International Journal on Software Tools for Technology Transfer (2024) 26:509–526
https://doi.org/10.1007/s10009-024-00757-y

GENERAL

Special Section: Challenges of Software Verification

Causal analysis of positive Reaction Systems

Linda Brodo[1] · Roberto Bruni[2] · Moreno Falaschi[3] · Roberta Gori[2] · Paolo Milazzo[2] · Valeria Montagna[2] · Pasquale Pulieri[2]

Accepted: 4 June 2024 / Published online: 19 June 2024
© The Author(s) 2024

Abstract

Cause/effect analysis of complex systems is instrumental in better understanding many natural phenomena. Moreover, formal analysis requires the availability of suitable abstract computational models that somehow preserve the features of interest. Our contribution focuses on the analysis of Reaction Systems (RSs), a qualitative computational formalism inspired by biochemical reactions in living cells. The primary challenge lies in dealing with inhibition mechanisms. On the one hand, inhibitors enhance the expressiveness of the computational abstraction; on the other hand, they can introduce nonmonotonic behaviors that can be computationally hard to deal with in the analysis. We propose an encoding of RSs into an equivalent formulation without inhibitors (called Positive RSs, PRSs for short) that is easier to handle, because PRSs exhibit monotonic behaviors. The effectiveness of our transformation is witnessed by its impact on two different techniques for cause/effect analysis. The first, called slicing, allows detecting the causes of some unforeseen phenomenon by reasoning backward along a given computation. Here, PRSs can be exploited to improve the quality of the analysis. The second technique, predictor analysis, is addressed by introducing a novel tool called MuMa, which is based on must/maybe sets, whence the tool name, an original abstraction for approximating ancestor formulas. MuMa exploits PRSs to improve the performance of the analysis.

Keywords Reaction Systems · Positive reactions · Abstract interpretation · Ancestor formula · Slicing

✉ L. Brodo
brodo@uniss.it

R. Bruni
roberto.bruni@unipi.it

M. Falaschi
moreno.falaschi@unisi.it

R. Gori
roberta.gori@unipi.it

P. Milazzo
paolo.milazzo@unipi.it

V. Montagna
v.montagna@studenti.unipi.it

P. Pulieri
p.pulieri@studenti.unipi.it

1 Dipartimento di Scienze Economiche e Aziendali, Università di Sassari, Sassari, Italy

2 Dipartimento di Informatica, Università di Pisa, Pisa, Italy

3 Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena, Siena, Italy

1 Introduction

Causality Cause/effect analysis (also called causality or causation) is the investigation of which event or action in the past, i.e., the cause, is *responsible* for the production of another event in the future, i.e., the effect. When such cause/effect relationship is established, we also say that the effect *depends on* its causes. The principle of causality is crucial in many disciplines, because it allows, e.g., foreseeing the future consequences of the available actions in decision making processes, or explaining the reasons why a certain error appears in automatic recovery mechanisms. In the modeling of biological systems, cause/effect analysis techniques can serve, e.g., to investigate and possibly predict which drugs are more effective for a given disease or to circumscribe the factors that led to the presence of an infection. To investigate causality from a formal point of view, one needs, of course, to design suitable computational models for the complex system under inspection.

Different notions of causality have been studied for different calculi (e.g., in systems biology by Bodei et al. [12], Gori and Levi [32] and for natural computing in Busi [20]) in this paper, we focus on a well-known computational model for the biochemical reactions in the living cell, called Reaction

Systems (RSs), as proposed by Ehrenfeucht and Rozenberg [29].

Reaction Systems Reaction Systems (RSs) are a qualitative computational formalism inspired by facilitation and inhibition mechanisms of biochemical processes. An RS is composed of a set of entities and a set of reactions. Each reaction involves some reactants, inhibitors, and products. Reactants are the entities required for the reaction to take place, while inhibitors are those entities whose presence would prevent the reaction from occurring. The dynamics of an RS is defined in terms of discrete steps: The current state of the system is defined by the set of entities that are present; when a reaction is enabled in the current state, it releases its set of products; all enabled reactions take place together at each step. It is important to remark that enabled reactions do not compete for the same reactant: thanks to the *threshold supply assumption*, whenever an entity is present in the current state, it is assumed that it is sufficient to enable all reactions that requires it. RSs have shown to be a computational model whose application ranges from the modeling of biological phenomena, see Azimi [4], Azimi et al. [5], Barbuti et al. [6], Corolli et al. [22] to molecular chemistry as in Okubo and Yokomori [36] and computer science as in Brijder et al. [14], Corolli et al. [22].

In the context of RSs, the study of causality amounts to establishing the way in which entities can influence each other. Despite the deterministic nature of RSs, many computational problems of this kind are shown to be intractable in Dennunzio et al. [27, 28], Formenti et al. [31], Salomaa [37, 38]. However, several techniques have been introduced to study causal dependencies in RSs. In this paper we study possible improvements for two such techniques, namely slicing (Brodo et al. [18]) and predictor analysis (Barbuti et al. [6]). Although slicing and predictor analysis are two very different techniques, the two improvements we will propose for them are both benefiting from the same idea: translating the RS under study into a form (called positive) in which behaviors become monotonic. This permits us to develop approaches that are more accurate (by allowing also inhibitors to be dealt with in slicing) and scalable (by introducing abstraction techniques for predictors analysis) than existing ones.

