1$ (or $C_0 = T_2$) is the initial context set. Then, since $W_1 = \mathrm{res}_A(C_0) \cup C_1$, the state W_1 in the interactive process π is a state with multiple predecessors. □

Now, one might be tempted to conclude that the assumption of Lemma 3.1 provides a sufficient condition for reversibility in the case of context-independent interactive processes (those with empty contexts except for C_0). Since there is a single reaction set producing every result set and the context is always empty (hence, $W_i = D_i$ for all $i \geq 1$), every state should have a unique predecessor. However, because of the *no permanency* assumption in reaction systems, this is not necessarily the case. According to the no permanency assumption, entities not sustained by at least one reaction will disappear, resulting in states with multiple predecessors even in context-independent interactive processes.

Example 3.2. Let $S = \{0, 1, 2\}$ be a set of entities and $A = \{a, b\}$ be a set of reactions where

$$a = (\{0\}, \emptyset, \{1\}),\ b = (\{0, 1\}, \emptyset, \{2\})$$

so Lemma 3.1 does not apply to this system.

Consider the sets $W_1 = \{0, 1, 2\}$ and $W_2 = \{0, 1\}$. If we apply the reactions in A to these sets, we get $\mathrm{res}_A(W_1) = \mathrm{res}_A(W_2) = \{1, 2\}$. Thus, if $W = \{1, 2\}$ is a state in some context-independent interactive process, then W has multiple predecessors.

In the above example, the non-uniqueness of the predecessor was caused by a vanishing entity (the entity 2), which was not a reactant of any of the enabled reactions. Given this observation, we might conjecture that the presence of entities not contained by the reactant sets of any of the reactions should imply the existence of states with multiple predecessors. The following example shows that this is not necessarily so.

Example 3.3. Let $S = \{0, 1\}$ be a set of entities and $A = \{a, b\}$ be a set of reactions where

$$a = (\{0\}, \emptyset, \{0\}),\ b = (\{0\}, \{1\}, \{1\}).$$

Consider the set $W_1 = \{0\}$. If we apply the reactions in A to W_1, we get $\mathrm{res}_A(W_1) = \{0, 1\}$. Given this result set, we can deduce that the applied reactions were a and b and can restore the original set W_1.

Now, let us take the set $W_2 = \{0, 1\}$. Since the entity 1 is not a reactant of the reactions, it will disappear when any of the above reactions are applied. In this case, however, we can only apply a since 1 is an inhibitor of b, so we get $\mathrm{res}_A(W_2) = \{0\}$. Even though the element 1 has vanished because none of the reactions sustained it, we can deduce its presence in the predecessor set of $\{0\}$ because it inhibited the reaction b.

As the above example demonstrates, facilitation (being a reactant) is not the only thing that leaves a trace. Since inhibitors also affect the result of reaction application, we might be able to recover them from the result set. Thus, the presence of an entity that is not a reactant of any applied reaction in some state of an interactive process does not necessarily imply the existence of multiple predecessors. With this in mind, we can reformulate our

previous observation: The "problematic states" contain entities whose presence or absence does not affect the set of enabled reactions. The following statement expresses this idea.

Lemma 3.2. *Let $\mathcal{A} = (S, A)$ be a reaction system. If there exists a set $T \subseteq S$, $\mathrm{en}_A(T) \neq \emptyset$, and an entity $e \in T$, such that*

$$\mathrm{en}_A(T \setminus \{e\}) = \mathrm{en}_A(T),$$

then the there is a state W in some interactive process in $\mathcal{A}$, such that W has multiple predecessors.

Proof. Assume, that there exist $T_1 \subseteq S$, $e \in T_1$, such that $\mathrm{en}_A(T_1 \setminus \{e\}) = \mathrm{en}_A(T_1)$, and let $T_2 = T_1 \setminus \{e\}$. This means that $\mathrm{res}_A(T_1) = \mathrm{res}_A(T_2)$, so if $W = \mathrm{res}_A(T_1) \cup C$ is a state of an interactive process for some context set $C \subseteq S$, then $T_1 \neq T_2$, but both T_1 and T_2 are predecessors of W.

To see that such an interactive process always exists, consider $\pi = W_0, W_1, \ldots W_n$, $n \geq 0$, with $W_0 = C_0$, where $C_0 = T_1$ (or $C_0 = T_2$) is the initial context set. Then, since $W_1 = \mathrm{res}_A(C_0) \cup C_1$, the state W_1 in the interactive process π is a state with multiple predecessors. □

The lemma above implies that whenever the same set of reactions is enabled by two or more different sets of entities, there is a state with multiple predecessors in the corresponding interactive processes.

Corollary 3.1. *Let $\mathcal{A} = (S, A)$ be a reaction system. If there exist $T_1, T_2 \subseteq S$, $T_1 \neq T_2$, such that $\mathrm{en}_A(T_1) = \mathrm{en}_A(T_2) \neq \emptyset$, then there exists a state W in some interactive process in $\mathcal{A}$ such that W has multiple predecessors.*

As we briefly noted, when discussing Definition 3.1, the notion of a unique predecessor depends on both the reactions of the containing system and the context sets of the enclosing process. Given an appropriate set of reactions, it might still be possible to construct states with multiple predecessors, even if none of the above lemmas are applicable. To see this, consider the following. When assembling a new interactive process, we can make arbitrary choices regarding the elements of the context sets. Consequently, for every pair of distinct result sets, we can always choose an appropriate context set so that the union of these sets will be equal.

Example 3.4. Let $S = \{0, 1\}$ be a set of entities and $A = \{a, b, c\}$ be a set of reactions where

$$a = (\{0\}, \{1\}, \{0, 1\}),\ b = (\{1\}, \{0\}, \{0\}),\ c = (\{0, 1\}, \emptyset, \{1\}).$$

None of the previous lemmas apply to this system, but it still produces a state with multiple predecessors. Consider the states $W = \{1\}$ and $W' = \{0\}$. As $\mathrm{res}_A(W) = \{0\}$ and $\mathrm{res}_A(W') = \{0, 1\}$, if we have a state $W_i = \{0, 1\}$ of an interactive process for some $i \geq 1$ with $C_i = \{1\}$, then W_i has multiple predecessors: Since $W_i = \mathrm{res}_A(W) \cup C_i = \mathrm{res}_A(W') \cup C_i = \{0, 1\}$, the predecessor of W_i can be any of the states W or W'.

As the context sets can be arbitrary subsets of the background set (even the background set itself can be a context), regardless of how well-chosen our reactions are, an appropriate context set can turn a state into one with multiple predecessors. Thus, we need to restrict which entities may appear in the contexts, or, in other words, there must be entities that cannot appear in any context set. To this aim, we write the background set S of a reaction system as the union of two not necessarily disjoint sets, the *product alphabet* $\Sigma_p \subseteq S$ (entities that appear in the product sets of the reactions) and the *context alphabet* $\Sigma_c \subset S$ (entities that can appear in the context sets). The model we obtain this way is similar to the one called Context Restricted Reaction Systems in [41], briefly introduced in Section 2.3.

Notice, however, that if we restrict the sets of possible contexts, there might be states that are not "reachable" in the sense that they cannot appear in any interactive process.

Definition 3.3. Let $\mathcal{A} = (S, A)$ be a reaction system with $S = \Sigma_p \cup \Sigma_c$ that is, the background set being the (not necessarily disjoint) union of Σ_p (entities that are allowed to appear as products of reactions) and Σ_c (the entities that are allowed to appear in the context sets).

A *state* $W \subseteq S$ *is reachable* if there exists an interactive process $W_0, W_1, \ldots, W_n$ in $\mathcal{A}$ with $W = W_i$ for some $0 \leq i \leq n$, such that $W_i = D_i \cup C_i$ with $D_i \subseteq \Sigma_p$, $C_i \subseteq \Sigma_c$, and $D_0 = C_n = \emptyset$.

The following statement establishes a relationship between the properties of the reaction sets, the context alphabet, and the existence of states with multiple predecessors.

Lemma 3.3. *Let $\mathcal{A} = (S, A)$ be a reaction system with $S = \Sigma_p \cup \Sigma_c$ (where Σ_p and Σ_c are not necessarily disjoint). If there exist $R_1, R_2 \in \mathrm{RES}_A(S)$ such that $R_1 \neq R_2$, $R_1 = \mathrm{res}_A(W)$ for some state $W \subseteq S$ which is reachable in $\mathcal{A}$, and*

$$R_1 \setminus \Sigma_c = R_2 \setminus \Sigma_c,$$

then there exists a state with multiple predecessors in some interactive process in $\mathcal{A}$.

Proof. Let $\mathcal{A} = (S, A)$ be a reaction system with $S = \Sigma_p \cup \Sigma_c$. Assume that there exist $R_1, R_2 \in \mathrm{RES}_A(S)$ satisfying the conditions of the statement.

As R_1 and R_2 are in $\mathrm{RES}_A(S)$, there exist $W, T \subseteq S$ such that $\mathrm{res}_A(W) = R_1$, $\mathrm{res}_A(T) = R_2$, and W is a reachable state in $\mathcal{A}$. (Note that $W \neq T$, since given a fixed set of reactions, different result sets may only be created from different states.) Furthermore, since $R_1 \neq R_2$ but $R_1 \setminus \Sigma_c = R_2 \setminus \Sigma_c$, if we choose the context set C as $C = (R_1 \cap \Sigma_c) \cup (R_2 \cap \Sigma_c)$, then we have

$$R_1 \cup C = R_2 \cup C,$$

so there exist $W, T \subseteq S$, $W \neq T$, and $C \subseteq \Sigma_c$ such that,

$$\mathrm{res}_A(W) \cup C = \mathrm{res}_A(T) \cup C = W'$$

for some state $W' \subseteq S$.

Since W is reachable in $\mathcal{A}$, there is an interactive process π, such that W' is a state in π, and as $W' = \mathrm{res}_A(W) \cup C = \mathrm{res}_A(T) \cup C$ with $W \neq T$, the state W' in π has multiple predecessors. □

To see the condition of the previous statement from a different point of view, we may also formulate it as follows.

Corollary 3.2. *Let $\mathcal{A} = (S, A)$ be a reaction system with $S = \Sigma_p \cup \Sigma_c$ (where Σ_p and Σ_c are not necessarily disjoint). If there exist $R_1, R_2 \in \mathrm{RES}_A(S)$ such that $R_1 \neq R_2$, $R_1 = \mathrm{res}_A(W)$ for some state $W \subseteq S$ which is reachable in $\mathcal{A}$, and*

$$(R_1 \cup R_2) \setminus (R_1 \cap R_2) \subseteq \Sigma_c,$$

then there exists a state with multiple predecessors in some interactive process in $\mathcal{A}$.

Since computation in Reaction Systems is done using interactive processes, we can naturally formalize the definition of Reversible Reaction Systems based on the reversibility of interactive processes.

Definition 3.4. A *reaction system $\mathcal{A}$ is reversible*, if every non-restarting interactive process in $\mathcal{A}$ is reversible.

Based on the lemmas above, we can formulate the necessary and sufficient conditions for the reversibility of a reaction system as follows.

Theorem 3.1. *Let $\mathcal{A} = (S, A)$ be a reaction system with $S = \Sigma_p \cup \Sigma_c$ (where Σ_p and Σ_c are not necessarily disjoint). The system $\mathcal{A} = (S, A)$ is reversible if and only if the following conditions hold.*

(1) For all $E_1, E_2 \in EN_A(S)$,

$$E_1 \neq E_2 \text{ implies } P_{E_1} \neq P_{E_2}.$$

(2) For all $T_1, T_2 \subseteq S$, $\mathrm{en}_A(T_1) \neq \emptyset$,

$$T_1 \neq T_2 \text{ implies } \mathrm{en}_A(T_1) \neq \mathrm{en}_A(T_2).$$

(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 $\mathcal{A}$,

$$R_1 \neq R_2 \text{ implies } R_1 \setminus \Sigma_c \neq R_2 \setminus \Sigma_c.$$

Proof. According to the Lemma 3.1, Corollary 3.1, and Lemma 3.3, no reaction system can be reversible if any of the above conditions does not hold.

To also see that the conditions imply the reversibility of a system, let us indirectly assume that there is a reaction system $\mathcal{A} = (S, A)$ (where $S = \Sigma_p \cup \Sigma_c$) which satisfies all three conditions of the theorem, but is not reversible. As $\mathcal{A}$ is not reversible, there is a non-restarting interactive process $\pi = (\gamma, \delta)$ in $\mathcal{A}$ with $\mathrm{sts}(\pi) = W_0, \ldots, W_n$, where $W_i = D_i \cup C_i$ with $D_i \subseteq \Sigma_p$, $C_i \subseteq \Sigma_c$, $0 \leq i \leq n$, such that there is an $i \geq 1$ for which W_i has multiple predecessors, that is,

$$\mathrm{res}_A(W) \cup C = \mathrm{res}_A(W_{i-1}) \cup C_i = W_i \text{ for some } W \neq W_{i-1}, \qquad (1)$$

with $W \subseteq S$, and $C, C_i \subseteq \Sigma_c$.

Since $W \neq W_{i-1}$, we have $\mathrm{en}_A(W) \neq \mathrm{en}_A(W_{i-1})$ according to condition (2), and then $P_{\mathrm{en}_A(W)} \neq P_{\mathrm{en}_A(W_{i-1})}$, that is,

$$\mathrm{res}_A(W) \neq \mathrm{res}_A(W_{i-1})$$

according to condition (1). Since W_{i-1} is reachable in $\mathcal{A}$, condition (3) is applicable, which implies

$$\mathrm{res}_A(W) \setminus \Sigma_c \neq \mathrm{res}_A(W_{i-1}) \setminus \Sigma_c.$$

This means that $\mathrm{res}_A(W)$ and $\mathrm{res}_A(W_{i-1})$ differ also in entities that are not in Σ_c, therefore, there is no $C, C' \subseteq \Sigma_c$ such that

$$\mathrm{res}_A(W) \cup C = \mathrm{res}_A(W_{i-1}) \cup C'$$

which contradicts our assumption at (1), and thus completes the proof. □

Example 3.5. Let $\mathcal{A} = (S, A)$ be the reaction system with $S = \Sigma_c \cup \Sigma_p$, $\Sigma_p = \{1, 3, 5\}$ being the product alphabet, $\Sigma_c = \{0, 2, 4\}$ being the context alphabet, and $A = \{a, b, c\}$ being the set of reactions, where

$$a = (\{0\},\ \{1, 2, 3, 4, 5\},\ \{1\}),\ b = (\{1, 2\},\ \{0\},\ \{3\}),\ c = (\{1, 4\},\ \{0\},\ \{5\}).$$

According to Theorem 3.1, $\mathcal{A}$ is reversible as it satisfies all three conditions.

In a reaction system satisfying the conditions of Theorem 3.1, every non-restarting interactive process is reversible, even in the presence of input from the environment in the form of context sets. Therefore, given any state of some non-restarting interactive process, the predecessor of this state is unique and can be restored. Using this environmental input, however, one can only control the forward computation of the system. In Section 4, we are going to extend these results so that the environment can also trigger backward computation.

3.4. Related Work

Aman and Ciobanu have also studied uncontrolled reversibility in the framework of Reaction Systems [6]. First, to enable reversibility, they introduce the notion of reverse reactions. One can obtain such a reaction by exchanging the reactants and products. Thus, given a reaction $a = (R_a, I_a, P_a)$, its

reverse is $\tilde{a} = (P_a, I_a, R_a)$. This definition, however, can produce reactions that are impossible to apply if there is an overlap between the forward reaction's inhibitors and products, as $R_a \cap I_a = \emptyset$ is assumed but $P_a \cap I_a = \emptyset$ is not. Therefore, their work resolves this problem by requiring $P_a \cap I_a = \emptyset$ for all reactions.

However, the previous definition of backward reactions is insufficient to implement backtracking reversibility even in this special setting. When applying reactions, a symbol might not be a reagent of any reaction. Consequently, it will not be present in the products of the appropriate backward reactions, making it impossible to reproduce the original starting set during backtracking. To overcome the limitation of such symbols vanishing without a trace, Aman and Ciobanu introduced memory external to the state set of the system. This piece of memory is a register T that records and backfills the vanishing symbols. In our solution, we wished to implement backtracking without resorting to any form of external memory or counting. Thus, our definition does not allow symbols to disappear without participating in a reaction as a reagent or an inhibitor.

Then, Aman and Ciobanu consider systems that contain both forward and reverse reactions. Given an ordinary reaction system $\mathcal{A} = (S, A)$, we can define such a system as $\tilde{\mathcal{A}} = (S, A \cup \{\tilde{a} \ : \ a \in A\})$. When examining the behavior of these systems, they first consider the case when both forward and backward reactions can be applied in the same step. This approach is rather interesting, as it can produce states that do not exist originally. As one of their main results, they prove that an equilibrium of entities can be achieved given that the system is insensitive to context ($C_i \subseteq D_i$ for all $i \geq 1$ in an interactive process).

As enabling forward and reverse reactions in the same step is somewhat unconventional, Aman and Ciobanu define uncontrolled reversibility for so-called Restricted Reaction Systems. This model extends the base framework by mutually exclusive reactions: given a pair of such reactions, we must choose which one to apply if both are enabled non-deterministically. This machinery is implemented via a relation $\#$ containing the mutually exclusive reactions, such that $\# \subseteq \{(a, b) | a, b \in A, a \neq b\}$.

By utilizing the above relation, we can overcome the problem of applying forward and reverse reactions in the same computational step. For this end, given a reaction a and its reverse pair $\tilde{a}$, we can say $(a, \tilde{a} \in \#)$. Then, Aman and Ciobanu prove that context-independent computation in such a Restricted Reaction System adheres to the Loop Lemma: backward steps are the inverse of forward ones and vice versa. Hence, the computation can

indefinitely loop between the same two states.

Compared to our contribution, where we define uncontrolled reversibility as the backward determinism of the interactive process steps, Aman and Ciobanu enable reversibility by adding reverse reactions and a register to track disappearing symbols. A further difference is the approach to the environmental input. Our model considers context sets with elements from a context alphabet. Aman and Ciobanu, on the other hand, work with context-independent processes, those for which context sets are empty. Thus, our model can accommodate a wider variety of interactive processes.