Dynamic slicing Slicing aims at discarding superfluous information when investigating the emergence of some phenomenon. Given the presence of a set of entities D in the current state, slicing is used to ascertain their origins in the initial state of the current computation, i.e., the set of initial entities ultimately responsible for the production of D. Initially applied to imperative programs in Korel and Laski [33] to facilitate the debugging process by pinpointing faulty code segments, dynamic program slicing has since been adapted for many programming paradigms (see Silva [39] for a survey). Leveraging the process algebraic version of Reaction Systems proposed in Brodo et al. [15, 16, 17], the dynamic slicing technique designed in Brodo et al. [18] commences from a partial computation viewed as erroneous. Its goal is to determine the minimal set of causes that led to the erroneous state in that specific trace. More broadly, dynamic slicing can be employed to discern the causes behind the production of an entity in a specific execution trace.

Predictor analysis Predictors are formulas that characterize all causes responsible for the production of some entity after some steps. A crucial distinction from slicing is that predictor analysis must account for the entire set of reactions that form the model, rather than just those executed in a specific trace. Given a target set of entities and a number of steps, the *ancestor formula* (i.e., the predictor) exactly delineates which entities must be initially present or absent to yield the target entities in the designated number of steps, as detailed in Barbuti et al. [6].

Inhibitors One significant source of complexity in the above analyses stems from the involvement of inhibitors in the reactions. This is because inhibitors introduce non-monotonic behavior, meaning that when a set of entities enable some reactions, then a superset could include some inhibitors for those reactions and they would not be longer enabled. Therefore, it can happen that the result set shrink when the set of entities grows. As a direct consequence, causal analysis techniques can become computationally intensive because, e.g., they cannot rely on the fact that, given a minimal set of causes for a certain phenomenon, any of its superset will still cause the phenomenon under observation. Furthermore, tracking the causes for the absence of a substance poses a greater challenge than tracking the causes for its presence. Specifically, the slicing analysis presented in Brodo et al. [18] abstracts away from inhibitors, while the ancestor analysis detailed in Barbuti et al. [6] can become impractical when dealing with large numbers of steps.

Positive Reaction Systems In this paper we study a particular class of RSs, called *Positive Reaction Systems* (PRSs), where reactions carry no inhibitor. The advantage is that PRSs exhibit a monotonic behavior, which can be exploited in causal analyses to improve performances and the quality of gathered information. In PRSs, both the presence and absence of entities must be explicitly recorded: if an entity is not mentioned as a cause, then it means that its status is irrelevant, not that it is assumed to be absent. Additionally, PRSs make it possible to investigate the causes for the absence of entities during the slicing, as well as for their presence.

Contribution The first contribution of this paper is the transformation from Reaction Systems to Positive Reaction Systems that exhibit a monotonic behavior while retaining equivalence to the original RS. This transformation involves several steps. First, new "negative" entities are introduced to represent the absence of inhibitors. Second, each reaction of the original RS is put in positive form: the set of inhibitors is removed and the set of reactants is joined with the negative entities that match the original inhibitors. Finally, new reactions are synthesized for negative entities, to guarantee that they are present when the corresponding original entities were absent.

This approach not only allows for the seamless application of existing tools designed for RSs to PRSs (given that they are a specialized instance of RSs), but it also enables the harnessing of PRS properties to boost tool applicability. To witness the advantages of RSs in positive form, our additional contribution consists of two applications related to cause/effect analysis.

The first application pertains to the RS slicer defined in Brodo et al. [18]. The slicer can determine the least information necessary to justify the presence of undesired entities in some state of a computation. One limitation of the slicer was its inability to take inhibitors into account, which were just disregarded. The transformation from standard RSs to PRSs addresses this limitation. Indeed, we make it possible to determine for the first time all the relevant information, both for reactants and for inhibitors, responsible for the production of the marked entities. Notably, this can be achieved just by reusing the existing tool BioReSolve presented in Brodo et al. [18]: conducting the analysis on the PRS produces more comprehensive insights.

The second application introduces a novel tool for predictor analysis, named MuMa. Its primary objective is to compute ancestor formulas more efficiently, a task where current techniques face scalability challenges. MuMa leverages two key features. First, it exploits a new abstract domain, rooted in the abstract interpretation framework from Cousot [23], Cousot and Cousot [24, 25], to overapproximate the computation of ancestor formula. Such an approximation applies to both standard and Positive RSs. However, our experiments will show that working with PRSs confers several advantages in terms of complexity. Specifically, by exploiting the monotonic reasoning in PRSs, the abstract computation of ancestor formulas will scale up.

A class of real systems suitable for modeling with RSs and that could benefit from the transformation of RSs into PRSs are gene regulatory networks (GRNs, Barbuti et al. [9]). Such networks describe how genes influence each other in order to regulate the activation of cell functionalities. GRNs are commonly modeled in terms of Boolean networks. In (Barbuti et al. [11]) a translation of Boolean networks into RSs has been defined, and in (Brodo et al. [19]) causal analysis methods have been applied to investigate properties of the gene network regulating T Cells differentiation, in the context of immune system development. The GRN investigated in (Brodo et al. [19]) was originally represented as a Boolean network consisting of 29 nodes (i.e., genes) and 97 arcs (i.e., interactions), and the corresponding RS consisted of 51 reactions over 29 entities. Analyses conducted on that model led to: (i) the characterization of initial environmental conditions leading to the activation of four genes of interest; and (ii) the identification of intermediate genes whose *activation* plays a role in the activation of the four interesting genes. Such investigations, that are based on causality analysis methods like those approached in this paper, are useful to identify possible targets for new drugs in case of diseases that involve the GRN under study. The improvements that could be obtained by translating RS models of GRNs into PRSs are the possibility to scale to much greater GRNs and to identify also genes whose *inactivation* plays a role.

To sum up, we highlight the following results:

- A new translation to remove inhibitors from RSs and recover monotonic reasoning;
- Some evidences that PRSs can enhance the spectrum of information collected from slicing;
- A new abstraction for computing (approximated) ancestor formulas in a more effective way;
- MuMa, a new tool for computing ancestor formulas based on the theoretical results in this paper.

Structure of the paper The paper is organized as follows. Section 2 introduces closed (standard) Reaction Systems. Section 3 presents the translation from standard RSs to Positive Reaction Systems. Section 4 discusses slicing techniques and highlights the advantage of utilizing the positive counterpart of a standard Reaction System, enabling the detection of inhibitors that impact the production of target objects. Section 5 defines the new abstraction that facilitates effective ancestor computation. It also covers MuMa, the novel tool based on these concepts, and provides a comparative analysis of its application to both standard and Positive RSs. Finally, Sect. 6 draws some conclusions and outlines future research avenues.

2 Reaction Systems

Reaction Systems (RSs) are a formalism born in the field of Natural Computing to qualitatively model the behavior of biochemical reactions in living cells. However, RSs are general enough to model many complex systems. Here, we give a brief introduction to RSs. For a more detailed presentation, the interested reader can refer to Brijder et al. [14]. In the

following, the term *entities* denotes generic elements (e.g., molecules, atoms, ions, ...) that may populate the system.

Let S be a (finite) set of entities. A subset $D \subseteq S$ is called a *state*. A reaction in S is a triple $r = (R, I, P)$, where $R, I, P \subseteq S$ are finite sets. Moreover, R and P are nonempty, and $R \cap I = \emptyset$. The sets R, I, P are the sets of *reactants*, *inhibitors*, and *products*, respectively. The triple (R, I, P) can be understood as follows. *All the reactants R* have to be present in the current state for the reaction to take place. The presence of *any of the inhibitors I* disables the reaction. *All the products P* are the outcome of the reaction: they are released in the next state. We denote with $rac(S)$ the set of all reactions over S.

Given a state $D \subseteq S$, the result of $r = (R, I, P) \in rac(S)$ on D, denoted by $res_r(D)$, is defined as follows:

$$res_r(D) \triangleq \begin{cases} P & \text{if } en_r(D), \\ \emptyset & \text{otherwise,} \end{cases}$$

$$en_r(D) \triangleq R \subseteq D \ \wedge \ I \cap D = \emptyset,$$

where $en_r(D)$ is called the *enabling predicate*.

Definition 1 (Reaction System)
A *Reaction System* is a pair $\mathcal{A} = (S, A)$, where S is the set of entities, and $A \subseteq rac(S)$ is a finite set of reactions over S.

Given a state $D \subseteq S$, the result of the Reaction System $\mathcal{A}$ on D, denoted $res_{\mathcal{A}}(D)$, is defined as follows:

$$res_{\mathcal{A}}(D) \triangleq \bigcup_{r \in A} res_r(D).$$

As it can be observed by the above definition, we assume that each reactant is present in a concentration level as high as needed by all the enabled reactions. This is called the threshold supply assumption and characterizes RSs as a qualitative modeling formalism.

In the general case, the behavior of an RS depends on the entities provided by the environment at each step, which is referred to as the *context*. In this work we only consider the closed version of RSs, i.e., the environment only provides some entities at the beginning, and the system evolves by applying all possible reactions. Therefore, the context consists of just an initial set $D_0 \neq \emptyset$, and starting from state D_0, the evolution of the system is deterministic, ruled by the set of reactions only. The semantics of a *closed* Reaction System can be simply defined as the *result state sequence*, $\delta = D_0, D_1, \ldots, D_n, \ldots$ where each set D_i, for $i \geq 1$, is obtained from the application of all enabled reactions of $\mathcal{A}$ to the state D_{i-1} computed at the previous step. Formally $D_i \triangleq res_{\mathcal{A}}(D_{i-1})$ for all $1 \leq i < n$, or, equivalently, as $D_i \triangleq res^{(i)}_{\mathcal{A}}(D_0)$ where for all $1 \leq i < n$, $res^{(i)}_{\mathcal{A}}$ is defined inductively as follows:

$$\begin{aligned} res^{(1)}_{\mathcal{A}}(D_0) &\triangleq res_{\mathcal{A}}(D_0), \\ res^{(i+1)}_{\mathcal{A}}(D_0) &\triangleq res_{\mathcal{A}}(res^{(i)}_{\mathcal{A}}(D_0)). \end{aligned}$$

Note that the result of any computation step does not depend on the order of application of the reactions.

To improve readability, we use a more compact notation for reactions by omitting set braces and some commas: for example, we write just $(\mathsf{ab}, \mathsf{c}, \mathsf{bc})$ instead of the more verbose $(\{\mathsf{a}, \mathsf{b}\}, \{\mathsf{c}\}, \{\mathsf{b}, \mathsf{c}\})$.