4. Controlled Backtracking in Reaction Systems

As discussed in Section 3.1, we believe that the Undo-Redo-Do paradigm fits well to the reversible variants of computational models with environmental interaction, as opposed to other, more static paradigms (such as Compute-Copy-Uncompute). However, the reversible reaction systems of Theorem 3.1 do not support this kind of controlled reversibility. To this aim, we also need to be able to construct "reverse reactions," which are reactions that execute the backward computations of reversible reaction systems. In this section, we discuss the interactive simulation of such systems to enable interactive environmental control over the direction of the computation.

The section is organized as follows. First, we give an overview of the main strategies for controlling reversibility. Then, we detail the interactive simulator construction. Finally, we conclude the section by considering prior work.

The results of this section were initially published in [J2].

4.1. Reversibility with Control

Models implementing uncontrolled reversibility define only the *how* of backward computation, but they give no hint about *when* the direction of computation should change. Hence, forward and backward steps can freely come after each other, resulting in an essentially non-deterministic model. While such a model is of little use from a practical standpoint, it can still act as a stepping stone when defining control over the direction of computation. In what follows, we give an overview of three main control strategies using the work of Lanese, Mezzina, and Stefani [34].

In the case of *internal control*, the computational process is in charge of deciding when to go forward or backward. One implementation of this approach is the use of irreversible actions. We can see such actions as one-way gates: once performed, they cannot be reversed. However, while they limit the ability of the system to go backward non-deterministically, they do not provide precise control over when the direction of computation should change. To this end, one can introduce a piece of dual machinery in the form of explicit rollback. When the system reaches an undesired or erroneous configuration, it can emit a rollback primitive requiring it to go backward. Then, the direction of computation changes once the reversal flow reaches

an irreversible action that forbids further backward steps.

While internal control is entirely self-contained, *external control* assumes the existence of two processes: one that might perform forward or backward steps and one that supervises the direction of the computation. Thus, we have a hierarchy of processes in which the supervisor process can detect and reverse the undesired steps of the computing process.

Finally, *semantic control* extends the semantics of the computation itself to decide whether a forward or a backward step should be chosen. For example, we can consider the case of state-space exploration: computation can backtrack if a branch is not "promising" for further exploration. The control over the direction of the computation is provided via the definition of promising branches and states.

4.2. Simulating Reversible Reaction Systems

Since Reaction Systems are inherently interactive with environmental control over the computation, we decided to augment this interactivity with external control over the direction of computation. The system will perform a backward computation when a designated rollback symbol is present in the context. On the other hand, the absence of this symbol signals a forward computation. In what follows, we construct so-called simulator systems for the Reversible Reaction Systems of Section 3.3. They are simulators because they freely compute the backward and forward steps of an underlying uncontrolled system while extending it with the aforementioned environmental control.

When creating such systems, we will extensively use interactive processes and, consequently, finite sequences of sets. To ease notation, here we will introduce some new notations regarding such sequences.

Notation 4.1. Let $\mathcal{W} = W_0, W_1, \ldots, W_n$ be a finite sequence of $n+1$ sets. Then

- the *length of* $\mathcal{W}$ is denoted by $|\mathcal{W}|$,
- the *reverse of* $\mathcal{W}$ is denoted by $\overleftarrow{\mathcal{W}}$ and is defined as $\overleftarrow{\mathcal{W}} = W_n, W_{n-1}, \ldots, W_0$.

For a finite set D, the finite sequence obtained by *subtracting D from each set* in $\mathcal{W}$ is denoted by $\mathcal{W} \setminus D$, that is, $\mathcal{W} \setminus D = W_0 \setminus D, W_1 \setminus D, \ldots, W_n \setminus D$.

For a finite sequence of sets $\mathcal{M} = M_0, M_1, \ldots, M_m$ where $m \leq n$, we say that $\mathcal{W}$ *contains* $\mathcal{M}$, denoted by $\mathcal{M} \subseteq \mathcal{W}$, if $\mathcal{M}$ is a consecutive subsequence of $\mathcal{W}$, or formally, there exists $0 \leq i \leq n - m$, such that for all $0 \leq j \leq m$ we have $W_{i+j} = M_j$.

For finite sequences of sets $\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_n$, $n \geq 0$, the finite sequence $\mathcal{W}$ obtained by *concatenating* these sequences is denoted by $\mathcal{W} = \mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_n$.

The intuition behind the simulator reaction systems is relatively simple. If we take some interactive process in the simulator, then state sets of this process can be divided into one or more shorter subsequences. The first subsequence always represents a forward computation of the simulated system. Then, the second subsequence corresponds to a backward computation of the simulated system, undoing some actions previously computed by the first subsequence. Afterward, another forward subsequence follows, which is, in turn, equivalent to a series of Redo and Do operations. Backward computation does not occur by accident but is controlled by the environment using a special auxiliary symbol ρ. When ρ is present in the context, the simulator undoes its last computation, simulating a backward step of the original simulated system. This intuition is formalized in the following two definitions.

Definition 4.1. Let $\mathcal{A}$ be a reaction system, $\pi = (\gamma, \delta)$ be an interactive process in $\mathcal{A}$, with $\gamma = C_0, C_1, \ldots, C_n$, $\delta = D_0, D_1, \ldots, D_n$, and let $\rho \in S$ be a special entity in the background set. We say that the interactive process π is a *well-formed simulating interactive process* if the following conditions hold for every $0 \leq i \leq n$:

- If $D_i \subseteq \Sigma_c$, then $\rho \notin C_i$.
- If $\rho \in C_i$, then $C_i = \{ \rho \}$.

The well-formedness of simulator interactive processes is a necessity since ρ cannot appear in the context arbitrarily. If a result set is empty or consists solely of entities from the context alphabet of the simulated system, then we have reached an initial state. In this case, backward computation makes no sense, as initial states lack predecessors. Additionally, if ρ is present in the context, then regardless of any other input entity, a simulated backward step will take place. As only entities in the product alphabet of the simulated system influence the backward computation, any unnecessary context entity other than ρ is forbidden.

Definition 4.2. Let $\mathcal{A} = (S, A)$ and $\mathcal{B} = (S \cup \{\rho\}, B)$ be two reaction systems, with ρ being an auxiliary symbol in the background set of $\mathcal{B}$ (forcing the system to compute a simulated backward step).

The system $\mathcal{B}$ *interactively simulates* the interactive processes of $\mathcal{A}$ if the following conditions hold.

(1) For every interactive process π in $\mathcal{A}$, there is a well-formed simulating interactive process σ in $\mathcal{B}$ such that $\mathrm{sts}(\pi) = \mathrm{sts}(\sigma)$.

(2) For every well-formed simulating interactive process σ in $\mathcal{B}$, $\mathrm{sts}(\sigma)$ can be written as $\mathrm{sts}(\sigma) = \mathcal{V}_0, \ldots, \mathcal{V}_k$, where each $\mathcal{V}_i$ is a finite sequence of sets such that:

- $\mathcal{V}_0 = \mathrm{sts}(\pi)$ for some interactive process π in $\mathcal{A}$.
- If $i = 2m$ for some $m \geq 1$, then there exists an interactive process π in $\mathcal{A}$ such that $\mathcal{V}_i \subseteq \mathrm{sts}(\pi)$.
- If $i = 2m + 1$ for some $m \geq 0$, then $\rho \in W$ for every W in $\mathcal{V}_i$ and there exists an interactive process π in $\mathcal{A}$, such that $\bigcup \mathcal{V}_i \setminus \{\rho\} \subseteq \mathrm{sts}(\pi)$.

According to the above definition, the simulating system can compute everything that the original simulated system can compute (because it includes all of its interactive processes). Furthermore, the simulator can traverse back and forth among the states of the interactive processes of the simulated system. Backward computation is initiated by the auxiliary symbol ρ. The appropriate subdivision of the states of the interactive processes captures this notion of back-and-forth traversal.

Now, our goal is to show that the Undo-Redo-Do paradigm of reversibility can be achieved for reversible reaction systems using the above definition of interactive simulation. In what follows, we first show how to construct appropriate simulator systems and then prove that they adhere to the requirements of Definition 4.2.

Definition 4.3. Let $\mathcal{A} = (S, A)$ (with $S = \Sigma_p \cup \Sigma_c$) be a reversible reaction system. A reaction system $\mathcal{B}$, called the *interactive undo-redo simulator* of $\mathcal{A}$ is constructed as follows.

Let $\mathcal{B} = (S \cup \{\rho\}, B)$ where $B = \overrightarrow{B} \cup \overleftarrow{B}$ such that

$$\overrightarrow{B} = \{\, (R_a,\, I_a \cup \{\, \rho \,\},\, P_a) \mid a \in A \,\},$$
$$\overleftarrow{B} = \{(P_E \cup \{\rho\},\, \mathrm{CONT}_A(S,E),\, R_E \cup \mathrm{DI}_A(S,E)) \mid E \in EN_A(S)\},$$

where $\mathrm{CONT}_A(S,E)$ is defined as

$$\mathrm{CONT}_A(S,E) = \bigcup_{\substack{F \in \mathrm{EN}_A(S) \\ P_E \subset P_F}} P_F \setminus P_E$$

and $\mathrm{DI}_A(S,E)$ is defined as

$$\mathrm{DI}_A(S,E) = \bigcup_{\substack{F \in \mathrm{EN}_A(S) \\ E \neq F,\ R_F = R_E}} I_F \setminus I_E.$$

The set $\overrightarrow{B}$ consists of reactions implementing forward computational steps of $\mathcal{A}$ (if ρ is not present in the context, the reactions of A can also be performed in the simulating system), while the reactions in $\overleftarrow{B}$ implement the simulated backward computational steps of $\mathcal{A}$.

The intuition behind the constructions of the backward reactions of $\overleftarrow{B}$ can be summarized as follows. For each simultaneously applicable set of reactions $E \in \mathrm{EN}_A(S)$, the presence of the product set P_E of E implies that a backward reaction produces the reactants of E, possibly together with the elements of $\mathrm{DI}_A(S,E)$ in addition. The set $\mathrm{DI}_A(S,E)$ contains those entities that might have also been necessary to make only the reactions of E simultaneously enabled by inhibiting other reactions that could have also been applied. This is how $\mathrm{DI}_A(S,E)$ is constructed: It contains the inhibitors of those simultaneously applicable sets of reactions that have the same set of reactants as E since these entities must have been present in the predecessor state (otherwise not E, but some other set of reactions would have been applied). There is also a set of inhibitors added to the backward simulating reactions, the set $\mathrm{CONT}_A(S,E)$, which is necessary, because there might be different sets of simultaneously enabled reactions $E_1, E_2 \in \mathrm{EN}_A(S)$, such that $P_{E_1} \subset P_{E_2}$, that is, the product set of E_1 is a subset of the product set of E_2. In this case, performing the reactions of E_1 backward should only be possible if the elements of $E_2 \setminus E_1$ are absent. (The presence of these entities would indicate that the state for which the predecessor should be produced was not the result of applying the reactions of E_1, but the reactions of E_2 instead.)

Example 4.1. Consider the reversible reaction system $\mathcal{A} = (S, A)$ with $\Sigma_p = \{1, 3, 5, 7\}$ being the product alphabet, $\Sigma_c = \{0, 2, 4\}$ being the context alphabet, and $A = \{a, b, c, d\}$ being the set of reactions

$$a = (\{0\}, \{1, 2, 3, 4, 5\}, \{1\}), \qquad c = (\{1, 4\}, \{0, 3, 5\}, \{5\}),$$
$$b = (\{1, 2\}, \{0, 5\}, \{3\}), \qquad d = (\{1, 2\}, \{0, 3\}, \{3, 7\}).$$

Based on Definition 4.3, the interactive undo-redo simulator $\mathcal{B} = (S \cup \{\rho\}, B)$ interactively simulating $\mathcal{A}$ is constructed as follows.

Let the $\{0, 1, 2, 3, 4, 5, 7\} \cup \{\rho\}$ be the background set and $B = \overrightarrow{B} \cup \overleftarrow{B}$ be the set of reactions where forward reactions are defined as

$$\overrightarrow{B} = \{\ (\{0\}, \{1, 2, 3, 4, 5, \rho\}, \{1\}),\ (\{1, 2\}, \{0, 5, \rho\}, \{3\}),$$
$$(\{1, 4\}, \{0, 3, 5, \rho\}, \{5\}),\ (\{1, 2\}, \{0, 3, \rho\}, \{3, 7\})\ \}.$$

To construct the backward reactions, consider

$$\mathrm{EN}_A(S) = \{\ \{a\}, \{b\}, \{c\}, \{b, d\}, \{b, c, d\}\ \}.$$

Then we compute

$$\mathrm{CONT}_A(S, \{a\}) = \emptyset, \qquad \mathrm{CONT}_A(S, \{b, d\}) = \{5\},$$
$$\mathrm{CONT}_A(S, \{b\}) = \{5, 7\}, \qquad \mathrm{CONT}_A(S, \{b, c, d\}) = \emptyset,$$
$$\mathrm{CONT}_A(S, \{c\}) = \{3, 7\},$$

and

$$\mathrm{DI}_A(S, \{a\}) = \emptyset, \qquad \mathrm{DI}_A(S, \{b, d\}) = \{0, 3, 5\},$$
$$\mathrm{DI}_A(S, \{b\}) = \{3\}, \qquad \mathrm{DI}_A(S, \{b, c, d\}) = \{0, 3, 5\}.$$
$$\mathrm{DI}_A(S, \{c\}) = \emptyset,$$

Based on these, the set of backward reactions is

$$\overleftarrow{B} = \{\ (\{1, \rho\}, \emptyset, \{0\}), (\{3, \rho\}, \{5, 7\}, \{1, 2, 3\}),$$
$$(\{5, \rho\}, \{3, 7\}, \{1, 4\}), (\{3, 7, \rho\}, \{5\}, \{1, 2, 5\})\ \}.$$

Just as in the case of Reversible Reaction Systems, we are going to take a step-by-step approach to prove that interactive undo-redo simulator systems constructed using Definition 4.3 are indeed interactive simulators in the sense of Definition 4.2. Taking an arbitrary well-formed simulating interactive process σ in a simulator system, we subdivide the state sequence of σ into smaller subsequences, and for each subsequence of interest, we prove that it satisfies the appropriate condition of Definition 4.2.

Lemma 4.1. *Let $\mathcal{A}$ be a reversible reaction system, let $\mathcal{B}$ be the interactive undo-redo simulator of $\mathcal{A}$, and let σ be a well-formed simulating interactive process in $\mathcal{B}$.*

Then, the state sequence of σ can be written as $\mathrm{sts}(\sigma) = \mathcal{V}_0\mathcal{V}_1$, where $\mathcal{V}_0$ is a finite sequence, such that $\mathcal{V}_0 \setminus \{\rho\} = \mathrm{sts}(\pi)$ for some interactive process π in $\mathcal{A}$, that is, $\mathcal{V}_0 \setminus \{\rho\}$ is the state sequence of a forward computation of the simulated system $\mathcal{A}$.

Proof. Let $\mathcal{A} = (S, A)$ ($S = \Sigma_p \cup \Sigma_c$) be a reversible reaction system, let $\mathcal{B}$ be the interactive undo-redo simulator constructed from $\mathcal{A}$ using Definition 4.3, and let σ be a well-formed simulating interactive process in $\mathcal{B}$ with $\mathrm{sts}(\sigma) = W_0, W_1, \ldots, W_n$.

Since σ is a well-formed simulating interactive process, there exists $0 \leq m_0 \leq n$ such that $\rho \notin C_i$, $0 \leq i \leq m_0$. This means that σ starts with a finite sequence of states such that the length of this sequence is greater than or equal to one, and none of the sets in the sequence includes ρ.

Now we prove by induction on the value of m_0 that, given the previous condition, there exists an interactive process π in $\mathcal{A}$ such that $\mathrm{sts}(\pi) = W_0, W_1, \ldots, W_{m_0}$ when $m_0 = n$, or $\rho \in W_{m_0+1}$ and $\mathrm{sts}(\pi) = W_0, W_1, \ldots, W_{m_0}, W_{m_0+1} \setminus \{\rho\}$ when $m_0 < n$. Thus, ρ might be present in the $(m_0 + 1)$st context; nevertheless, with ρ not taken into consideration, the state can still be part of some interactive process in $\mathcal{A}$.

Given the well-formedness of σ, we can be sure that $\rho \notin C_0$ i.e. $C_0 \subseteq \Sigma_c$. Hence, for the initial state $W_0 = C_0$ of σ there exists some interactive process π in $\mathcal{A}$ sharing the very same initial state. Thus, for $m_0 = 0$, the previous statement is true.

Assuming that the statement holds for an arbitrary $m_0 = k$, we now show that it also holds for $m_0 = k + 1$. In this case, we know that there exists some interactive process π in $\mathcal{A}$ such that $\mathrm{sts}(\pi) = W_0, W_1, \ldots, W_k$. This

implies that $W_k \subseteq S$. Let us now enumerate the reactions applicable to this set. As $\rho \notin W_k$, one can only apply reactions from $\overrightarrow{B}$, since ρ is a reactant in every reaction of $\overleftarrow{B}$. Note that $\overrightarrow{B}$ contains the same reactions as A with a small difference: the inhibitor sets were augmented with ρ. In our current case, however, this role of ρ is irrelevant since $\rho \notin W_k$. Consequently, we have that $\mathrm{res}_A(W_k) = \mathrm{res}_B(W_k) = D_{k+1}$. Thus, there exists some interactive process $\pi = (\gamma_\pi, \delta_\pi)$ in $\mathcal{A}$ such that $\delta_\pi = D_1, D_2, \ldots D_k, D_{k+1}$.

Let us now consider the context. As $\rho \notin C_{k+1}$ we know that $C_{k+1} \subseteq S$. Since $D_{k+1} \subseteq S$ and $C_{k+1} \subseteq S$ we have that $W_{k+1} \subseteq S$ and, in turn, there exists some interactive process π in $\mathcal{A}$ such that $\mathrm{sts}(\pi) = W_0, W_1, \ldots, W_{k+1}$.