Example 1
Let us consider the RS $\mathcal{A} \triangleq (S, A)$, where $S \triangleq \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ is the set of entities, and $A \triangleq \{r_1, r_2, r_3\}$ is the set of reactions with $r_1 = (\mathsf{ab}, \mathsf{c}, \mathsf{b})$, $r_2 = (\mathsf{b}, \mathsf{c}, \mathsf{bc})$, and $r_3 = (\mathsf{c}, \mathsf{a}, \mathsf{ab})$. By setting $D_0 = \{\mathsf{a}, \mathsf{b}\}$, the result state sequence is $\tau = \{\mathsf{a}, \mathsf{b}\}, \{\mathsf{b}, \mathsf{c}\}, \{\mathsf{a}, \mathsf{b}\}, \{\mathsf{b}, \mathsf{c}\}, \ldots$

3 Positive Reaction Systems

We now want to focus on a particular kind of Reaction Systems, namely those that are formed by reactions that do not have inhibitors, i.e., where $I = \emptyset$. Such reactions are called *positive* and can simply be written as pairs (R, P): they correspond to ordinary reactions $(R, \emptyset, P)$. The idea we exploit in this paper is that any standard RS $\mathcal{A} = (S, A)$ can be encoded into an equivalent one without inhibitors. To this aim, for each entity $\mathsf{a} \in S$ we need to introduce a corresponding negative entity, let us indicate it with $\bar{\mathsf{a}}$, that explicitly represents the absence of a. Note that, following this idea, in any meaningful state there is always either a or $\bar{\mathsf{a}}$, but never both.[1] In the following, we will refer to the elements of the set S as positive entities.

Definition 2 (Negative Entities)
Let S be a set of entities. Its corresponding set of *negative entities* is $\bar{S} \triangleq \{\bar{\mathsf{a}} \mid \mathsf{a} \in S\}$.

Without loss of generality, we leave implicit that S and $\bar{S}$ are always disjoint sets. For brevity, we let $\mathbf{S} \triangleq S \cup \bar{S}$ be the set of positive and negative entities and use boldface symbols to denote its elements and its subsets.

Remark 1
Please, note that throughout this paper we will use overlining solely for symbol decoration and never for set complementation.

Definition 3 (Noncontradictory Set)
A set $\mathbf{T} \subseteq \mathbf{S}$ is *noncontradictory* if for all entities $\mathsf{a} \in S$ one has $\{\mathsf{a}, \bar{\mathsf{a}}\} \not\subseteq \mathbf{T}$.

[1] The idea of negative entities shares some similarities with the introduction of complementary places in the field of Petri nets.

As already mentioned, not all subsets of $\mathbf{S} = S \cup \bar{S}$ represent meaningful states. For example, for $S = \{\mathsf{a}, \mathsf{b}\}$, the set $\{\mathsf{a}\} \subseteq \mathbf{S}$ misses to specify whether $\{\mathsf{b}\}$ is present or not, while the set $\{\mathsf{a}, \mathsf{b}, \bar{\mathsf{b}}\} \subseteq \mathbf{S}$ is contradictory about the presence of b. Therefore, we want to consider only those states that are consistent, according to the following definition.

Definition 4 (State Consistency)
A noncontradictory state $\mathbf{D} \subseteq \mathbf{S}$ is *consistent* if, for any entity a, either $\mathsf{a} \in \mathbf{D}$ or $\bar{\mathsf{a}} \in \mathbf{D}$ holds.

We are now ready to define Positive RSs.

Definition 5 (Positive RS)
A *Positive RS* is a Reaction System $\mathcal{A}^+ = (\mathbf{S}, A)$ that satisfies the following conditions:

1. Each reaction r in A is positive, i.e., it has the form $(\mathbf{R}, \emptyset, \mathbf{P})$, and, moreover, $\mathbf{R}$ and $\mathbf{P}$ are noncontradictory;
2. The reactions must transform consistent states into consistent states: for each consistent state $\mathbf{D}$, the result set $res_{\mathcal{A}^+}(\mathbf{D})$ must be consistent.

Assuming that the initial state $\mathbf{D}_0$ is consistent, the second condition guarantees that all result states traversed by the computation will be such.

Next we show that for each (closed) standard RS $\mathcal{A} = (S, A)$ it is possible to construct a Positive RS $\mathcal{A}^+$ that can exactly mimic the behavior of $\mathcal{A}$. The Positive RS $\mathcal{A}^+$ takes the form $(\mathbf{S}, A^+)$ whose reactions in A^+ can be split in two categories: A^+_{pos} that simply embeds the original reactions A and A^+_{neg} whose reactions serve for negative entities bookkeeping. Roughly, for the first case, there will be one positive reaction $(R \cup \bar{I}, P) \in A^+$ for each original reaction $(R, I, P) \in A$, while for the second case we need to guarantee that the negative entity $\bar{\mathsf{a}}$ is produced by the reactions in A^+_{neg} if and only if no reaction in A that produces a is enabled. For this purpose, suppose we collect all reactions in A that are capable of producing a: to ensure that none of them is enabled, we must make sure that, for each one, at least one reactant is absent or at least one inhibitor is present. We formalize this intuition by introducing the *prohibiting* set for each entity a. Since prohibiting sets will be used to derive auxiliary reactions, we can restrict the attention to minimal prohibiting sets to avoid redundant reactions.