We now continue by considering $m_0 = k + 1 < n$, meaning that the next state starts a new, simulated backward computation. In this case $C_{m_0+1} = \{\rho\}$. On the other hand, since we proved that there exists some interactive process in $\mathcal{A}$ such that the state sequence of the process is equal to the sequence $W_0, W_1, \ldots, W_{m_0}$, then it also holds, that extending this sequence with $W_{m_0+1} = \mathrm{res}_A(W_{m_0}) \cup \{\rho\}$, there exists some interactive process in $\mathcal{A}$ such that the state sequence of the process is equal to $W_0, W_1, \ldots, W_{m_0}, W_{m_0+1} \setminus \{\rho\}$.

This means that the statement holds for $m_0 = k + 1$, which, in turn, renders our initial statement true. The argument above also implies that every interactive process in $\mathcal{A}$ is an interactive process in $\mathcal{B}$ as well, since both the C_i context sets and the length k of the initial forward computing sequence can be chosen arbitrarily. □

Now, we continue by showing that after the initial state sequence simulates a forward computation, the well-formed simulating interactive processes might continue with the simulation of backward computations.

Lemma 4.2. *Let $\mathcal{A}$ be a reversible reaction system, let $\mathcal{B}$ be the interactive undo-redo simulator of $\mathcal{A}$, and let σ be a well-formed simulating interactive process in $\mathcal{B}$.*

Then, the state sequence of σ can be written as $\mathrm{sts}(\sigma) = \mathcal{V}_0\mathcal{V}_1$, where $\mathcal{V}_0 \setminus \{\rho\}$ is the state sequence of a forward computation of the simulated system $\mathcal{A}$, and $\mathcal{V}_1$ can be written as $\mathcal{V}_1 = \mathcal{V}_2\mathcal{V}_3$ where $\mathcal{V}_2$ is a finite sequence such that there exists some interactive process π in $\mathcal{A}$ for which $\overleftarrow{\mathcal{V}_2} \setminus \{\rho\} \subseteq \mathrm{sts}(\pi)$. Thus, $\mathcal{V}_2 \setminus \{\rho\}$ is a state sequence of a backward computation of the simulated system $\mathcal{A}$.

Proof. Let $\mathcal{A} = (S, A)$ ($S = \Sigma_p \cup \Sigma_c$) be a reversible reaction system, and let $\mathcal{B}$ be the interactive undo-redo simulator constructed from $\mathcal{A}$ using Definition 4.3, and let σ be a well-formed simulating interactive process in $\mathcal{B}$ with $\mathrm{sts}(\sigma) = W_0, W_1, \ldots, W_n = \mathcal{V}_0\mathcal{V}_1$, where $\mathcal{V}_0 \setminus \{\rho\}$ is the state sequence of a forward computation of the simulated system $\mathcal{A}$. Since $\mathcal{V}_0$ is covered by Lemma 4.1, we know that there exists $m_0 < n$ such that the finite sequence $W_0, W_1, \ldots W_{m_0}$ is the state sequence of some interactive process π in $\mathcal{A}$. In this case, for some $m_0 < m_1 \leq n$, the well-formedness of σ implies that $\rho \in C_i$ for $m_0 + 1 \leq i \leq m_1$, and if $m_1 < n$, then $\rho \notin C_{m_1+1}$.

Now we prove by induction on the value of m_1 that there exists some interactive process π in $\mathcal{A}$ for which the $\mathcal{V}_1 = W_{m_0+1}, W_{m_0+2}, \ldots, W_{m_1}$ sequence is such that $\overleftarrow{\mathcal{V}_1} \setminus \{\rho\} \subseteq \mathrm{sts}(\pi)$ holds.

First, let us prove the statement for $m_1 = m_0 + 1$ when the sequence consists of a single element. In this case, $\rho \in C_{m_1}$ and, implied by the proof of Lemma 4.1, there exists some interactive process π in $\mathcal{A}$ such that $\mathrm{sts}(\pi) = W_0, W_1, \ldots W_{m_0+1} \setminus \{\rho\}$. Accordingly, if $\overleftarrow{\mathcal{V}_1} = W_{m_1}$, then $\overleftarrow{\mathcal{V}_1} \setminus \{\rho\} \subseteq \mathrm{sts}(\pi)$. Thus, for $m_1 = m_0 + 1$, the statement holds.

Let us now assume, that the statement holds if $m_1 = m_0 + k$, and then prove that it also holds for $m_1 = m_0 + k + 1$. We know that the state sequence $\mathcal{V}_2 = W_{m_0+1}, W_{m_0+2}, \ldots W_{m_0+k}$ is such that $\overleftarrow{\mathcal{V}_2} \setminus \{\rho\} \subseteq \mathrm{sts}(\pi)$ for some interactive process π in $\mathcal{A}$, furthermore, $C_{m_0+k} = \{\rho\}$ and $C_{m_1} = \{\rho\}$.

Building on the previous facts, let us define W_{m_1}. Since $\rho \in W_{m_0+k}$, only reactions in $\overleftarrow{B}$ can be applied to W_{m_0+k} (as the presence of ρ forbids the application of reactions in $\overrightarrow{B}$). $\mathcal{A}$ is reversible, therefore no two reaction sets $E_1, E_2 \in \mathrm{EN}_A(S)$ produces the same result set. Thus, given an arbitrary result set $D = P_E$ for some $E \in \mathrm{EN}_A(S)$, we are able to restore W such that $\mathrm{res}_{\mathcal{A}}(W) = D$. In order to do so, we take every reaction in E and create a new reaction with a reactant set equal to $D = P_E$ and a product set equal to $W = R_E$. However, care should be taken, as there might be one or more $F \in \mathrm{EN}_A(S)$ reaction set such that $P_E \subset P_F$. If so, then elements not in P_E must be forbidden by including every element in $P_F \setminus P_E$ in the inhibitor set of the newly created reaction. This is exactly how $\mathrm{CONT}_A(E)$ is defined. That way, we will not falsely apply reactions because of the subset relationships. Our job is not done yet, however, since we must also consider inhibitors. As demonstrated in Example 3.3, inhibitors might also leave a

trace by inhibiting some reactions and thus affecting the result. Hence, given the reactant set R_E, we need to find reaction sets with the same reactants. Then, the inhibitors of the reactions in these sets should also be restored since their presence denied the application of these reactions. Such inhibitor entities are captured by $\mathrm{DI}_A(S, E)$ in Definition 4.3, and as a result, we will have a product set $R_E \cup \mathrm{DI}_A(S, E)$.

Turning back to the definition of $\mathcal{B}$, we can see that reactions in $\overleftarrow{B}$ are constructed using this very method. What remains is to find the reaction in $\overleftarrow{B}$ that can be applied to W_{m_0+k}. Such a reaction always exists and is always unique: unique, since $\mathcal{A}$ is reversible, which means that there is only a single way to produce every result set, and exists, since $\overleftarrow{\mathcal{V}_2} \setminus \{\rho\} \subseteq \mathrm{sts}(\pi)$ for some interactive process π in $\mathcal{A}$, which means that the first set in $\overleftarrow{\mathcal{V}_2} \setminus \{\rho\}$, that is, $W_{m_0+k} \setminus \{\rho\}$ was indeed produced from some state by applying some set of reactions in A. The only case in which the previous would fail is if $W_{m_0+k} \setminus \{\rho\}$ was the initial state; however, the well-formedness of σ forbids this case.

Concluding the previous reasoning, we have that the only reaction applicable to W_{m_0+k} is a uniquely determined reaction $b \in \overleftarrow{B}$ for which $R_b = W_{m_0+k} \setminus \{\rho\}$ holds. By applying this reaction to $R_b = W_{m_0+k} \setminus \{\rho\}$ we obtain $P_b = D_{m_1}$ such that $\mathrm{res}_A(D_{m_1}) = W_{m_0+k} \setminus \{\rho\}$. Since $C_{m_1} = \{\rho\}$, we have that $W_{m_1} = P_b \cup \{\rho\}$. As a consequence, these states can be written as a sequence $\mathcal{V}_2 = W_{m_0+1}, W_{m_0+2}, \ldots, W_{m_0+k}, W_{m_1}$ such that $\overleftarrow{\mathcal{V}_2} \setminus \{\rho\} \subseteq \mathrm{sts}(\pi)$ for some interactive process π in $\mathcal{A}$ (as $\mathcal{V}_2$ is essentially an appropriate continuation of $\mathcal{V}_1$ for which we already proved the same). This means that the statement holds for $m_1 = m_0 + k + 1$, which, in turn, renders our initial statement true. □

Now, we combine the previous two statements to obtain the following.

Lemma 4.3. *Let $\mathcal{A}$ be a reversible reaction system, let $\mathcal{B}$ be the interactive undo-redo simulator of $\mathcal{A}$, and let σ be a well-formed simulating interactive process in $\mathcal{B}$, such that*

- $\mathrm{sts}(\sigma) = \mathcal{V}_0\mathcal{V}_1$, *where $\mathcal{V}_0 \setminus \{\rho\}$ is the state sequence of a forward computation of $\mathcal{A}$, and*
- *$\mathcal{V}_1$ can be written as $\mathcal{V}_1 = \mathcal{V}_2\mathcal{V}_3$, where $\mathcal{V}_2 \setminus \{\rho\}$ is a state sequence of a backward computation of $\mathcal{A}$.*

Then, if $\mathcal{V}_3$ is not of zero-length, subdividing $\mathcal{V}_3$ into smaller subsequences will result in forward and backward computations of the simulated system $\mathcal{A}$, analogous to those covered by Lemma 4.1 and Lemma 4.2.

Proof. Let $\mathcal{A} = (S, A)$ ($S = \Sigma_p \cup \Sigma_c$) be a reversible reaction system, let $\mathcal{B}$ be the interactive undo-redo simulator constructed from $\mathcal{A}$ using Definition 4.3, and let σ be a well-formed simulating interactive process in $\mathcal{B}$ with $\text{sts}(\sigma) = W_0, W_1, \ldots, W_n = \mathcal{V}_0\mathcal{V}_1$, where $\mathcal{V}_0 \setminus \{\rho\}$ is the state sequence of a forward computation of $\mathcal{A}$, and $\mathcal{V}_1 = \mathcal{V}_2\mathcal{V}_3$, where $\mathcal{V}_2 \setminus \{\rho\}$ is a state sequence of a backward computation of $\mathcal{A}$.

Since $\mathcal{V}_2$ is covered by Lemma 4.2, we know that there exists $m_0 < m_1 < n$ such that the finite sequence $W_{m_1} \setminus \{\rho\}, W_{m_1-1} \setminus \{\rho\}, \ldots W_{m_0+1} \setminus \{\rho\}$ is the state sequence of some interactive process π in $\mathcal{A}$. We now show that the proof for subsequent sequences in σ is analogous to those of Lemma 4.1 and Lemma 4.2.

Now, because of the well-formedness of σ, we have that $C_{m_1} = \{\rho\}$ and $\rho \notin C_{m_1+1}$. As a consequence of the former, we have that D_{m_1+1} is the result of a simulated backward step, which means that $\text{res}_{\mathcal{A}}(D_{m_1+1}) = W_{m_1} \setminus \{\rho\}$. Also, $W_{m_1+1} = D_{m_1+1} \cup C_{m_1+1}$. As for the sequence $\mathcal{V}_4 = W_{m_0+1}, W_{m_0+2}, \ldots W_{m_1}, D_{m_1+1}$, there exists some interactive process π in $\mathcal{A}$ such that $\mathcal{V}_4 \setminus \{\rho\} \subseteq \text{sts}(\pi)$, we have two possibilities regarding D_{m_1+1}:

- $D_{m_1+1} \subseteq \Sigma_c$, thus D_{m_1+1} is equal to a restarting or initial state of π, or
- there exists some state W in π such that $D_{m_1+1} \setminus \Sigma_c = \text{res}_A(W)$.

In both of the above cases, however, the sequence $\mathcal{V}_5 = D_{m_1+1}$ is contained within the sequence of states of π. Since D_{m_1+1} can only be extended with elements from Σ_c to form W_{m_1+1} then there is going to be an interactive process τ in $\mathcal{A}$ such that the sequence of states of τ contains W_{m_1+1}. Suppose we apply the induction for forward computations (from Lemma 4.1). In that case, we can say that given the subsequent contexts do not contain ρ, the newly started sequence of states is going to adhere to Definition 4.2 (i.e., there exists some interactive process in $\mathcal{A}$ with an appropriate sequence of states containing this sequence).

Now, suppose some C_j context set contains ρ. In that case, we can apply the same induction as in Lemma 4.2, that is, starting from a set that is

included within the state sequence of some interactive process in $\mathcal{A}$, we can see that only valid backward steps can be simulated.

Therefore, we have shown that any further subsequence of σ is either a proper forward or backward computation of the simulated system $\mathcal{A}$. □

Combining the three previous lemmas, we can state the following.

Theorem 4.1. *For every reversible reaction system $\mathcal{A}$, the interactive undo-redo simulator constructed from $\mathcal{A}$ using Definition 4.3 interactively simulates the interactive processes of $\mathcal{A}$.*

Proof. Let $\mathcal{A} = (S, A)$ ($S = \Sigma_p \cup \Sigma_c$) be a reversible reaction system, let $\mathcal{B}$ be the interactive undo-redo simulator constructed from $\mathcal{A}$ using Definition 4.3, and let σ be a well-formed simulating interactive process in $\mathcal{B}$

Given the previous assumptions, Lemma 4.1, Lemma 4.2 and Lemma 4.3 can be applied. In these lemmas, by subdividing the state sequence of σ, we showed that σ satisfies the requirements of condition (2) in Definition 4.2. Since our choice of σ (apart from the well-formedness) is arbitrary, this also means that every well-formed interactive process in $\mathcal{B}$ satisfies the requirements of condition (2). When considering the first subsequence of σ in Lemma 4.1, we also showed that well-formed interactive processes in $\mathcal{B}$ may start with any interactive process in $\mathcal{A}$. Thus, for every interactive process in $\mathcal{A}$, we have an interactive process in $\mathcal{B}$ with the very same state sequence. Consequently $\mathcal{B}$ satisfies condition (1) in Definition 4.2.

This implies that $\mathcal{B}$ interactively simulates the interactive processes of $\mathcal{A}$. □

Based on the above proof, we can construct an interactive simulator system (the interactive undo-redo simulator) for every reversible reaction system. The construction method is relatively straightforward: First, we include a new symbol ρ in the original background set, and we create the forward reactions by including ρ in every inhibitor set. Then, we assemble the backward reactions by enumerating the reaction sets that can ever become enabled (the elements of $\mathrm{EN}_A(S)$) and follow the construction of Definition 4.3.

The simulators obtained that way may freely perform the forward and the backward computations of the original system, allowing for an Undo-Redo-Do-like semantics of reversibility where the environment controls the direction of the computation.

4.3. Related Work

Considering controlled reversibility in the domain of Reaction Systems, we can refer to the work of Aman and Ciobanu [5]. Similarly to the uncontrolled case of [6], they define reverse reactions by exchanging the reactants and the products of a reaction, and they also extend the model with a memory. They extend each state W_i with a register T_i that keeps track of the vanished symbols. A register T is a set of ordered pairs $T \subseteq S \times N$ where each pair represents a vanished symbol and the number of steps since it disappeared. Then, forward computation proceeds by adding the disappearing entities to the register and incrementing the appropriate counters.

Control over the direction of the computation is implemented via a designated rollback symbol ρ that none of the reactions may produce. The system can only perform reverse reactions if ρ is present in the context. Thus, ρ is a reactant for reverse reactions and an inhibitor for forward ones.

Finally, they prove that systems with the above modifications adhere to the Loop Lemma in the context-independent case ($C_i = \emptyset$ for all $i \geq 1$).

When compared to our contribution, we can note two key differences. First, the model constructed by Aman and Ciobanu uses registers to track vanished items. Such registers implement counting, which is impossible in the base model because of its qualitative nature. On the other hand, the proposed simulator systems do not use any form of memory to perform a reversal. Second, our work allows for non-context-independent interactive processes.

5. Transition Graphs of Backtracking Reaction Systems

As Reversible Reaction Systems, according to the notion of reversibility introduced in Section 4, seem somewhat restricted, our subsequent research question is studying and determining the class of the possible computations they can perform. However, instead of precisely defining their computational capabilities, we take a more intuitive route by constructing and examining the so-called transition graphs of the different models. Given a system, such a graph contains every possible state of every interactive process. Thus, by looking at the properties of these graphs, we can assume the model's intuitive computational power.

In this section, we first give an overview of transition graphs, including prior work and our definition. Since the class of possible computations in the previously defined reversible model is limited, we will study and compare the transition graphs corresponding to different reversible system definitions. After considering the base model, we discuss one inspired by the Initialized Context Restricted Reaction Systems of [41], followed by a so-called lookback model, with the ability to examine the current context (similar to automata).

The results of this section were initially published in [I1].

5.1. Transition Graphs

When introducing reversibility into a particular model of computation, the question naturally arises: How does this affect the model's computational properties? In the case of Reaction Systems, interactive processes are the only means of computation; thus, when examining the higher-level computational properties of a specific system, we should start by looking at the contained processes. Since any system may only contain a finite number of possible states, so-called transition graphs concisely depict every possible interactive process a particular system may enclose. In this section, we introduce the definition of transition graphs and then explore the graphs generated by the reversible systems of Section 3.

Transition graphs were first introduced in [24] as vertices representing the subsets of the background set (usually denoted as S) connected by directed edges equivalent to the relation of "can be obtained from". Formally, the edge set is defined as $E = \{(W_1, W_2) \mid W_1 \subseteq S,\ \mathrm{res}_A(W_1) \subseteq W_2\}$. Here, we wish to underline the subset relationship between the underlying sets of

the connected vertices. This transition graph definition incorporates context sets (environmental input) by considering two states connected if the result of the former can be augmented with context to form the latter.

As we are exclusively interested in subsets of S that appear in interactive processes, we will modify this notion to include only reachable result sets as vertices. Assuming the standard definition of interactive processes (see Section 2), D_0 must be empty; hence, there might be result sets that cannot occur in any interactive process.

Definition 5.1. Let $\mathcal{A} = (S, A)$ be a reversible reaction system with $S = \Sigma_p \cup \Sigma_c$ (where Σ_p and Σ_c are not necessarily disjoint). The result set $D \subseteq \Sigma_p$ is *reachable* if there exists a non-restarting interactive process $\pi = (\gamma, \delta)$ in $\mathcal{A}$ with $\delta = D_0, D_1, \ldots D_n$ such that $D = D_i$ for some $0 \leq i \leq n$. The set of *reachable result sets* in $\mathcal{A}$ is denoted by $\mathrm{REACH}_{\mathcal{A}}$.

If D_0 was allowed to be non-empty (thus, the reaction system can initiate its computation from an arbitrary set), then every result set is reachable (since we can freely choose the initial result set). By requiring D_0 to be empty, we restrict the possible result sets in interactive processes to the reachable sets of Definition 5.1. The set of reachable result sets is, in turn, determined by the reactions of the underlying reaction system.

Compared to the definition of [24], a further change is the labeling of the edges. Since we want to emphasize the role of the input, in our definition of transition graphs, edges are labeled with input sets from the environment. Such a labeled edge is drawn between two vertices if the union of the source vertex and the label produces the destination vertex as a result.

Definition 5.2. Let $\mathcal{A} = (S, A)$ be reaction system, with $S = \Sigma_c \cup \Sigma_p$ as above. The *transition graph* of $\mathcal{A}$ is the graph $\mathrm{TG}_{\mathcal{A}} = (V, E)$, where $V = \mathrm{REACH}_{\mathcal{A}}$ is the set of vertices and

$$E = \{\, (D, C, D') \mid D, D' \in V \text{ and } C \subseteq \Sigma_c \text{ such that } \mathrm{res}_A(D \cup C) = D' \,\}$$

is the set of directed edges with D being the starting vertex, D' the end vertex and C the label.

Example 5.1. Let $\mathcal{A}$ be a reaction system in which $\Sigma_p = \{1, 3, 5\}$ is the product alphabet, $\Sigma_c = \{0, 2, 4\}$ is the context alphabet and $A = \{a, b, c\}$ is the set of reactions, where

$$a = (\{0\}, \{2, 4\}, \{1\}),\ b = (\{1, 2\}, \{0\}, \{3\}),\ c = (\{1, 4\}, \{0\}, \{5\}).$$

Then, $\mathrm{TG}_{\mathcal{A}}$ consists of the vertices $V = \{\ \emptyset, \{1\}, \{3\}, \{5\}, \{3,5\}\ \}$ and edges

$$E = \{\ (\emptyset, \{0\}, \{1\}),\ (\{1\}, \{2\}, \{3\}),\ (\{1\}, \{4\}, \{5\}),\ (\{1\}, \{2,4\}, \{3,5\})\ \}.$$

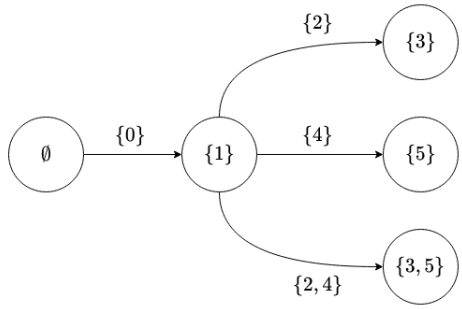

Figure 5.1: Transition graph of the reaction system from Example 5.1.

5.2. Variants

5.2.1. Reversible Reaction Systems

With all the necessary notions in place, we now examine the transition graphs of Reversible Reaction Systems. In such systems, each result set may result from exactly one other state. Consequently, for example, loops are forbidden (explained in more detail and proved in Theorem 5.1), which puts a firm constraint on the complexity of the non-restarting interactive processes. Therefore, the transition graphs of these systems are relatively simple, containing only finite computational branches.

Theorem 5.1. *If $\mathcal{A}$ is a reversible reaction system, then the transition graph of $\mathcal{A}$ is either a single vertex or a directed rooted tree with all the edges pointing away from the root.*

Proof. Let $\mathcal{A} = (S, A)$ be a reversible reaction system with $S = \Sigma_p \cup \Sigma_c$ (where Σ_p and Σ_c are not necessarily disjoint), and let $\mathrm{TG}_{\mathcal{A}}$ be its transition graph.

By definition, $\mathrm{TG}_{\mathcal{A}}$ only includes an edge between two vertices if they are subsequent result sets in some non-restarting interactive process. Thus, the empty result set's vertex has no incoming edges. Since the empty set is

the initial result set (D_0) of every non-restarting interactive process in any reaction system, it will always be included in the transition graph.

If there is no context set C such that $\mathrm{en}_A(\emptyset \cup C) \neq \emptyset$ then $\mathrm{TG}_{\mathcal{A}}$ consists of a single vertex: the empty result set.

Now we show that $\mathrm{TG}_{\mathcal{A}}$ is a directed rooted tree if it has multiple vertices. A graph is a directed rooted tree if there is exactly one path between the root vertex and any other vertex. Since every vertex in the transition graph is a result set in some non-restarting interactive process, there must be a path between the root and vertex representing this set. Thus, at least one path connecting the root vertex with every other vertex must exist.

Since $\mathcal{A}$ is reversible, every state has a unique predecessor. Consequently, every result set has a unique predecessor. As the vertices in the transition graph represent result sets and edges represent predecessor/successor relations, every vertex other than the root has exactly one incoming edge. Therefore, there must be at most one path going from the root vertex to every other vertex. Because the lower and the upper bound are equal to one, we have a single path from the root vertex to any other vertex. Thus, if $\mathrm{TG}_{\mathcal{A}}$ has more than one vertex, it is a directed rooted tree with all edges pointing away from the root. □

Due to the above theorem, non-restarting interactive processes in reversible systems (essentially computations) are just paths in a finite tree. Since this is a rather strict limitation, we might start experimenting with small relaxations in the underlying definitions to give rise to more complex graphs (those with vertices having in-degree greater than one or even containing cycles).

5.2.2. Reversible Initialized Reaction Systems

In the standard setting, D_0 (the initial result set) is empty for every interactive process. If the context sets can incorporate arbitrary entities from the background set, this does not constrain the initial state. On the other hand, in the case when the context and the product alphabets are different (as in the case of reversible systems), the product and the context alphabet can be disjoint, and some results sets may not even be reachable at all. Similar ideas motivated the introduction of so-called initialized context-restricted reaction systems in [41] where non-empty initial product sets D_0 are also allowed. Now, let us examine how non-empty D_0 sets affect the transition

graphs of Reversible Reaction Systems. Following [41], we call our model Initialized Reversible Reaction Systems.

Theorem 5.2. *If $\mathcal{A}$ is an initialized reversible reaction system, then every component of the transition graph of $\mathcal{A}$ is either*

- *a single vertex,*
- *a directed rooted tree with edges pointing away from the root or*
- *one directed cycle, such that each vertex of the cycle can also be the root of a tree with edges pointing away from the cycle.*

Proof. Let $\mathcal{A} = (S, A)$ be an initialized reversible reaction system with $S = \Sigma_p \cup \Sigma_c$ (where Σ_p and Σ_c are not necessarily disjoint) and with interactive processes that might start with a non-empty D_0 set.

Since our definition for the transition graph is the same as in Theorem 5.1, the reversibility of $\mathcal{A}$ results in a maximum of one for the in-degree of every vertex.

Now, let us consider the components of the transition graph. Given a result set $D \subseteq \Sigma_p$, if there is no $W \subseteq S$ such that $\mathrm{res}_A(W) = D$ (in any of the non-restarting interactive processes of $\mathcal{A}$), then the in-degree of the vertex corresponding to D is equal to 0. In this case, this vertex is either a component or the root of a directed rooted tree. The former holds if no result set can be reached from D in any of the non-restarting interactive processes (thus, the out-degree of the vertex is zero), while the latter is proved in the proof of Theorem 5.1.

With the first two cases (single vertex and tree) covered, we examine components with exactly one cycle. We know that vertices with in-degree equal to zero form single-vertex components or act as tree roots. Therefore, we only need to consider components in which the in-degree of every vertex is equal to one (as we previously proved that no vertex has an in-degree greater than one). In this case, the component must include at least one cycle; otherwise, there could be vertices with no incoming edges. However, while a component can include a single cycle when the in-degree of every vertex is one, multiple cycles are impossible. Single-cycle components can take the form of a "branching ring", where the component includes a ring (or cycle) at its core, and each ring member can additionally be the root of a

tree branching out. On the other hand, multiple cycles can only be realized by including at least one vertex in each cycle, with an edge coming from the cycle itself and an edge coming from a vertex outside of the appropriate cycle. As such configurations are forbidden for transition graphs of reversible systems, all remaining components must form a branching ring. □

Example 5.2. Let $\mathcal{A} = (S, A)$ be an initialized reversible reaction system with $S = \Sigma_p \cup \Sigma_c$ where $\Sigma_p = \{1, 3, 5, 7, 9, 11, 13, 15\}$ is the product alphabet, $\Sigma_c = \{0, 2, 4, 6, 8, 10, 12\}$ is the context alphabet and $A = \{a, b, c, d, e, f, g\}$ is the set of reactions, where

$a = (\{0, 1\}, \{6\}, \{3\}), \quad b = (\{2, 3\}, \emptyset, \{5\}), \quad c = (\{4, 5\}, \emptyset, \{1\}),$
$d = (\{1, 6\}, \{0\}, \{7\}), \quad e = (\{7, 8\}, \{10\}, \{9\}), \quad f = (\{7, 10\}, \{8\}, \{11\}),$
$g = (\{12, 13\}, \emptyset, \{15\}).$

The transition graph $\mathrm{TG}_{\mathcal{A}}$ consists of three components: a branching ring, a tree, and a single vertex, as shown in Figure 5.2.

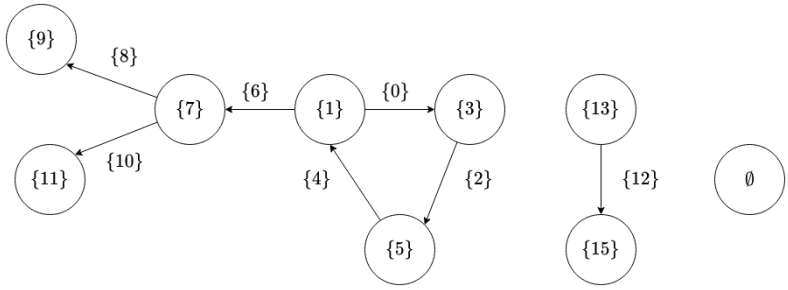

Figure 5.2: Transition graph of the reaction system from Example 5.2.

As stated by Theorem 5.2, with a slight modification in the definition of interactive processes, we can achieve more involved transition graphs: those with multiple components and even a cycle per component. Such constructs allow for computations with loops, increasing the intuitive computational power of the model. Nevertheless, examined solely from an informal perspective, even with this change, we cannot achieve the computational power of reversible finite automata, for example. The lack of higher in-degrees and multiple cycles severely constrain possible non-restarting interactive processes.

5.2.3. Reaction Systems Reversible with Lookbehind

As we have discussed in Section 3, we would like to look at the notion of reversibility as a kind of backward determinism; that is, given any "state" or configuration of the system, we should be able to determine the preceding computational step. The previous two variants have interpreted the concept of configurations (or "states") as the state sets (that is, as the unions of product sets and context sets) of interactive processes. In this section, we follow a different approach, one that is similar to how the reversibility of (finite) automata is usually defined: The "state" of the machine is interpreted as the internal state of the finite control, together with additional information regarding the position of the reading head on the input tape, and the contents of the corresponding tape cell. See [8] and [51] for more details, or [27] for some more recent work regarding reversibility of finite automata. In such an interpretation, the predecessor configurations in computations should be unique with respect to the pairs of internal states and input symbols that the reading head has just left behind, that is, the symbol that was read from the input tape in the previous computational step.

Our notion of reversibility for reaction systems in Section 3 revolves around the unique predecessors of state sets. If a combination of reaction results and context entities (together forming what is called a state) has precisely one way to be produced, then it is said to have a unique predecessor. In what follows, we introduce reversibility with lookbehind by taking a different approach to defining unique predecessors. Inspired by finite state automata, which are considered backward deterministic (and thus, reversible) if the current internal state and the previously consumed input symbol uniquely determine the previous internal state, reaction systems that are reversible with lookbehind can inspect both the current state and the previous context set. Consequently, multiple state sets can produce the same result sets (without losing the reversible property), given that they include distinct context sets.

To reiterate the difference, the reversible reaction systems of Section 3 consider the state set as a single set. In contrast, the newly introduced lookbehind systems can individually examine the result and context sets.

Definition 5.3. Let $\mathcal{A} = (S, A)$ be a reaction system and $\pi = (\gamma, \delta)$ be an interactive process in $\mathcal{A}$, such that $\gamma = C_0, C_1, \ldots C_n$, $\delta = D_0, D_1, \ldots D_n$ and $\mathrm{sts}(\pi) = W_0, W_1, \ldots W_n$.

A state W_i, $1 \leq i \leq n$, has *multiple predecessors with lookbehind* if there exist $D \subseteq S$ such that $D \neq D_{i-1}$, but $\mathrm{res}_A(D \cup C_{i-1}) = D_i$. If there is no

such D, then W_i has a *unique predecessor with lookbehind*.

The interactive process π is *reversible with lookbehind* if every state W_i, $1 \leq i \leq n$, has a unique predecessor with lookbehind.

Now, we can define reversible systems using the above definition.

Definition 5.4. A reaction system $\mathcal{A}$ is *reversible with lookbehind* if every non-restarting interactive process in $\mathcal{A}$ is reversible with lookbehind.

Regarding transition graphs, an immediate consequence of the new definition of reversibility is the possibility of in-degrees higher than one. As shown in Figure 4, despite the two incoming edges of the vertex $\{4\}$, when reversing the previous computation, we can now decide which one to take based on the preceding context (the label of the edge).

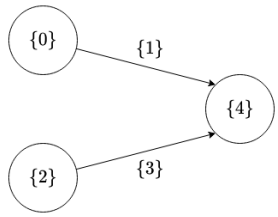

Figure 5.3: Transition graph configuration that is not permitted for ordinary reversible systems but is allowed for systems reversible with lookbehind.

Continuing our previous discussion, we now compare the state diagrams and the transition graphs of reversible finite transition systems and reversible reaction systems with lookbehind, respectively.

A finite transition system is usually denoted as a triplet $F = (Q, \Sigma, \delta)$, where Q is a finite set of states, Σ is the finite input alphabet, and $\delta : Q \times \Sigma \to Q$ is the state transition function, mapping the current state and an input symbol to a resulting state.

Computation in this model takes place in the form of transitions governed by the δ transition function. Given two states, $q_0, q_1 \in Q$ and a symbol $s \in \Sigma$, the system can transition from q_a into q_b reading s if $\delta(q_a, s) = q_b$. We can use the notation $q_a \xrightarrow{s} q_b$ to capture such transitions. Then, $q_0 \xrightarrow{s_0} q_1 \xrightarrow{s_1} \cdots \xrightarrow{s_{n-1}} q_n$ is a transition sequence over the string $s_0 \ldots s_n$ if $\delta(q_i, s_i) = q_{i+1}$ $(0 \leq i \leq n-1)$ for all $q_0, \ldots, q_n \in Q$.

Finite transition systems differ from finite state automata by not having distinguished start and final states. Although the following results can also

be stated for finite state automata, reaction systems are more closely related to transition systems since interactive processes in reaction systems lack the concept of a final (or accepting) state. This similarity is further supported by [14], where a method is presented to convert finite transition systems to reaction systems.

In what follows, when constructing transition graphs, we assume the definition of interactive processes in which D_0 (the initial result set) is empty. Hence, every interactive process must start from the empty result set, making it easier to create reaction systems from transition systems, as it allows for more control over the reachable result sets. Example 5.3 explores this idea in greater detail.

Example 5.3. Let F be a reversible finite transition system for which we wish to construct a corresponding reaction system. Starting with the background set, we can create an entity for each input symbol of F and each state of F. The entities created from the input symbols comprise the input alphabet Σ_c, while the entities representing the states belong to the product alphabet Σ_p. Now, we have that the result sets of the reaction system (D_i) represent the actual state of the underlying transition system, and the context sets correspond to the received input.

If the initial result set D_0 was allowed to be non-empty, then any entity of the product alphabet could be present in this set, even multiple entities. Since each entity represents a distinct state of the underlying transition system, multiple entities would mean the transition system is in multiple states simultaneously. As this is not permitted, one should require D_0 to be empty since that way, the facilitation and inhibition aspects of the reaction can prevent such cases.

Theorem 5.3. *For every reversible finite transition system, there is a reaction system that is reversible with lookbehind and has the same states and transitions (apart from a starting state and its corresponding transitions).*

Proof. Let $F = (Q, \Sigma, \delta)$ be a reversible finite transition system. Then, the state diagram of F, $\mathrm{SD}_F = (V_F, E_F)$ is a directed graph, where $V_F = Q$ is the set of vertices and $E_F = \{(v, l, w) \mid \delta(v, l) = w\}$ is the set of directed, labeled edges.

Now, let us construct a reaction system $\mathcal{A} = (S, A)$ from F. Initially, we choose the background set S to be the union of two disjoint sets ($S =$

$\Sigma_p \cup \Sigma_c$): the product alphabet corresponds to the states of the transition system (thus, $\Sigma_p = Q$), while the input alphabet is equivalent to the input alphabet of F (thus, $\Sigma_c = \Sigma$). We can define a set of reactions using the transition function of F:

$$\{(\{q, i\}, \Sigma_c \setminus \{i\}, \{r\}) \mid \delta(q, i) = r, \text{ for } q, r \in Q,\ i \in \Sigma\}.$$

Finite transition systems do not have a designated initial state but may begin their computation in an arbitrary state. In the case of reaction systems, however, an interactive process must start with the empty result set ($D_0 = \emptyset$). Since the context and the product alphabets are, in this case, disjoint ($\Sigma_p \cap \Sigma_c = \emptyset$), it is not possible to put a symbol representing some state $q \in Q$ in the initial context set (C_0). To overcome this issue and allow the reaction system to start its computation by jumping to an arbitrary state q, we introduce a new entity, α_q, for each state $q \in Q$ and an appropriate reaction that leads from the empty result set to the result set representing this state q of F.