Definition 6 (Prohibiting Set)
Let $\mathcal{A} = (S, A)$ be a standard RS and $\mathsf{a} \in S$ one of its entities. A noncontradictory set $\mathbf{T} \subseteq \mathbf{S}$ is a *prohibiting set* for a if for any reaction $(R, I, P \cup \{\mathsf{a}\}) \in A$ we have that $\mathbf{T} \cap (I \cup \bar{R}) \neq \emptyset$. We denote by $Proh_{\mathcal{A}}(\mathsf{a})$ the set of all minimal (with respect to set inclusion) prohibiting sets for a.

We can transform standard RSs into positive form.

Definition 7 (Encoding RS into PRS)
Let $\mathcal{A} = (S, A)$ be a standard RS, its encoding into a Positive RS is obtained by considering $\mathcal{A}^+ \triangleq (\mathbf{S}, A^+)$ whose set of reactions $A^+ \triangleq A^+_{pos} \cup A^+_{neg}$ is defined as follows:

$$A^+_{pos} \triangleq \{(R \cup \bar{I}, P) \mid (R, I, P) \in A\},$$
$$A^+_{neg} \triangleq \textstyle\bigcup_{\mathsf{a} \in S} \{(\mathbf{T}, \bar{\mathsf{a}}) \mid \mathbf{T} \in Proh_{\mathcal{A}}(\mathsf{a})\}.$$

Lemma 1
Let $\mathcal{A}^+$ as in Definition 7. Then, $\mathcal{A}^+$ is a Positive RS.

Proof
According to Definition 5, we must prove: (1) that all reactant and product sets of reactions in A^+ are noncontradictory, and (2) that for any consistent state $\mathbf{D} \subseteq \mathbf{S}$, the result $res_{\mathcal{A}^+}(\mathbf{D})$ is also consistent.

For (1), we note that all reactant and product sets of reactions in A^+_{pos} are noncontradictory because in standard RSs we assume that reactants and inhibitors are always disjoint and their products are sets of positive entities. Similarly, all reactant sets of reactions in A^+_{neg} are noncontradictory by definition of (minimal) prohibiting set and their product sets are singletons and thus trivially noncontradictory.

For (2), let $\mathbf{D} \subseteq \mathbf{S}$ be a consistent state and let $\mathsf{a} \in S$. We have that $\mathsf{a} \in res_{\mathcal{A}^+}(\mathbf{D})$ iff $\mathsf{a} \in res_{A^+_{pos}}(\mathbf{D})$, iff there exists a reaction $(R \cup \bar{I}, P \cup \{\mathsf{a}\}) \in A^+_{pos}$ with $R \cup \bar{I} \subseteq \mathbf{D}$, iff there exists a reaction $(R, I, P \cup \{\mathsf{a}\}) \in A$ with $R \cup \bar{I} \subseteq \mathbf{D}$, iff for any set $\mathbf{T} \in Proh_{\mathcal{A}}(\mathsf{a})$ we have $\mathbf{T} \cap (I \cup \bar{R}) \neq \emptyset$ with $R \cup \bar{I} \subseteq \mathbf{D}$, iff for any reaction $(\mathbf{T}, \bar{\mathsf{a}}) \in A^+_{neg}$ we have $\mathbf{T} \not\subseteq \mathbf{D}$ (because $\mathbf{D}$ is consistent), iff $\bar{\mathsf{a}} \notin res_{A^+_{neg}}(\mathbf{D})$, iff $\bar{\mathsf{a}} \notin res_{\mathcal{A}^+}(\mathbf{D})$. □

Example 2
Considering again the RS defined in Example 1, we have that $A^+_{pos} = \{r'_1, r'_2, r'_3\}$ with $r'_1 = (\mathsf{ab\bar{c}}, \mathsf{b})$, $r'_2 = (\mathsf{b\bar{c}}, \mathsf{bc})$ and $r'_3 = (\mathsf{c\bar{a}}, \mathsf{ab})$. Moreover, since

$$Proh_{\mathcal{A}}(\mathsf{a}) = \{\{\bar{\mathsf{c}}\}, \{\mathsf{a}\}\},$$
$$Proh_{\mathcal{A}}(\mathsf{b}) = \{\{\bar{\mathsf{b}}, \bar{\mathsf{c}}\}, \{\mathsf{a}, \bar{\mathsf{b}}\}, \{\mathsf{a}, \mathsf{c}\}\}, \text{ and}$$
$$Proh_{\mathcal{A}}(\mathsf{c}) = \{\{\bar{\mathsf{b}}\}, \{\mathsf{c}\}\},$$

we have that

$$A^+_{neg} = \{\, r'_4 = (\bar{\mathsf{c}}, \bar{\mathsf{a}}), r'_5 = (\mathsf{a}, \bar{\mathsf{a}}), r'_6 = (\bar{\mathsf{b}}\bar{\mathsf{c}}, \bar{\mathsf{b}}), r'_7 = (\mathsf{a}\bar{\mathsf{b}}, \bar{\mathsf{b}}), r'_8 = (\mathsf{ac}, \bar{\mathsf{b}}), r'_9 = (\bar{\mathsf{b}}, \bar{\mathsf{c}}), r'_{10} = (\mathsf{c}, \bar{\mathsf{c}})\}.$$

While the standard RS and its positive counterpart exploit different set of entities, respectively S and $\mathbf{S} = S \cup \bar{S}$, it is possible to mark a bijective correspondence between the states of the two different formalisms: the states of the standard RS are interpreted by adding negative entities corresponding to all missing entities.