With this in mind, let us redefine the background and the reaction set of $\mathcal{A}$. The background set S is now the union of the following two sets: $\Sigma_p = Q$ and $\Sigma_c = \Sigma \cup \{\alpha_q \mid q \in Q\}$. The reaction set A is then defined as follows:

$$\begin{aligned} A = &\{(\{q, i\}, \Sigma_c \setminus \{i\}, \{r\}) \mid \delta(q, i) = r \text{ for } q, r \in Q,\ i \in \Sigma\} \cup \\ &\{(\{\alpha_q\}, \Sigma_p \cup (\Sigma_c \setminus \{\alpha_q\}), \{q\}) \mid q \in Q\}. \end{aligned}$$

The transition graph of $\mathcal{A}$ is defined as a directed graph based on the result sets and inputs of the non-restarting interactive processes in the system. Because of the definition of the transition graph, given any vertex q in the state diagram of F, we have a vertex in $\mathrm{TG}_{\mathcal{A}}$ corresponding to $D = \{q\}$. Additionally, for every edge (v, l, w) in the state diagram of F, we have an appropriate edge pointing from the vertex $D_1 = \{v\}$ to the vertex $D_2 = \mathrm{res}_{\mathcal{A}}(\{v, l\}) = \{w\}$ (because of the definition of A).

Consequently, apart from the vertex representing the empty state and its outgoing edges, the transition graph of $\mathcal{A}$ and the state diagram of F are isomorphic.

What is left to prove is that the states in the non-restarting interactive processes of $\mathcal{A}$ have unique predecessors with lookbehind (making $\mathcal{A}$ reversible with lookbehind). Since F is a reversible transition system, no

vertex in its state diagram has more than one incoming edge with the same label. Consequently, each vertex in the transition graph of $\mathcal{A}$ satisfies the same property. Combining this fact with the disjointness of the product and input alphabets, we have that no state can be reached with the same input (or label, in the transition graph) from two different result sets. That is precisely the definition of having a unique predecessor with lookbehind. □

Theorem 5.4. *For every reaction system which is reversible with lookbehind, there is a reversible finite transition system with the same states and transitions.*

Proof. Let $\mathcal{A} = (S, A)$ be a reversible lookbehind reaction system with $S = \Sigma_p \cup \Sigma_c$ (where Σ_p and Σ_c are not necessarily disjoint). Also, let $\mathrm{TG}_\mathcal{A} = (V_\mathcal{A}, E_\mathcal{A})$ be the transition graph of $\mathcal{A}$.

Now, let us construct a finite transition system $F = (Q, \Sigma, \delta)$ from $\mathcal{A}$. Using the transition graph of $\mathcal{A}$, we can define the states of the transition system as $Q = V_\mathcal{A}$. The input alphabet of F is going to contain the subsets of the input alphabet of $\mathcal{A}$, thus $\Sigma = 2^{\Sigma_c}$. By considering the edges in $\mathrm{TG}_\mathcal{A}$, we can define the transition function as

$$\delta(q, i) = r, \text{ if there is an edge in } E_\mathcal{A} \text{ from } q \text{ to } r \text{ with label } i.$$

Because of the above definition of F, given any vertex (representing a result set) in the transition graph of $\mathcal{A}$, we will have a corresponding vertex in the state diagram of F. Furthermore, since the edges in the state diagram correspond to the transition function δ, which in turn was defined via the edges of $\mathrm{TG}_\mathcal{A}$, we have that each edge in the state diagram of F will map to an edge in $\mathrm{TG}_\mathcal{A}$. Thus, the state diagram of F and the transition graph of $\mathcal{A}$ are isomorphic.

Analogous to the proof of Theorem 5.3, we need to show that F is a reversible transition system. Since $\mathcal{A}$ is reversible, no vertex in its transition graph has more than one incoming edge with the same label. As the state diagram of F is isomorphic to $\mathrm{TG}_\mathcal{A}$ and each edge has the same label as its counterpart in $\mathrm{TG}_\mathcal{A}$, the same is true for each vertex in the state diagram. Thus, because the edges represent the state transitions induced by δ, we have that F is reversible. □

Based on the previous two theorems, we can state the following.

Proposition 5.1. *The state transition graphs of reversible finite transition systems correspond to the transition graphs of reaction systems, which are reversible with lookbehind, apart from the particular initial vertex corresponding to the initial empty result set of the reaction system. Vice versa, the transition graph of any reaction system that is reversible with lookbehind corresponds to the state transition graph of a reversible finite transition system.*

5.3. Related Work

Holzer and Rauch also explored uncontrolled reversibility in the context of Reaction Systems while also considering appropriate graph representations. In [28], rather than making systems reversible, they investigate the computational properties of reversible systems. This investigation is based on reversibility as a bijection: a reaction system $\mathcal{A} = (S, A)$ is bijective if the transition function on the subsets of S induced by $\mathcal{A}$ is bijective. They use state graphs to prove this relation between bijective systems and reversibility.

While this definition of reversibility aligns with the one in this dissertation, it is vital to be aware of a few differences. In our work, we only consider non-restarting interactive processes. Thus, we explicitly forbid backward transitions in the case of one particular set: the empty result set. Moreover, this section explored the intuitive computational power of reversible reaction systems using transition graphs instead of the state graphs of Holzer and Rauch. The crucial difference between these two graphs is their definition of nodes: in the case of transition graphs, nodes correspond to result sets, while state graphs equate nodes with state sets.

6. Backtracking in Communicating Reaction Systems

The main goal of our research was to explore different paradigms of reversibility within the same computational model. In the previous section, we studied backtracking, a paradigm applicable when the concept of the last computational step exists. As the original definition of Reaction Systems describes a model that operates in a serial, step-by-step manner, the backtracking approach to reversibility is perfectly suitable. However, because of their serial nature, Reaction Systems cannot accommodate paradigms that involve asynchronicity and concurrency. Therefore, we needed a sufficient modification to the base model to continue our explorations.

In this section, we consider Communicating Reaction Systems by Direct Communication introduced by [19] that may contain an arbitrary number of interconnected components. We explore how backtracking works in the context of this model to show that such a networked or multi-entity model needs more sophisticated notions of reversibility than those suited for serial ones.

The results of this section were initially published in [J1].

6.1. Related Work

Several researchers studied forming computational models from multiple interconnected cooperating reaction systems. This section gives a brief overview of these approaches.

Męski, Koutny, and Penczek first explored the idea of organizing multiple reaction systems into a single computational device, calling the resulting model Distributed Reaction Systems (DRS, for short) [40]. This model presents three fundamental extensions over the original definition. First, Distributed Reaction Systems allow compartmentalization (like membrane or tissue systems) by containing an arbitrary number of ordinary reaction systems called agents. Each agent has its own reaction set defined over a shared background set. Then, such agents may perform operations synchronously or asynchronously, providing the ability to model concurrent execution and related synchronization schemes. Finally, the context sets of the individual agents are generated by a context automaton, allowing conditional input generation based on the current state.

The flow of computation in this model is as follows. In the case of ordi-

nary reaction systems, the environmental input for each interactive process step is a set of context entities. Distributed Reaction Systems add another component to this input: the set of activated agents. Only these agents perform reactions, while the result set of the others remains the same. Thus, by allowing every agent to be active in every step, one can model synchronous execution. Conversely, one can achieve asynchronous semantics by activating a different subset of agents in the subsequent steps. To compute the following result set, an active agent takes its input from the environment and combines it with the current result set of every active agent (including itself). Then, the agent applies its enabled reactions, and the process continues.

As we can see from the above description, the agents perform reactions over a shared result set. This shared pool of entities allows for communication and cooperation between the agents. Many agents may use the same entity thanks to the threshold assumption of reaction systems. However, the above definition also implements the idea of compartments as the set of reactions and the input from the environment varies by agent.

A notable research direction concerning Distributed Reaction Systems is exploring them as language-generating devices. Ciencialová, Cienciala, and Csuhaj-Varjú showed that the languages of Distributed Reaction Systems correspond to right-linear matrix languages and constructed a representation of the recursively enumerable languages by such systems [16].

Bottoni, Labella, and Rozenberg explored a different direction by organizing multiple reaction systems into a graph and calling the resulting computational devices Networks of Reaction Systems [12]. Such a network is a graph in which each node is a reaction system that contributes symbols to the context sets of its neighbors. The individual systems may have different background sets and reaction sets. Then, the flow of a so-called network interactive process is as follows. Each reaction system applies its enabled reactions according to a global clock (thus, the steps are always synchronized, unlike in Distributed Reaction Systems). The products of the reactions form the subsequent result set of the system and get transported to the adjacent systems. Since the background sets can differ, each receiving system filters the incoming symbols before accepting them into the following context set. Finally, the network interactive process continues as expected: each system applies reactions to its current state set.

Building on the above model, Bottoni, Labella, and Rozenberg focused their research on the behavior of a single graph node called the central reaction system as the topology of the graph changes. They treated the other

nodes as merely producing the input to this central system while investigating the state sequence of the central node.

Another implementation of the base idea is the model of Communicating Reaction Systems with Direct Communication (cdcR systems, for short) by Csuhaj-Varjú and Sethy [19]. In the definition of Bottoni, Labella, and Rozenberg, the individual systems transport their results to themselves and each of their neighbors. Conversely, cdcR systems introduce the notion of targeted products: each reaction product is assigned a label indicating the target system. Thus, the resulting product is transmitted solely to the target when such a reaction is successfully applied. Notably, the target system may be the same as the sender; that is how each system's subsequent result set is formed. As a result, the topology is encoded within the reactions instead of a predefined graph. The computational flow is the same as in the case of networks of reaction systems, meaning interactive processes apply reactions according to a global clock. In the following section, we will cover this model in more detail.

Csuhaj-Varjú and Sethy also studied a variation communicating by reactions instead of products. When an enabled reaction is successfully applied, it is not the products but the reaction itself that gets transmitted to the appropriate target nodes. In some sense, this model results in a dynamically evolving system, treating the rules (or "code") as data.

We can relate cdcR systems to other reaction systems variants. For both flavors of cdcR systems, Csuhaj-Varjú and Sethy proved that the model can be translated into a single reaction system. Furthermore, Aman also established translations between networks of reaction systems and cdcR systems [4].

6.2. cdcR Systems Communicating by Products

Now, we recall the most important notions and definitions concerning cdcR systems communicating by products. For more details, see [19] and [18].

Remark 6.1. The variant we consider in the following sections is called cdcR systems communicating by products, 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.

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 cdcR system communicating by products 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 $\{1, 2, \ldots, n\}$.

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

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 $W_{j+1} = \mathrm{succ}(W_j)$ for all $0 \leq j \leq m-1$. This can also be written as $((W^1)_m, \ldots, (W^n)_m)$ where each $(W^i)_m$ is a finite sequence $(W^i)_m = W_0^i, W_1^i, \ldots, W_m^i$, $1 \leq i \leq n$, and $(W_j^1, \ldots, W_j^n) = W_j$, $0 \leq j \leq m$, $(W_0^1, \ldots, W_0^n) = W_0$ 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, $C_j^i = \bigcup_{1 \leq k \leq n, k \neq i} \{p \mid (p, i) \in \mathrm{res}_{A_k}(W_{j-1}^k)\}$ for $j \geq 1$, 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, $D_j^i = \{p \mid (p, i) \in \mathrm{res}_{A_i}(W_{j-1}^i)\}$ for $j \geq 1$.

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

$W_0^i = C_0^i$ and $D_0^i = \emptyset$.

6.3. Backtracking Through Local Reversibility

In this section, we study the possibility of backtracking reversibility in cdcR systems. As discussed, this model organizes multiple independent reaction systems into a graph. Such a structure would make cdcR systems an ideal candidate to model asynchronous and concurrent computation. However, the presence of the global clock makes the individual components, thus, the entire computation synchronous.

Therefore, we first explore backtracking, a paradigm suited for serial or synchronous models. The idea is to reuse our definition of reversibility from the previous sections and create an entire reversible communicating system.

Definition 6.1. Let $\Delta = (S, \mathcal{A}_1, \ldots, \mathcal{A}_n)$ be a cdcR system, and let $W = (W^1, \ldots, W^n)$, $W^i \subseteq S,\ 1 \leq i \leq n$, 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 $\text{sts}(\pi^i) = (W^i)_m = W_0^i, W_1^i, \ldots, W_m^i$, $1 \leq i \leq n$. The interactive *process* Π *is reversible*, if every global state $W_j = (W_j^1, \ldots, W_j^n)$, $1 \leq j \leq m$, has a unique predecessor, and an interactive process Π is *non-restarting*, if $W_j^i \neq \emptyset$ for all $1 \leq i \leq n$, $1 \leq j \leq m$.

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 of non-restarting interactive communicating processes. We did this for unity with the corresponding definition for individual reaction systems: Reversing a restarting process in an individual reaction system would mean producing "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 system's reversibility and its components' reversibility. However, 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 [41] to describe a case when only entities from a subset of the background set are allowed to appear in the context sets (except for 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 Section 5, 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. Due to 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 considering time or step counting.

Definition 6.2. 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, $1 \leq i \leq n$, with a set of extended reactions $A_i \subseteq \mathrm{erac}(S)$.

Consider the reaction system $\mathcal{A}'_i = (S, A'_i)$ where the background set S contains the subsets $\Sigma^i_c, \Sigma^i_{localp} \subseteq S$ with $\Sigma^i_c = \bigcup_{1 \leq j \leq n, j \neq i} \{p \mid (p, i) \in P_a \text{ for some } a \in A_j\}$, $\Sigma^i_{localp} = \{p \mid (p, i) \in P_a \text{ for some } a \in A_i\}$, and the set of (non-extended) reactions is $A'_i \subseteq \mathrm{rac}(S)$ 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 of its components are reversible.

Local reversibility (the reversibility of the components) implies the reversibility of the cdcR system, as seen in the following proposition.

Proposition 6.1. *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 6.2) 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, $W_k^i \neq \emptyset$ 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_k^i, $1 \leq k \leq j$, in the state sequences $\mathrm{sts}(\pi^i)$, $1 \leq i \leq n$, have unique predecessors. This means that the state $(W_j^1, \ldots, W_j^n)$ must also have a unique predecessor, and since $(W_j^1, \ldots, W_j^n)$ 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. $\square$

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

Proposition 6.2. *There are reversible cdcR systems that 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

$$\begin{aligned} A_1 &= \{a_1^1 = (\{a\}, \{b, c, d, e\}, \{(b, 1)\}), \\ &\quad\ a_2^1 = (\{b\}, \{a, c, d, e\}, \{(d, 1)\}), \\ &\quad\ a_3^1 = (\{b, e\}, \{a, c, d\}, \{(c, 1)\})\}, \\ A_2 &= \{a_1^2 = (\{a\}, \{b, c, d, e\}, \{(b, 2), (e, 1)\}), \\ &\quad\ a_2^2 = (\{d\}, \{a, b, c, e\}, \{(b, 2)\}), \\ &\quad\ a_3^2 = (\{b\}, \{a, c, d, e\}, \{(c, 2)\})\}. \end{aligned}$$

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_0^1, W_0^2) can be such that $W_0^1 \in \{\{a\}, \{b\}, \{b, e\}\}$ and $W_0^2 \in \{\{a\}, \{b\}, \{d\}\}$ which gives us nine combinations with nine possible state sequences. The reader should check that all global states in these sequences have unique predecessors; we look at the case when $W_0^1 = W_0^2 = \{a\}$ 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_3^1 and a_3^2, and the

inhibitors only allow these to be applied to the state $(\{b, e\}, \{b\})$, which is W_1^i, 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_1^2, 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 Defintion 6.2.

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

$$\begin{aligned} A_2' \;=\; & \{a_1^{2\prime} : (\{a\}, \{b, c, d, e\}, \{b\}),\ a_2^{2\prime} : (\{d\}, \{a, b, c, e\}, \{b\}), \\ & a_3^{2\prime} : (\{b\}, \{a, c, d, e\}, \{c\})\}. \end{aligned}$$

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 $a_1^{2\prime}$ or from a state $\{d\}$ by the reaction $a_2^{2\prime}$. 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 why 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 predecessor, 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.)

6.4. Limitations of Synchronization

Summarizing the results of the previous section, we can build reversible cdcR systems by ensuring each global state has a unique predecessor. The most straightforward approach is to construct the system only from locally reversible components; however, this is not a requirement.

This result is a consequence of the property shown by Csuhaj-Varjú and Sethy in [19]: a cdcR system is essentially a compartmentalized version of a single reaction system with a simple encoding between the two. Thus, the global state of a cdcR system translates to a state in an ordinary reaction

system. Therefore, by showing that the global state has a unique predecessor, we guarantee the same for the translated state, resulting in backward determinism.

As mentioned in the introduction of this chapter, backtracking is a paradigm best suited for computational models that perform their steps serially. Usually, these models contain a single entity (or agent), as in the case of ordinary Reaction Systems. Nevertheless, we implemented backtracking in a model that is a virtual graph of independent components. However, as suggested by the translation between cdcR systems and reaction systems, this independence of components is superficial. As the steps of the interactive processes are synchronized, each component depends on the tick of a global clock. Since the computation may only proceed when the clock ticks, the entire graph behaves like a single reaction system.

Backtracking is the ideal (and only) paradigm of reversibility for cdcR systems because of the synchronization between the components. Even if we wanted to implement a more advanced paradigm, for example, causal consistent reversibility, it would reduce to simple backtracking as computation in cdcR systems is strictly synchronous. The individual components cannot perform concurrent actions, leading to a well-ordered series of steps one can readily traverse backward. Thus, to experiment with a more complex paradigm of reversibility, we need a model that is not limited by synchronous execution semantics but allows for concurrent execution.