Definition 8 (State Bijection)
Let us define the following functions:

– $pos : \wp(S) \to \wp(\mathbf{S})$ such that $pos(D) \triangleq D \cup \overline{(S \setminus D)}$;
– $std : \wp(\mathbf{S}) \to \wp(S)$ such that $std(\mathbf{D}) \triangleq \mathbf{D} \cap S$.

We note that:

– for any $D \subseteq S$, $pos(D)$ is consistent by construction;
– for any $D \subseteq S$, $std(pos(D)) = D$;
– for any consistent $\mathbf{D} \subseteq \mathbf{S}$, $pos(std(\mathbf{D})) = \mathbf{D}$.

Following this intuitive correspondence, we can prove that standard RS and its corresponding positive version compute exactly the same states.

Proposition 1
Let $\mathcal{A}^+$ as in Definition 7. For any state $D \subseteq S$,

$$res_{\mathcal{A}}(D) = std(res_{\mathcal{A}^+}(pos(D))).$$

Proof
Let us take a generic entity $\mathsf{a} \in S$. We have that $\mathsf{a} \in res_{\mathcal{A}}(D)$ iff there exists a reaction $(R, I, P) \in A$ such that $\mathsf{a} \in P$, $R \subseteq D$ and $I \cap D = \emptyset$, iff there exists a reaction $(R \cup \bar{I}, P) \in A^+_{pos}$ such that $\mathsf{a} \in P$, $R \subseteq D$ and $I \subseteq S \setminus D$, iff there exists a reaction $(R \cup \bar{I}, P) \in A^+_{pos}$ such that $\mathsf{a} \in P$ and $R \cup \bar{I} \subseteq D \cup \overline{(S \setminus D)}$, iff there exists a reaction $(R \cup \bar{I}, P) \in A^+_{pos}$ such that $\mathsf{a} \in P$, $R \cup \bar{I} \subseteq pos(D)$, iff $\mathsf{a} \in res_{\mathcal{A}^+}(pos(D))$ (because, by Lemma 1, $pos(D)$ is consistent and thus $res_{\mathcal{A}^+}(pos(D))$ is also consistent), iff $\mathsf{a} \in std(res_{\mathcal{A}^+}(pos(D)))$. □

Example 3
Consider again the RS presented in Examples 1 and 2. Recall that $D_1 \triangleq res_{\mathcal{A}}(\{\mathsf{a}, \mathsf{b}\}) = \{\mathsf{b}, \mathsf{c}\}$. By drawing the correspondence discussed before between standard and positive states, we can verify that

$$res_{\mathcal{A}^+}(pos(\{\mathsf{a}, \mathsf{b}\})) = res_{\mathcal{A}^+}(\{\mathsf{a}, \mathsf{b}, \bar{\mathsf{c}}\}) = \{\bar{\mathsf{a}}, \mathsf{b}, \mathsf{c}\},$$

by applying r'_1, r'_2, r'_4 and r'_5, from which

$$std(res_{\mathcal{A}^+}(pos(\{\mathsf{a}, \mathsf{b}\}))) = std(\{\bar{\mathsf{a}}, \mathsf{b}, \mathsf{c}\}) = \{\mathsf{b}, \mathsf{c}\} = D_1.$$

Conversely, the correspondence can be drawn also by starting from any consistent state of a Positive RS.

Proposition 2
Let $\mathcal{A}^+$ as in Definition 7. For any consistent state $\mathbf{D} \subseteq \mathbf{S}$, one has

$$res_{\mathcal{A}^+}(\mathbf{D}) = pos(res_{\mathcal{A}}(std(\mathbf{D}))).$$

Proof
By Proposition 1 and since $\mathbf{D}$ is consistent, we have

$$\begin{aligned} pos(res_{\mathcal{A}}(std(\mathbf{D}))) &= pos(std(res_{\mathcal{A}^+}(pos(std(\mathbf{D}))))) \\ &= res_{\mathcal{A}^+}(\mathbf{D}). \end{aligned}$$ □

As a direct consequence of the aforementioned results, we can seamlessly extend the precise correspondence between the result sequences generated by a standard Reaction System and those produced by its positive counterpart.

Corollary 1
Let $\mathcal{A}^+$ as in Definition 7.

1. *For any initial state D_0 of $\mathcal{A}$ and $i \geq 1$,*

$$res^{(i)}_{\mathcal{A}}(D_0) = std(res^{(i)}_{\mathcal{A}^+}(pos(D_0))).$$

2. *For any consistent initial state $\mathbf{D}_0$ of $\mathcal{A}^+$ and $i \geq 1$,*

$$res^{(i)}_{\mathcal{A}^+}(\mathbf{D}_0) = pos(res^{(i)}_{\mathcal{A}}(std(\mathbf{D}_0))).$$

Proof
The proof is a trivial induction on i whose base cases correspond to Propositions 1–2 and whose inductive cases exploit the facts that $std(pos(D)) = D$ and $pos(std(\mathbf{D})) = \mathbf{D}$ (when $\mathbf{D}$ is consistent). □

4 Slicing

Computational models for complex systems can involve extensive data spaces as well as operations spanning numerous variables and datasets. Consequently, when some corrupted data is detected, the problem can be due to some error propagation, and in general it is quite difficult to pinpoint its original cause amidst the vast array of data and operations. Slicing allows us to transition from a global analysis of the system to a more focused, local one. The idea behind slicing is to identify the portion of a computation, or of a program, which is responsible for a bug by exploiting a localized analysis. This analysis discards irrelevant objects, focusing primarily on those elements that could have potentially influenced the corrupted data.

Slicing was introduced in Weiser [40] as a static technique for imperative programs, independent of any particular input of the program. Static slicing was then extended by introducing dynamic program slicing by Korel and Laski [33]. Thereafter, many slicing techniques have been developed for a full variety of programming paradigms, including, e.g., functional programming by Ochoa et al. [35], Term Rewriting by Alpuente et al. [1], functional logic programming by Alpuente et al. [2, 3], and Constraint Concurrent Programming by Falaschi et al. [30] (see Silva [39] for a survey).

4.1 Slicing for Reaction Systems

In the case of Reaction Systems, the typical error that is detected in a computation is the presence of a group of entities

that was not expected. Slicing allows us to focus the attention on the reactions that were responsible for the generation of the undesired entities and consequently on the reactants involved in their enabling. In Brodo et al. [18] we have defined a framework for dynamic (backward) slicing of RSs. By considering a finite computation trace, the analyst can mark a subset of the entities in the last state of the computation as unwanted entities. The slicing algorithm looks at the states traversed by the computation, tracking only the entities and reactions that can have played a role in the generation of the marked entities. The analyst can thus focus on a much smaller set of causes and using her expertise, she can understand the source of error, e.g., that some reaction is missing or mistyped or erroneously introduced in the system. The process of marking the entities can also be automated by monitoring the execution of the system with respect to a safety specification in modal logic. When the monitor detects a state which does not satisfy the specification, the entities which cause the failure are marked and the slicing algorithm starts from such a state.

Alpuente et al. [1, 3] have some similarity with our work in the adoption of a backward style of computation of the slicer. Moreover, Alpuente et al. [3] uses assertions to stop the computation and to start the slicing process. It is worth noting that our framework and theirs are otherwise quite different from a technical point of view. Alpuente et al. [3] considers the Maude language, and Alpuente et al. [1] is oriented towards functional computations.

One possible shortcoming of the slicing algorithm in Brodo et al. [18] is that it does not take into account the role of inhibitors. While this assumption can quite improve the performance and reduce the complexity of the algorithm, it may ultimately lead to disregarding which missing entities were crucial for the production of the target entity. However, when inhibitors are not present in reactions, no information is lost. For these reasons, we now consider the application of dynamic backward slicing on Positive Reaction Systems obtained from the transformation defined in Sect. 3.

4.2 Slicing for positive Reaction Systems

Following Brodo et al. [18], the slicing technique consists of three steps.

Enriched semantics (Step S1) The slicing process requires some extra information to be collected at run time. More precisely, (1) at each operational step, we need to highlight the reactions that have been applied; and (2) we need to determine the part of the state that is necessary to produce the marked entities in the next state. For solving (1) and (2), we consider an enriched semantics that records computation sequences.

Input: – a trace $D_0 \xrightarrow{N_1} \cdots \xrightarrow{N_m} D_m$,
– a marking $D_{sliced} \subseteq D_m$
Output: a sliced trace $D'_0 \xrightarrow{N'_1} \cdots \xrightarrow{N'_m} D_{sliced}$

```
1  begin
2    let D'_m = D_sliced
3    for i = m to 1 do
4      let D'_{i-1} = ∅ ∧ N'_i = ∅
5      for j ∈ N_i where r_j = (R_j, P_j) such that
         D'_i ∩ P_j ≠ ∅ do
6        let N'_i = N'_i ∪ {j}
7        let D'_{i-1} = D'_{i-1} ∪ R_j
8      end
9    end
10 end
```

Algorithm 1: Trace Slicer for context independent computations in PRS

Marking the state (Step S2) Let us suppose that the final configuration in a partial computation is D_m. The analyst selects a subset $D_{sliced} \subseteq D_m$ that may explain the (wrong) behavior of the program.

Trace slice (Step S3) Starting from the marked entities D_{sliced}, we define a backward slicing step, which is applied iteratively. Roughly, this step allows us to eliminate from the execution trace all the information not related to D_{sliced}. Starting from this sliced final state and proceeding backwards we can compute at each step the information which is relevant to produce the marked elements in the final state.

4.3 Trace slicer algorithm

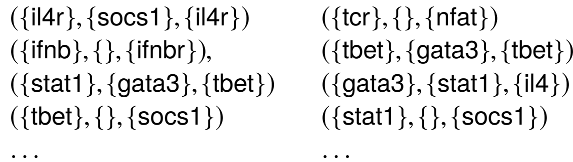
Let us explain how the slicing Algorithm 1 works. In the following we assume that the reactions are numbered consecutively by natural numbers, and denote the jth reaction in the RS by the notation r_j.