7. Causal-Consistency in Communicating Reaction Systems

A common trait of ordinary Reaction Systems and cdcR systems is using a global clock, which drives the computational steps of interactive processes. 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.

However, 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 [20].

This section focuses on causal consistency as a paradigm better fitting distributed or networked models, such as cdcR systems. Since we can think of each component as a computational unit of its own, we should give them the ability to change the direction of their computation independently. This idea also resonates with the concept of distributed systems: even if some part of the whole reaches an error state needing recovery, other parts of the system might continue their computation unbothered. Thus, our goal in this section is to construct communicating reaction systems in which the components can compute backward and forward independently, synchronizing only when necessary while adhering to the requirements of causal-consistent reversibility.

The section is organized as follows. We first give an overview of this paradigm and its applications. Then, we consider our modification of cdcR systems, called Distributed cdcR systems (d-cdcR systems, for short), that adds concurrency to the model. Then, we implement causal-consistent reversibility for d-cdcR Systems.

The results of this section were initially published in [J1].

7.1. Reversibility Paradigms: Causal-Consistency

As highlighted in Section 3.1 and Section 6.4, different computational semantics require different approaches to reversibility.

For models with sequential execution, reversibility is equal to walking backward (or "backtracking") in the sequence of performed steps. In such a setting, the computations always follow a well-ordered, linear flow. At any given point, the end of this flow, the "last performed action," is unambiguous. Conversely, if, during the forward steps, we record appropriate history information, such that we know the step before the last performed action, we can recursively work our way back to the beginning of the computational sequence.

Thus, the reversibility paradigm for sequential execution, backtracking, relies on the last performed action. However, this concept is not practical in the case of concurrent and distributed computational models. In such a setting, many actions might happen without an exact ordering between them, making it impossible to define which action was last. Therefore, we can only establish the "last performed action" notion if we enforce a "global" ordering on the computation steps, eliminating the concurrent behavior. As a result, one may have concurrency or backtracking, but never both.

Backtracking, on the other hand, is only one particular paradigm of reversibility. Hence, if one cannot implement concurrency and backtracking, is it conceivable to pair concurrency with a different approach to reversibility?

A suitable definition is causal consistent reversibility, introduced by Danos and Krivine [20]. A single "global" ordering of computational steps in concurrent systems is impossible; nonetheless, some ordering is still imperative to decide which step to undo next. Causal consistent reversibility establishes this ordering based on the causal dependencies between the actions, relating reversibility and causality. Essentially, it says that we can only reverse a computational step once we reversed all of its consequences. Thus, the concept of consequence (as opposed to backtracking's concept of the "last performed action") is fundamental to constructing models according to this paradigm.

To understand the causal relationship between actions and the notion of consequence, let us assume we have two independent agents, A_1 and A_2. First, we consider the case when A_1 sends a message to A_2. In this case, the receipt of the message and any subsequent action performed by A_2 is a consequence of the message sent by A_1. Then, if we want to reverse the message-sending action of A_1, we can only do so if the consequent actions

performed by A_2 are undone first. Conversely, if A_1 and A_2 do not exchange messages, the lack of interaction allows us to reverse their computational steps in any order.

The above definition of causal consistency, the ability to reverse actions only when they have no consequences left, is succinct yet informal. Therefore, the question naturally arises about checking if a model implements causal consistency. One such tool is the axiomatic framework of Lanese, Phillips, and Ulidowski, providing a unified, model-independent set of properties and theorems. Given a computational model that adheres to some appropriate lower-level axioms, the property of causal consistency automatically follows. As we built our reversible cdcR system using this framework, we will introduce it in more detail in Section 7.3.

Since Danos and Krivine introduced the notion of causal consistent reversibility in their seminal paper, it has become the standard for defining reversibility in the concurrent setting. While it was initially defined on top of Milner's Calculus of Communicating Systems (CCS) [42], a type of process calculus, since then this notion has seen wide use in quite different models as well, such as Petri Nets [39], the Erlang programming language [36] or term rewriting systems [45]. A comprehensive survey on such models and the general idea of causal consistency can be found in [35].

7.2. Distributed Communicating Reaction Systems

Since causal consistency connects reversibility and concurrency, we first consider a modification of cdcR systems, called Distributed cdcR Systems, that adds concurrency to the model.

First, we remove the synchronization between component interactive processes by allowing external control to decide which components should perform computation. Here, "computation" 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 applies when performing computation. Hence, idle components, in turn, can be considered 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, we say it is blocked and cannot continue 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 blocking to represent the inability to 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 [41]. However, their solution for handling communication between the components differs significantly from how cdcR systems work.

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

Definition 7.1. 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 7.2. 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_j^i in $(\gamma^i)_m$, $1 \leq j \leq m$, are as follows:

$$ComTo_j^i = \bigcup_{\substack{1 \leq k \leq n,\ k \neq i \\ k \in Active_{j-1}}} \{p \mid (p,i) \in \mathrm{res}_{A_k}(W_{j-1}^k)\} \cup C_j^{env,i},$$

$$C_j^i = \begin{cases} ComTo_j^i, \text{ if } i \in Active_{j-1}, \\ ComTo_j^i \cup C_{j-1}^i, \text{ otherwise,} \end{cases}$$

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

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

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

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. 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 $ComTo_j^i$ and $C_j^{env,i}$, respectively) and retain their result sets.

In the following, we will consider cdcR systems with distributed interactive processes and 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 7.1. 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

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

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

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

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_1^3 from A_3, so we might obtain

$$(C_1^1, D_1^1), (C_1^2, D_1^2)(C_1^3, D_1^3)) = ((\{a, b, c\}, \emptyset), (\{a, b\}, \{c\})(\{c, d\}, \{d\})).$$

$\mathcal{A}_1$ is blocked, so $D_1^1 = D_0^1$, and $C_1^1 = C_0^1 \cup \{c\}$ because the product c is received from $\mathcal{A}_2$ due to the application of $a_1^2 \in A_2$. The product set of the second component is $D_1^2 = \{c\}$ (because of the application of $a_1^2 \in A_2$), and $C_1^2 = \{a, b\}$ where b is obtained from $\mathcal{A}_3$ due to the reaction a_1^3, while a is added as input from the environment. The set of products at the third component is $D_1^3 = \{d\}$ because $a_1^3 \in A_3$ was applied, and $C_1^3 = \{c, d\}$ since d is received from the second component and c is added from the environment.

The states are now

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

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_1^2 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 7.2, $Step_0 = \{2, 3\}$, $Block_0 = \{1\}$, so $Active_0 = \{2, 3\}$. Further, $Block_1 = \{2\}$ and $Step_1 = Active_1 = \{1\}$.

7.3. Reversible Distributed Communicating Reaction Systems

By enabling concurrent computation in cdcR systems, we can now focus on our approach to reversibility. As discussed, 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 that has 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.

As noted above, causal-consistent reversibility has been considered for many computational models. However, these models use their terms to define causal consistency and related properties, making 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, Phillips, and Ulidowski introduced a common framework based on labeled transition systems [38]. 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.

In this section, we construct reversible d-cdcR systems based on the axiomatic framework of Lanese, Phillips, and Ulidowski. Since the axioms and properties are defined for labeled transition systems (LTSs), our approach will be to construct LTSs corresponding to individual d-cdcR systems and prove that they adhere to the requirements. Instead of separating the framework and our implementation, we will introduce the appropriate concepts and axioms as they appear. We refer the reader to [38] for more details.

Implementing causal-consistent reversibility requires the model to record history and causality information. To store the latter, we will extensively use stacks tracking 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 $K = pop(e \bullet K)$.

The first step is to introduce labeled transition systems that are nearly the same as the finite transition systems of Section 5, with terminology that is more appropriate to our current needs. One can define such a system by specifying its processes ($Proc$), labels (Lab), and transitions (Tr).

Definition 7.3. 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, labelled with a, is denoted as $t : P \xrightarrow{a} Q$.

In our first attempt to define an LTS corresponding to d-cdcR systems, let us start with the processes of the LTS. Since computation in d-cdcR systems occurs in distributed interactive processes, it is straightforward to turn the global states into LTS processes. 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 must 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 for 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 $D = \text{res}(W^i)$ 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 require each state to have a uniquely determined predecessor. As stated previously, 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 must 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 communication flow between the system components. Thus, in what follows, we modify our definitions accordingly.

Definition 7.4. An interactive *process with delays* $\bar{\pi}$ with $\text{sts}(\bar{\pi}) = W_0, W_1, \ldots W_m$ 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 6.2, if we also add to $\Sigma_c^i,\ 1 \leq i \leq n$, the entities that we would like to be able to input to the context sets of component $\mathcal{A}_i$ from the environment.

Definition 7.5. 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 $A_i \subseteq \text{erac}(S)$.

By requiring that each component of a reversible d-cdcR systems is locally reversible (as in the case of the backtracking systems of Section 6), we ensure a uniquely determined predecessor for each result set that might occur. What remains is recording the communication flow between the components.

Definition 7.6. 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 7.2.

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 7.2.

For each component $\mathcal{A}_i$, $(send^i)_m$ is a series of stack families in which we define each stack 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 track the system components' communication. 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, 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 has 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 similarly 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 7.2. Consider a variant of the system in Example 7.1, 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

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

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

$$\begin{aligned}
send_0^1 &= (send_0^{1,2}, send_0^{1,3}), \\
send_0^2 &= (send_0^{2,1}, send_0^{2,3}), \\
send_0^3 &= (send_0^{3,1}, send_0^{3,2}),
\end{aligned}$$

and

$$\begin{aligned}
recv_0^1 &= (recv_0^{2,1}, recv_0^{3,1}, recv_0^{env,1}), \\
recv_0^2 &= (recv_0^{1,2}, recv_0^{3,2}, recv_0^{env,2}), \\
recv_0^3 &= (recv_0^{1,3}, recv_0^{2,3}, recv_0^{env,3}),
\end{aligned}$$

with 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

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

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 are now

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

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

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

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 7.7. 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 7.8. 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 7.9. 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 the 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 7.2.

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

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

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

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 7.6. 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 7.6. 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 7.3. Consider the system $\Delta' = (S, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ from Example 7.2, 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 7.2, then $P \xrightarrow{a} Q$ according to Definition 7.9 above.

The connection between reversibility, causality, and concurrency is central to causal consistency. 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 [17].

Definition 7.10. 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 7.11. 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 7.1. If t is in conflict with u, then u is in conflict with t.

Definition 7.12. Let t and u be two consecutive transitions. If t does not cause u, 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 7.2. 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 each other's computational steps by communicating targeted products. These symbols will appear in the context set of the recipient, potentially impacting the set of enabled reactions in the proceeding step—consequently, 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. In other words, we aimed to allow computation in interactive processes to flow backward as easily as forward. To this end, the *send-recv* stacks of Definition 7.6 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 7.13. 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 7.14. 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 $i \notin \underline{a}$:

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

and the reversed state is computed as

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

$$D_P^i = D_Q^i \text{ and } C_P^i = \text{EffectiveContext}_P^i.$$

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

$$\begin{aligned} send_P^{i,k} &= \begin{cases} pop(send_Q^{i,k}) \text{ if } top(send_Q^{i,k}) = D_Q^i, \\ send_Q^{i,k}, \text{ otherwise,} \end{cases} \\ W_P^i &= \text{UniquePredecessor}_{A_i}(D_Q^i), \end{aligned}$$

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

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, condition (1) forbids the reversal of initial states by requiring a non-empty result set for every reversed component. Then, condition (2) states that we cannot undo a state where 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 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.

The backward transition is relatively straightforward once the above conditions are satisfied. 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 7.15. With the guidance of our end goal, the Square Property (Definition 3.1, [38]), 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 $1 \leq k \leq n$ such that $top(send_P^{i,k}) = D_P^i$, $top(recv_P^{i,k}) = (D_P^k, C)$, $k \notin \underline{b}$, and $k \in active_a$.

   (c) There is $(env, p, k) \in comm_a$ such that $k \in \underline{b}$.

   (d) There is $1 \leq i \leq n$ such that $env_i \in \underline{b}$ and $i \in active_a$.

Let us elaborate further on the above cases. Item (1), the case of coinitial backward transitions is a consequence of the validity criteria of Definition 7.14 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, the backward label becomes invalid if we first compute using the forward label a. 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 removing 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 we still need to add, however, is a connection between these two. The question naturally arises

whether it is possible to reverse every transition, and if so, how. First, we define the inverse of a forward transition.

Definition 7.16. 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 $a = \mathit{active}; \mathit{comm}$.

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

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

The above construction is relatively 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. If component $\mathcal{A}_i$ receives symbols from the environment, a corresponding env_i is placed in the label.

Lemma 7.1. *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 $a = \mathit{active}; \mathit{comm}$. Then, let us construct the inverse transition, $\underline{t} : Q \xrightarrow{\underline{a}} S$ according to Definition 7.16. 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 7.14.

(1) For each $i \in \mathit{active}$, we have that $W_Q^i = \mathrm{res}(W_P^i) \cup \{p \mid (k, p, i) \in \mathit{comm}, 1 \leq k \leq n\}$. Since $\underline{a} \cap \{1, \ldots, n\} = \mathit{active}$, this means that $\mathrm{res}(W_P^i)$ is non-empty for each $i \in \underline{a}$. Thus, D_Q^i 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 $\mathit{top}(\mathit{recv}_Q^{k,i}) = (D_Q^i, C)$ for some C. This is only possible if $k \in \mathit{active}$. Now, since $\underline{a} \cap \{1, \ldots, n\} = \mathit{active}$ we know that $k \in \underline{a}$ as well. Furthermore, because k was the sender of the received communication, $\mathit{top}(\mathit{send}_Q^{k,i}) = D_Q^k$ holds.

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

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

(5) For each $i \in \mathit{active}$, 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_Q^i to receive multiple communications from the same sender. Consequently, as $i \in \underline{a}$ only if $i \in \mathit{active}$, reversed components cannot have their current result set D_Q^i on top of any recv stack.

Additionally, since $\mathit{active} \neq \emptyset$, 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 $k \notin \mathit{active}$, then there is no p such that $(k, p, i) \in \mathit{comm}$. In that case, $k \notin \underline{a}$ either. Thus, $\mathit{recv}_S^{k,i} = \mathit{recv}_Q^{k,i} = \mathit{recv}_P^{k,i}$. On the other hand, if $k \in \mathit{active}$ and $(k, p, i) \in \mathit{comm}$ for some p, then $\mathit{recv}_Q^{k,i} = (D_Q^i, \{p \mid (k, p, i) \in \mathit{comm}\}) \bullet \mathit{recv}_P^{k,i}$. Now, because $k \in \underline{a}$, we have that $\mathit{recv}_S^{k,i} = \mathit{pop}(\mathit{recv}_Q^{k,i}) = \mathit{recv}_P^{k,i}$.

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

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

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

In this case $i \in \underline{a}$ as well, thus, $W_S^i = \mathrm{UniquePredecessor}_{A_i}(D_Q^i)$. Because $D_Q^i = \mathrm{res}_{A_i}(W_P^i)$ its unique predecessor is W_P^i. Therefore $W_S^i = W_P^i$.

Finally, if $i \notin active$ then

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

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 examine the other direction: constructing forward transitions from valid backward ones.

Definition 7.17. 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

$$active = \underline{a} \cap \{1, \ldots, n\},$$

$$comm = \bigcup_{i \in \underline{a}} \text{SentBy}_i \cup \bigcup_{i,\ env_i \in \underline{a}} \text{EnvironmentTo}_i,$$

and

$$\text{SentBy}_i = \{(i,p,k) \mid (p,k) \in \text{res}_{A_i}(\text{UniquePredecessor}_{A_i}(W_Q^i))\},$$

$$\text{EnvironmentTo}_i = \begin{cases} \{(env,p,i) \mid p \in C\}, \text{ if } top(recv_Q^{env,i}) = (D_Q^i, C), \\ \emptyset, \text{ otherwise.} \end{cases}$$

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. We chose the former approach for communication received from other components, 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 7.2. *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 $\underline{\underline{t}} : P \xrightarrow{\underline{\underline{a}}} Q$.*

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 $\underline{\underline{t}} : P \xrightarrow{\underline{\underline{a}}} S$ according to Definition 7.17. In what follows, we prove that $\underline{\underline{t}}$ 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 concerning the conditions of Definition 7.9.

(1) Since $\underline{a}$ is a valid backward label, $\underline{a} \cap \{1, \ldots, n\} \neq \emptyset$. Consequently $\mathit{active} = \underline{a} \cap \{1, \ldots, n\} \neq \emptyset$.

(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 equals the communicated entities of every stepped component. Thus, other elements of *comm* must be an environmental context of the form (env, p, k) for some p, k.

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

Now, we examine 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 7.13, 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 also holds for environmental input, 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. □

Given any transition, regardless of its direction, we can compute its inverse due to the above results. 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 [38].

Lemma 7.3 (Loop Lemma, [20], 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 $\underline{t} : Q \xrightarrow{a} P$ *there exists a forward transition* $\underline{\underline{t}} : P \xrightarrow{a} Q$

Proof. Follows from Lemma 7.1 and Lemma 7.2. □

By establishing all the prerequisites defined in [38], 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 7.4 (Square Property, [38], 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 7.12, distinct coinitial transitions are independent if they are not in conflict. Since we know that t and u are independent, they cannot conflict. This presents the same cases as that of Definition 7.15.

(1) If both transitions are forward, 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, 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 7.13) 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 7.15 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 7.5 (Backward Transitions are Independent, [38], 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 7.12, and the extension of the notion of conflict in Definition 7.15. □

Lemma 7.6 (Well-Foundedness, [38], 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 7.13. □

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 [20]. The Parabolic Lemma guarantees that reversibility does not introduce new processes that cannot be reached by pure forward computation.

Definition 7.18 (Parabolic Lemma, [38], 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, equal to the count of comprising transitions.

Proposition 7.1. *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 [38], Parabolic Lemma holds. □

We constructed Distributed cdcR systems with the intention of introducing concurrent behavior in the framework of communicating reaction systems. We aimed to examine how reaction systems, concurrency, and reversibility concepts 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. Compared to backward determinism and its concept of the last action, this allows for a relaxed reversible computational order.

Definition 7.19 (Causal-Consistency, [38], Definition 3.5)**.** Let r and s be two paths in an LTS. If r and s are coinitial and cofinal, they are causally equivalent.

Proposition 7.2. *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 [38], Causal-Consistency holds. □

8. Summary

In this dissertation, we presented our research about Reversible Reaction Systems. With the initial motivation of exploring different paradigms of reversibility within the same computational model, we started by implementing backtracking and finished with a variant that hosts causal-consistent reversibility. In what follows, we summarize the main contributions of this journey.

Backtracking is a paradigm of reversibility best suited for sequential models of computation. This paradigm builds on the notion of "last performed action" and involves recursive undo until reaching the desired state. Since the original definition of Reaction Systems describes a model that operates sequentially on the level of interactive processes (with parallelism only in terms of applying reactions), in Section 3, we implemented uncontrolled backtracking reversibility for Reaction Systems. Our implementation is based on the concept of non-restarting interactive processes: those that do not contain the empty result set twice. Then, using this notion, we defined that a reaction system is reversible if every non-restarting interactive process is reversible; that is, every state of the process has a unique predecessor. With this definition of reversibility, we stated the necessary and sufficient conditions for a reaction system to be reversible, posing restrictions on the reactions and creating a separate, non-disjoint input and product alphabet. The latter allows for input from the environment, keeping interactive processes interactive, as opposed to the definition of [5].

In Section 4, we extended our previous definition of Reversible Reaction Systems with external control. We constructed simulators of reversible systems that change the direction of computation based on the input from the environment. They are simulators because they compute the same states as the underlying simulated reversible systems with the option to undo and redo their previous computational step. We implemented undo in terms of inverse reactions that produce the unique predecessors. Such an undo operation is triggered by a distinguished environmental symbol, enabling externally controlled reversibility. Because of the no-permanency rule, unsustained symbols vanish, resulting in a redo operation unless the environment supplies another undo symbol in the subsequent computational step.

After defining a reversible model, our next step was to explore its intuitive computational power in Section 5. We did so with the help of transition graphs. Such a graph has a vertex for every result set of the system. Then, the vertices A and B are connected if some environmental input (that ap-

pears as the edge label) allows the underlying system to produce B from the result set A. Since such a graph includes every possible interactive process of the underlying reaction system (as an interactive process is equivalent to a path in the graph), it is the perfect tool to reason about the computational abilities of the model.

We first examined the transition graphs of ordinary reaction systems. They are somewhat limited: for an ordinary reversible reaction system, the transition graph is either a single vertex or a directed rooted tree with all the edges pointing away from the root. With the intention of relaxing the constraints, we then studied Initialized Reversible Reaction Systems: those that allow a non-empty starting result set. In the case of this model, the transition graphs are more complex: they can contain multiple components with up to one cycle per component.

The introduction of Reaction Systems Reversible with Lookbehind is a more fundamental twist on the base model: it allows the system to consider its most recent result and context set separately as opposed to only being capable of seeing their union (the current state). This results in a behavior similar to finite automata, drawing a parallel between the system's context and state set and the automata's input symbol and state. After looking at the transition graphs of this model, we concluded that they are much more capable as far as their intuitive computational power goes. The state transition graphs of reversible finite transition systems correspond to the transition graphs of Reaction Systems Reversible with Lookbehind. Vice-versa, the transition graph of any Reaction System Reversible with Lookbehind corresponds to the state transition graph of a reversible finite transition system.

Our initial motivation was experimenting with different reversibility paradigms, so we focused on causal consistency after backtracking. Causal consistency is suited for models lacking the "last performed action" concept, for example, models with concurrent computational semantics. In such a model, causal consistency defines the order of reversal on causal relationships, intuitively saying that an action can be reversed if all its consequences are undone first. However, the model of Reaction Systems is still sequential without any concurrent behavior. Thus, in Section 6, we investigated Communicating Reaction Systems with Direct Communication, initially defined by Csuhaj-Varjú and Sethy [19]. This model organizes an arbitrary number of reaction systems into a virtual graph. The individual systems (called components) can communicate with each other by sending and receiving products. Nevertheless, the computational steps are synchronous,

resulting in a lack of concurrency. Hence, as an interlude to highlight the limitations of global synchronization, we defined backtracking for this model based on our notion of individual reversible systems.

Section 7 introduced our variant of Communicating Reaction Systems called Distributed Communicating Reaction Systems (d-cdcR Systems, for short). This model removes global synchronization by defining blocking computation and active components. A component is in a blocked state if it has no enabled reactions. On the other hand, active components are those selected by external control to proceed with their computation. Of course, a component might not be blocked nor active: idle components have enabled reactions but are not selected to perform them. We achieve concurrent computational behavior by allowing external control to choose which components are active and idle.

With the last obstacle, the lack of concurrency, out of the way, we implemented causal consistent reversibility to the model. We did so based on the axiomatic framework of Lanese, Phillips, and Ulidowski [38]. This framework establishes axioms from which more involved properties, such as causal consistent reversibility, can be inferred.

Our implementation for causal consistent d-cdcR Systems comprises two layers. On the component level, we require local reversibility: each component must be reversible according to the results in Section 3. Then, on the level of the system as a whole, we track the communicated symbols using stacks attached to each component: we push onto the *recv* stack when receiving a symbol and onto *send* when sending one. Thus, these stacks capture and store the causal dependencies between the individual components. Translating this implementation to the axiomatic framework, we concluded the section proving the reversibility of d-cdcR systems.

The results of Sections 3 and 4 were published in the journal paper [J2], the material of Section 5 is based on the conference paper [I1], and the results of Sections 6 and 7 are from the journal paper [J1].

9. Összefoglalás

E disszertációban a reakciórendszerek (Reaction Systems) nevű számítási modell reverzibilitását vizsgáló kutatásunkat mutattuk be. Motivációnkat az jelentette, hogy különböző reverzibilitási paradigmákat próbálhassunk ki egyazon számítási modellen belül. Ennek megfelelően, munkánk a visszalépéses (backtracking) reverzibilitás megvalósításától egy olyan variánsig terjedt, mely az okozati konzisztens (causal consistent) reverzibilitás implementálására is alkalmas. A következőkben ezeket, továbbá a köztes lépéseket foglaljuk össze röviden.

A visszalépéses reverzibilitási paradigma olyan számítási módszerek esetén ideális, melyek a számításokat lépésről lépésre, azaz szekvenciálisan végzik. A paradigma középpontjában az utolsóként végrehajtott lépés áll; ennek az ismételt visszavonásával juthatunk el a kívánt, múltbéli állapotig. Mivel a reakciórendszerek is szekvenciális végrehajtással rendelkeznek, ezért természetesen adódott, hogy elsőként a kontrollálatlan visszalépéses paradigmát valósítjuk meg benne, melyhez kötődő eredményeinket a 3. fejezet mutatja be. Megvalósításunk az úgynevezett nem-újrainduló interaktív folyamatokra épül, melyek az üres állapotot csak egyszer tartalmazzák, megakadályozva ezzel, hogy az üres állapotból lényegében újrainduljon a számítási folyamat. Definíciónk szerint egy reakciórendszer reverzibilis, amennyiben annak minden interaktív folyamata reverzibilis. E definícióra építve meghatároztuk a reverzibilitás szükséges és elégséges feltételeit, megfelelő követelményeket támasztva a reakciókkal és a háttérhalmazzal szemben. Utóbbi követelmény a halmaz két, nem feltétlenül diszjunkt részre osztását jelenti: az egyik az eredményhalmaz, melyből a reakciók eredményszimbólumai kerülhetnek ki, míg a másik a bemeneti halmaz, mely a lehetséges bemeneti szimbólumokat tartalmazza. E feltételnek köszönhetően lesz a megoldásunk egyszerre reverzibilis és interaktív, szemben az Aman és Ciobanu által elkészített megvalósítással [5].

A 4. fejezetben az előzőleg felírt reverzibilis reakciórendszer definíciónkat egészítettük ki egy külső kontroll mechanizmussal (external control). Azaz, egy olyan modellt készítettünk, melyben a számítás iránya a környezettől kapott bemenettől függ. E modellt szimulátornak nevezzük, hiszen pontosan ugyanazon állapotok kiszámítására képes, mint egy választott szimulált reverzibilis reakciórendszer, kiegészítve azonban egy visszavonási mechanizmussal (undo-redo). Undo, azaz visszavonás akkor történik, amikor a környezet egy speciális bemeneti szimbólumot ad a rendszernek. Amennyiben a környezet nem biztosít egy újabb, visszalépést kiváltó szimbólumot, e szim-

bólum azonnal eltűnik, egy redo (avagy, ismét) lépést eredményezve. Azaz, a szimulátor kívülről kontrollált reverzibilitást ad a kontroll nélküli reverzibilis modellhez.

A reverzibilis modell definiálása után, az 5. fejezetben annak intuitív számítási erejét tekintettük, segítségül hívva az átmeneti gráfokat (transition graph). Az átmeneti gráf egy olyan eszköz, mely lehetővé teszi, hogy egy adott reakciórendszer összes interaktív folyamtát együttesen ábrázoljuk és vizsgáljuk. Felírásának alapját a vizsgált reakciórendszer eredményhalmazai (result set) képzik, melyek mindegyikéhez egy-egy csúcsot rendelünk. Ezt követően az A halmazt reprezentáló csúcsból egy irányított élt húzunk a B halmaznak megfelelő csúcsba, amennyiben van olyan reakcióhalmaz és környezeti bemenet (utóbbi lesz az élhez tartozó címke), melyekkel A-ból előáll B.

Elsőként a 3. fejezet definíciójának megfelelő reverzibilis reakciórendszereket tekintettük, melyek igen korlátozottak: egy ilyen rendszer átmeneti gráfja vagy egyetlen csúcs, vagy pedig egy olyan irányított fa, melyben minden él a gyökérrel ellentétes irányba mutat. Lazítandó a modell által szabott korlátokon, következőnek az inicializált reverzibilis reakciórendszereket vizsgáltuk. E modellben a kezdeti eredményhalmaz bármilyen halmaz lehet, nem csak az üres halmaz. Csupán ez az egyszerű módosítás alapvető változást jelentett a modell által generált átmeneti gráfokban, hiszen egy gráf immár több komponenst is tartalmazhatott, mely komponensek mindegyikében megjelenhetett legfeljebb egy kör is.

Folytatva vizsgálódásainkat, egy még alapvetőbb változtatást eszközöltünk, bevezetve a visszatekintő reverzibilis reakciórendszereket. Míg egy hagyományos rendszer csupán az aktuális állapothalmazt látja, addig egy visszatekintő rendszer külön-külön hozzáfér az eredmény- és a bemeneti halmazhoz. Ezáltal a modell viselkedése hasonló a véges automatákéhoz, hiszen párhuzamot vonhatunk a reakciórendszer bemeneti és állapothalmaza, valamint a véges automata bemeneti szimbóluma és állapota között. E változtatás eredményeként egy olyan modell állt elő, melynek viselkedése lényegében megegyezik a véges állapotátmeneti rendszerekével (finite transition system). Azaz, a reverzibilis véges állapotátmeneti rendszerek állapotátmeneti gráfjai megfelelnek a visszatekintő reverzibilis reakciórendszerek átmeneti gráfjainak. És fordítva, bármely visszatekintő reverzibilis reakciórendszer gráfjából felírható egy reverzibilis véges állapot-átmeneti rendszer állapot-átmeneti gráfja.

A kutatómunkánk elsődleges célja és motivációja a reverzibilitás különböző paradigmáinak vizsgálata volt. E motiváció mentén, a visszalépéses

reverzibilitás után a figyelmünket egy másik paradigmára, az okozati konzisztenciára (causal consistency) fordítottuk. Szemben a visszalépéssel, az okozati konzisztencia olyan számítási modellek esetén alkalmazható, melyekben a legutoljára végrehajtott számítási lépés nem azonosítható egyértelműen. Ilyenek például a konkurens számításokat lehetővé tevő modellek. Mivel ezek a modellek nem definiálják a számítási lépések globális sorrendjét, ezért az okozati konzisztencia az ok-okozati viszonyokat használja fel az egyes lépések visszavonásához. Egyszerűen, azt a – valós tapasztalatainkból is jól ismert – gondolatot valósítja meg, hogy egy lépést csak akkor vonhatunk vissza, ha előtte az összes következményét visszavontuk. Tehát, az ok visszavonását meg kell előznie az okozat visszavonásának.

Visszatekintve azonban a reakciórendszer eredeti definíciójára, az egy szekvenciális modellt ír le, mely nem tartalmaz konkurens viselkedést, értelmetlenné téve ezzel az okozati konzisztencia alkalmazását. Ahhoz, hogy a szekvenciális végrehajtáson továbbléphessünk, egy megfelelő variánsra van szükségünk. Egy ilyen variánst mutat be a 6. fejezet, a közvetlenül kommunikáló reakciórendszerek (Communicating Reaction Systems with Direct Communication, cdcR systems) formájában [19]. E modell tetszőleges számú reakciórendszerből épít egy virtuális gráfot, mely rendszerek aztán kommunikálhatnak is egymással. A kommunikáció szimbólumokon keresztül történik: a gráf komponensei képesek egymásnak elküldeni a reakciók alkalmazásának eredményét. Mindazonáltal, hiába lesz így elosztott a modell, a működése továbbra is szinkronizált. Ennek folytán, elsőként a visszalépést implementáltuk a modellben, hogy illusztráljuk a szinkronizáció okozta korlátokat.

A 7. fejezetben bemutattunk egy általunk felírt közvetlenül kommunikáló reakciórendszer-variánst, melyet közvetlenül kommunikáló elosztott reakciórendszernek (Distributed Communicating Reaction Systems, d-cdcR systems) neveztünk el. E változat bevezeti a blokkolt (blocking) számításokat és az aktív komponenseket, megszüntetve ezzel a globális szinkronizációt. Egy komponens blokkolt állapotban van, amennyiben nem rendelkezik alkalmazható reakciókkal. Ezzel szemben, aktívak lesznek azok a komponensek, melyeket a külső kontroll továbblépésre jelöl ki. Természetesen, lesznek olyan komponensek is, melyek se nem blokkoltak, se nem aktívak: a tétlen (idle) komponensek rendelkeznek ugyan alkalmazható reakciókkal, ugyanakkor nem jelöli ki őket a külső kontroll a továbblépésre. Pontosan e mechanizmusnak köszönhetően válnak lehetségessé a konkurens számítások, hiszen minden számítási lépésben a külső kontroll dönthet arról, mely komponensek legyenek aktívak és melyek tétlenek.

Mivel e módon lehetségessé váltak a konkurens számítások, így erre a

modellre már felírhattuk az okozati konzisztens reverzibilitást, felhasználva Lanese, Phillips és Ulidowski kapcsolódó keretrendszerét [38]. E keretrendszer olyan axiómákat definiál, melyekre felépíthetők olyan, összetettebb tulajdonságok, mint például az okozati konzisztens reverzibilitás.

Implementációnk két rétegből áll. Egyfelől, megköveteltük, hogy minden egyes komponens lokálisan reverzibilis legyen, megfelelve a 3. fejezetben támasztott követelményeknek. Másfelől, a teljes d-cdcR rendszert tekintve, az egyes komponensek között zajló kommunikációt a komponensekhez csatolt vermekben tároltuk: a *recv* veremre akkor helyezünk új elemet, ha a komponens fogad valamilyen szimbólumokat, míg a *send* verem küldéskor kerül új elem. Mivel minden kommunikáció megjelenik e vermeken, így lehetővé teszik a komponensek között létrejövő függőségek követését. E két réteget lefordítva aztán az előzőleg említett axiomatikus keretrendszerre bizonyítottuk, hogy a d-cdcR modell okozati konzisztens módon reverzibilis.

A 3. és 4. fejezetben bemutatott eredményeket a [J2] folyóiratcikkben közöltük, az 5. fejezet az [I1] konferenciacikk alapján készült, míg a 6. és 7. fejezet a [J1] folyóiratcikk eredményeit tartalmazza.

10. Köszönetnyilvánítás

Bár egy disszertáció megírása semmilyen szempontból sem tekinthető egyszerűnek, mégis, én úgy gondolom, hogy az ezzel töltött néhány hónap volt a kutatással eltöltött éveim legegyértelműbb feladata. E feladat egyértelmű, már-már egyszerű volta pedig a megelőző évek kemény, néha akár kilátástalan erőfeszítéseinek eredménye. Ezek pedig nem is a PhD-képzéssel, hanem már sokkal korábban elkezdődtek; ezért a köszönetnyilvánításomban szeretnék sorra venni mindenkit, aki ezen a hosszú úton akár ilyen, akár olyan formában segítséget nyújtott.

Mindenekelőtt, köszönettel tartozok Battyányi Péternek, aki a közös munkával elindított a kutatóvá válás útján.

Köszönet illeti Pethő Attilát, akinek tanácsai, témavezetése révén egy OTDK III. helyezést érhettünk el, felcsillantva a tudományos munkában rejlő lehetőségeket. Hálás vagyok továbbá, hogy Pethő Attila mindvégig egyengette a pályafutásomat, akár jótanácsok, akár különböző pályázatok támogatásának formájában.

Szeretném megköszönni a Számítógéptudományi Tanszék oktatóinak a szakmai hozzájárulását is: minden alkalommal javaslatokkal, elmélyült diszkusszióval segítették a kutatásomat.

Kevésbé formális, de annál nagyobb segítséget nyújtottak a barátaim. Közülük is elévülhetetlen érdemekkel rendelkezik Vécsi Ádám, akivel évekig napi szinten együtt dolgoztunk. Köszönet illeti továbbá Tóth Róbertet is, akinél jobban senki sem ismeri a pályázatokat és persze – a határidőket.

A szakmai segítség azonban csupán az érem egyik oldalát jelenti. A szüleim, családom támogatása nélkül egészen biztosan nem tudtam volna eljutni idáig. Kiváltképp hálás vagyok azért, hogy bármely döntést is hoztam, mindig szeretettel támogattak, még akkor is, amikor a doktori tanulmányaim megszakítása mellett döntöttem. Köszönöm a barátnőmnek, Csengének is, hogy bármennyit is kellett dolgoznom, mindig levette a vállamról a terheket, lehetővé téve, hogy a kutatással foglalkozhassak.

Köszönetnyilvánításom végére szerettem volna hagyni a témavezetőmet, Vaszil Györgyöt, akinek akármilyen hosszú bekezdést szentelnék, sosem volna a szerepéhez méltó. Mindenekelőtt, végtelenül hálás vagyok azért, hogy elvállalt, mint doktori hallgatót, és mindvégig hitt abban, hogy ez a disszertáció el fog készülni. Köszönöm, hogy ott volt minden leadási határidőnél, minden visszadobott változatnál, minden utolsó utáni korrekciónál. Hálás vagyok a sok baráti jellegű beszélgetésért, továbbá a számtalan, informatikai témájú diskurzusért, melyekben mindvégig egyenrangú partnerként kezelt.

References

[1] Oana Agrigoroaiei and Gabriel Ciobanu. „Dual P Systems". In: *Membrane Computing - 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers*. Ed. by David W. Corne, Pierluigi Frisco, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. Vol. 5391. Lecture Notes in Computer Science. Springer, 2008, pp. 95–107. DOI: 10.1007/978-3-540-95885-7_7.

[2] Oana Agrigoroaiei and Gabriel Ciobanu. „Reversing computation in membrane systems". In: *J. Log. Algebraic Methods Program.* 79.3-5 (2010), pp. 278–288. DOI: 10.1016/j.jlap.2010.03.003.

[3] Artiom Alhazov, Rudolf Freund, and Kenichi Morita. „Sequential and maximally parallel multiset rewriting: reversibility and determinism". In: *Nat. Comput.* 11.1 (2012), pp. 95–106. DOI: 10.1007/s11047-011-9267-8.

[4] Bogdan Aman. „From Networks of Reaction Systems to Communicating Reaction Systems and Back". In: *Machines, Computations, and Universality*. Ed. by Jérôme Durand-Lose and György Vaszil. Cham: Springer International Publishing, 2022, pp. 42–57. ISBN: 978-3-031-13502-6.

[5] Bogdan Aman and Gabriel Ciobanu. „Controlled Reversibility in Reaction Systems". In: *Membrane Computing*. Ed. by Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron. Cham: Springer International Publishing, 2018, pp. 40–53. ISBN: 978-3-319-73359-3.

[6] Bogdan Aman and Gabriel Ciobanu. „Mutual exclusion and reversibility in reaction systems". In: *Journal of Membrane Computing* 2 (Oct. 2020), pp. 171–178. DOI: 10.1007/s41965-020-00043-1.

[7] Bogdan Aman and Gabriel Ciobanu. „Reversibility in Parallel Rewriting Systems". In: *J. UCS* 23.7 (2017), pp. 692–703.

[8] Dana Angluin. „Inference of Reversible Languages". In: *J. ACM* 29.3 (1982), pp. 741–765. DOI: 10.1145/322326.322334.

[9] Holger Bock Axelsen and Robert Glück. „On reversible Turing machines and their function universality". In: *Acta Informatica* 53.5 (2016), pp. 509–543. DOI: 10.1007/s00236-015-0253-y.

[10] Attila Bagossy and György Vaszil. „Transition Graphs of Reversible Reaction Systems". In: *Membrane Computing*. Ed. by Rudolf Freund, Tseren-Onolt Ishdorj, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron. Cham: Springer International Publishing, 2021, pp. 1–16. ISBN: 978-3-030-77102-7.

[11] C. H. Bennett. „Logical Reversibility of Computation". In: *IBM Journal of Research and Development* 17.6 (Oct. 1973), pp. 525–532.

[12] Paolo Bottoni, Anna Labella, and Grzegorz Rozenberg. „Networks of Reaction Systems". In: *International Journal of Foundations of Computer Science* 31 (Jan. 2020), pp. 53–71. DOI: 10.1142/S0129054120400043.

[13] Paolo Bottoni, Anna Labella, and Grzegorz Rozenberg. „Reaction systems with influence on environment". In: *Journal of Membrane Computing* 1 (Mar. 2019). DOI: 10.1007/s41965-018-00005-8.

[14] Robert Brijder, Andrzej Ehrenfeucht, Michael Main, and Grzegorz Rozenberg. „A Tour of reaction Systems." In: *Int. J. Found. Comput. Sci.* 22 (Nov. 2011), pp. 1499–1517. DOI: 10.1142/S0129054111008842.

[15] Robert Brijder, Andrzej Ehrenfeucht, and Grzegorz Rozenberg. „Reaction Systems with Duration". In: *Computation, Cooperation, and Life: Essays Dedicated to Gheorghe Păun on the Occasion of His 60th Birthday*. Ed. by Jozef Kelemen and Alica Kelemenová. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 191–202. ISBN: 978-3-642-20000-7. DOI: 10.1007/978-3-642-20000-7_16. URL: https://doi.org/10.1007/978-3-642-20000-7_16.

[16] Lucie Ciencialová, Luděk Cienciala, and Erzsébet Csuhaj-Varjú. „Languages of Distributed Reaction Systems". In: *Machines, Computations, and Universality*. Ed. by Jérôme Durand-Lose and György Vaszil. Cham: Springer International Publishing, 2022, pp. 75–90. ISBN: 978-3-031-13502-6.

[17] Ioana Cristescu, Jean Krivine, and Daniele Varacca. „A Compositional Semantics for the Reversible pi-Calculus". In: June 2013, pp. 388–397. ISBN: 978-1-4799-0413-6. DOI: 10.1109/LICS.2013.45.

[18] Erzsébet Csuhaj-Varjú and Pramod Sethy. „Properties of Communicating Reaction Systems". In: *Proceedings of the 21st Conference Information Technologies - Applications and Theory*. Ed. by Broňa Brejová, Lucie Ciencialová, Martin Holeňa, František Mráz, Dana Pardubská Martin Plátek, and Tomáš Vinař. Sept. 2021, pp. 217–221.

[19] Erzsébet Csuhaj-Varjú and Pramod Kumar Sethy. „Communicating Reaction Systems with Direct Communication". In: *Membrane Computing*. Ed. by Rudolf Freund, Tseren-Onolt Ishdorj, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron. Cham: Springer International Publishing, 2021, pp. 17–30. ISBN: 978-3-030-77102-7.

[20] Vincent Danos and Jean Krivine. „Reversible Communicating Systems". en. In: *CONCUR 2004 - Concurrency Theory*. Ed. by Philippa Gardner and Nobuko Yoshida. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 292–307. ISBN: 9783540286448. DOI: 10.1007/978-3-540-28644-8_19.

[21] Vincent Danos and Jean Krivine. „Transactions in RCCS". In: *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*. Ed. by Martín Abadi and Luca de Alfaro. Vol. 3653. Lecture Notes in Computer Science. Springer, 2005, pp. 398–412. DOI: 10.1007/11539452_31.

[22] A. Ehrenfeucht and G. Rozenberg. „Reaction Systems". In: *Fundam. Inf.* 75.1–4 (Jan. 2007), pp. 263–280. ISSN: 0169-2968.

[23] Michael P. Frank. „Introduction to Reversible Computing: Motivation, Progress, and Challenges". In: *Proceedings of the 2nd Conference on Computing Frontiers*. CF '05. New York, NY, USA: Association for Computing Machinery, 2005, pp. 385–390. ISBN: 1595930191. DOI: 10.1145/1062261.1062324.

[24] Daniela Genova, Hendrik Jan Hoogeboom, and Nataša Jonoska. „A graph isomorphism condition and equivalence of reaction systems". In: *Theoretical Computer Science* 701 (2017), pp. 109–119.

[25] Robert Glück, Ivan Lanese, Claudio Antares Mezzina, Jarosław Adam Miszczak, Iain Phillips, Irek Ulidowski, and Germán Vidal. „Towards a Taxonomy for Reversible Computation Approaches". In: *Reversible Computation*. Ed. by Martin Kutrib and Uwe Meyer. Cham: Springer Nature Switzerland, 2023, pp. 24–39. ISBN: 978-3-031-38100-3.

[26] Mika Hirvensalo. „On probabilistic and quantum reaction systems". In: *Theoretical Computer Science* 429 (2012). Magic in Science, pp. 134–143. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2011.12.032. URL: https://www.sciencedirect.com/science/article/pii/S0304397511009984.

[27] Markus Holzer and Martin Kutrib. „Reversible Nondeterministic Finite Automata". In: *Reversible Computation*. Ed. by Iain Phillips and Hafizur Rahaman. Cham: Springer International Publishing, 2017, pp. 35–51. DOI: 10.1007/978-3-319-59936-6_3.

[28] Markus Holzer and Christian Rauch. „Computational Complexity of Reversible Reaction Systems". In: *Reversible Computation*. Ed. by Martin Kutrib and Uwe Meyer. Cham: Springer Nature Switzerland, 2023, pp. 40–54. ISBN: 978-3-031-38100-3.

[29] Oscar H. Ibarra. „On Strong Reversibility in P Systems and Related Problems". In: *Int. J. Found. Comput. Sci.* 22.1 (2011), pp. 7–14. DOI: 10.1142/S0129054111007782.

[30] Lila Kari and Grzegorz Rozenberg. „The Many Facets of Natural Computing". In: *Commun. ACM* 51 (Oct. 2008), pp. 72–83. DOI: 10.1145/1400181.1400200.

[31] Stefan Kuhn, Bogdan Aman, Gabriel Ciobanu, Anna Philippou, Kyriaki Psara, and Irek Ulidowski. „Reversibility in Chemical Reactions". In: *Reversible Computation: Extending Horizons of Computing: Selected Results of the COST Action IC1405*. Ed. by Irek Ulidowski, Ivan Lanese, Ulrik Pagh Schultz, and Carla Ferreira. Cham: Springer International Publishing, 2020, pp. 151–176. ISBN: 978-3-030-47361-7. DOI: 10.1007/978-3-030-47361-7_7. URL: https://doi.org/10.1007/978-3-030-47361-7_7.

[32] R. Landauer. „Irreversibility and Heat Generation in the Computing Process". In: *IBM Journal of Research and Development* 5.3 (July 1961), pp. 183–191.

[33] Ivan Lanese, Claudio Antares Mezzina, Alan Schmitt, and Jean-Bernard Stefani. „Controlling Reversibility in Higher-Order Pi". In: *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*. Ed. by Joost-Pieter Katoen and Barbara König. Vol. 6901. Lecture Notes in Computer Science. Springer, 2011, pp. 297–311. DOI: 10.1007/978-3-642-23217-6_20. URL: https://doi.org/10.1007/978-3-642-23217-6_20.

[34] Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. „Controlled Reversibility and Compensations". en. In: *Reversible Computation*. Ed. by Robert Glück and Tetsuo Yokoyama. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 233–240. ISBN: 9783642363153. DOI: 10.1007/978-3-642-36315-3_19.

[35] Ivan Lanese, Claudio Antares Mezzina, and Francesco Tiezzi. „Causal-Consistent Reversibility". In: *Bulletin-European Association for Theoretical Computer Science* 114 (2014), p. 17.

[36] Ivan Lanese, Naoki Nishida, Adrián Palacios, and Germán Vidal. „A theory of reversibility for Erlang". In: *Journal of Logical and Algebraic Methods in Programming* 100 (2018), pp. 71–97. ISSN: 2352-2208. DOI: https://doi.org/10.1016/j.jlamp.2018.06.004. URL: https://www.sciencedirect.com/science/article/pii/S2352220817301402.

[37] Ivan Lanese, Naoki Nishida, Adrián Palacios, and Germán Vidal. „CauDEr: A Causal-Consistent Reversible Debugger for Erlang". In: *Functional and Logic Programming*. Ed. by John P. Gallagher and Martin Sulzmann. Cham: Springer International Publishing, 2018, pp. 247–263. ISBN: 978-3-319-90686-7.

[38] Ivan Lanese, Iain Phillips, and Irek Ulidowski. „An Axiomatic Approach to Reversible Computation". In: *Foundations of Software Science and Computation Structures*. Ed. by Jean Goubault-Larrecq

and Barbara König. Cham: Springer International Publishing, 2020, pp. 442–461. ISBN: 978-3-030-45231-5.

[39] Hernán Melgratti, Claudio Antares Mezzina, and Irek Ulidowski. „Reversing Place Transition Nets". In: *Logical Methods in Computer Science* Volume 16, Issue 4 (Oct. 2020). DOI: 10.23638/LMCS-16(4:5)2020. URL: https://lmcs.episciences.org/6843.

[40] Artur Meski, Maciej Koutny, and Wojciech Penczek. *Model Checking for Temporal-Epistemic Properties of Distributed Reaction Systems*. Tech. rep. CS-TR-1526. School of Computing, Newcastle University, Apr. 2019. DOI: 10.13140/RG.2.2.26859.80161.

[41] Artur Meski, Wojciech Penczek, and Grzegorz Rozenberg. „Model checking temporal properties of reaction systems". In: *Inf. Sci.* 313 (2015), pp. 22–42. DOI: 10.1016/j.ins.2015.03.048.

[42] R. Milner. *Communication and Concurrency*. USA: Prentice-Hall, Inc., 1989. ISBN: 0131149849.

[43] Kenichi Morita. *Theory of Reversible Computing*. Springer Japan, 2017. DOI: 10.1007/978-4-431-56606-9.

[44] Benedek Nagy. „State-deterministic $5' \to 3'$ Watson-Crick automata". In: *Natural Computing* 20 (Dec. 2021). DOI: 10.1007/s11047-021-09865-z.

[45] Naoki Nishida, Adrián Palacios, and Germán Vidal. „Reversible computation in term rewriting". In: *J. Log. Algebraic Methods Program.* 94 (2018), pp. 128–149.

[46] Taishin Yasunobu Nishida. „Reversible P Systems with Symport/Antiport Rules". In: *WMC 10*. Ed. by G. Paun, M.J. Pérez-Jiménez, and A. Riscos-Núñez. 2009, pp. 452–460.

[47] Gheorghe Păun. „Computing with Membranes". In: *J. Comput. Syst. Sci.* 61.1 (2000), pp. 108–143. DOI: 10.1006/jcss.1999.1693.

[48] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, eds. *The Oxford Handbook of Membrane Computing*. USA: Oxford University Press, Inc., 2010. ISBN: 0199556679.

[49] Kalyan S. Perumalla. *Introduction to Reversible Computing*. Chapman & Hall/CRC, 2013.

[50] Iain Phillips, Irek Ulidowski, and Shoji Yuen. „A Reversible Process Calculus and the Modelling of the ERK Signalling Pathway". en. In: *Reversible Computation*. Ed. by Robert Glück and Tetsuo Yokoyama. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 218–232. ISBN: 9783642363153. DOI: 10.1007/978-3-642-36315-3_18.

[51] Jean-Eric Pin. „On reversible automata". In: *Proceedings of the first LATIN conference*. Ed. by I. Simon. Lecture Notes in Computer Science 583. Springer, 1992, pp. 401–416. URL: https://hal.archives-ouvertes.fr/hal-00019977.

[52] Michele G. Pinna. „Reversing Steps in Membrane Systems Computations". en. In: *Membrane Computing*. Ed. by Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 245–261. ISBN: 9783319733593. DOI: 10.1007/978-3-319-73359-3_16.

[53] Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok. *Handbook of Natural Computing*. Springer, Berlin, Heidelberg, 2012. ISBN: 978-3-540-92909-3. DOI: 10.1007/978-3-540-92910-9.

[54] A.C. Cem Say and Abuzer Yakaryılmaz. „Quantum Finite Automata: A Modern Introduction". In: *Computing with New Resources: Essays Dedicated to Jozef Gruska on the Occasion of His 80th Birthday*. Ed. by Cristian S. Calude, Rūsiņš Freivalds, and Iwama Kazuo. Cham: Springer International Publishing, 2014, pp. 208–222. ISBN: 978-3-319-13350-8. DOI: 10.1007/978-3-319-13350-8_16. URL: https://doi.org/10.1007/978-3-319-13350-8_16.

[55] Irek Ulidowski, Ivan Lanese, Ulrik Pagh Schultz, and Carla Ferreira, eds. *Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405*. Vol. 12070. Lecture Notes in Computer Science. Springer, 2020. ISBN: 9783030473600. DOI: 10.1007/978-3-030-47361-7.

A. List of relevant publications

Journal papers

[J1] Attila Bagossy and György Vaszil. „Controlled reversibility in communicating reaction systems". In: *Theoretical Computer Science* 926 (2022), pp. 3–20. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2022.05.030. URL: https://www.sciencedirect.com/science/article/pii/S0304397522003528.

[J2] Attila Bagossy and György Vaszil. „Simulating reversible computation with reaction systems". In: *Journal of Membrane Computing* 2 (Oct. 2020), pp. 179–193. DOI: 10.1007/s41965-020-00049-9.

Papers in conference proceedings

[I1] Attila Bagossy and György Vaszil. „Transition Graphs of Reversible Reaction Systems". In: *Membrane Computing*. Ed. by Rudolf Freund, Tseren-Onolt Ishdorj, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron. Cham: Springer International Publishing, 2021, pp. 1–16. ISBN: 978-3-030-77102-7.

Conference talks

[C1] Attila Bagossy. „Paradigms of Reversibility in Reaction Systems". 12th International Workshop on Non-Classical Models of Automata and Applications. Debrecen, 2022. URL: https://konferencia.unideb.hu/en/program-ncma-2022.

[C2] Attila Bagossy. „Reverzibilitás és annak szimulációja a Reaction Systems számítási modellben". ADA. Debrecen, 2020. URL: https://konferencia.unideb.hu/hu/program-december-15-ada-konferencia.

[C3] Attila Bagossy. „Simulating Reversible Computation with Reaction Systems". 11th International Conference on Applied Informatics. Eger, 2020. URL: https://icai.uni-eszterhazy.hu/2020/?program#j29c.