As a matter of notation, please notice that each history $D_0, \ldots, D_m$ computed in m steps defines a *trace* $D_0 \xrightarrow{N_1} \cdots \xrightarrow{N_m} D_m$, on which we perform the slicing computation, where N_i is the set of reactions applied in the ith computation step. Here each reaction is simply represented by its numeric position in the list of reactions, i.e., $N_i \triangleq \{j \mid en_{r_j}(D_{i-1})\}$ for any $i \in [1, m]$. Abusing the notation, in the following we write $r_j \in N$ whenever $j \in N$. In this version of the algorithm, we consider Positive Reaction Systems. The algorithm is completely similar to that in Brodo et al. [18], but the results that we can derive from the analysis of Positive Reaction Systems are much more informative, as we will show by our example.

In the following example we compare the application of the slicing algorithm to PRSs with respect to standard RSs.

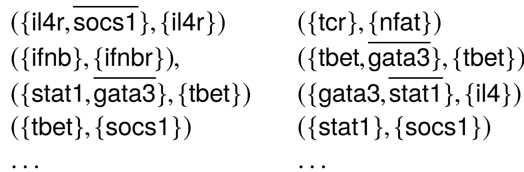
Example 4
T-cell differentiation is a widely studied biological phenomenon for which several regulatory network models have

been proposed. We consider here the model investigated in Mendoza and Xenarios [34]. This model describes a realistic regulation system that is involved in many diseases. We have coded by an RS the regulatory network governing the differentiation of CD4+ T cells presented in CellCollective Org [21]. The resulting RS is composed by 26 reactions. A few of them are below:



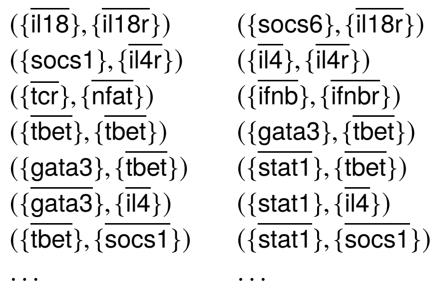
$(\{il4r\}, \{socs1\}, \{il4r\})$ $(\{tcr\}, \{\}, \{nfat\})$
$(\{ifnb\}, \{\}, \{ifnbr\})$, $(\{tbet\}, \{gata3\}, \{tbet\})$
$(\{stat1\}, \{gata3\}, \{tbet\})$ $(\{gata3\}, \{stat1\}, \{il4\})$
$(\{tbet\}, \{\}, \{socs1\})$ $(\{stat1\}, \{\}, \{socs1\})$
... ...

Then, we derive the corresponding PRS, where each reaction is composed of only two fields, reactants, and the products, since inhibitors are no longer present (i.e., the reactions in the set A^+_{pos}).



$(\{il4r, \overline{socs1}\}, \{il4r\})$ $(\{tcr\}, \{nfat\})$
$(\{ifnb\}, \{ifnbr\})$, $(\{tbet, \overline{gata3}\}, \{tbet\})$
$(\{stat1, \overline{gata3}\}, \{tbet\})$ $(\{gata3, \overline{stat1}\}, \{il4\})$
$(\{tbet\}, \{socs1\})$ $(\{stat1\}, \{socs1\})$
... ...

We recall that, for each entity in the original RS, the corresponding PRS includes some reactions for producing the negative version of the entity (i.e., the reactions in the set A^+_{neg}):



$(\{\overline{il18}\}, \{\overline{il18r}\})$ $(\{socs6\}, \{\overline{il18r}\})$
$(\{socs1\}, \{\overline{il4r}\})$ $(\{\overline{il4}\}, \{\overline{il4r}\})$
$(\{\overline{tcr}\}, \{\overline{nfat}\})$ $(\{\overline{ifnb}\}, \{\overline{ifnbr}\})$
$(\{\overline{tbet}\}, \{\overline{tbet}\})$ $(\{gata3\}, \{\overline{tbet}\})$
$(\{gata3\}, \{\overline{tbet}\})$ $(\{\overline{stat1}\}, \{\overline{tbet}\})$
$(\{\overline{gata3}\}, \{\overline{il4}\})$ $(\{stat1\}, \{\overline{il4}\})$
$(\{\overline{tbet}\}, \{\overline{socs1}\})$ $(\{\overline{stat1}\}, \{\overline{socs1}\})$
... ...

Now we consider a computation in the original RS consisting of 6 steps which leads to a state containing the entity tbet. Table 1 shows the result sequence (left column), the step number (center column), and the simplified trace produced by the slicer (right column).

The simplified computation explains which reactants are involved in the computation in this Reaction System, but does not explain the role of the inhibitors.

By applying the slicer to the PRS, we get the sliced trace in Table 2. Concerning the positive entities, the sliced trace is identical to that of the RS. However, it adds the negative entities which explain which inhibitors must be absent for tbet to appear. We derive also an interesting property. If we add to any state of the sliced trace any positive entity which does not lead to an inconsistency, the entity tbet will still be produced.

Table 1 The original RS trace and its sliced trace

Computation trace	step	sliced trace
$\{il12, il18, tcr\}$	0	$\{tcr\}$
$\{il12r, il18r, nfat\}$	1	$\{nfat\}$
$\{ifng, irak, stat4\}$	2	$\{ifng\}$
$\{ifng, ifngr\}$	3	$\{ifngr\}$
$\{ifngr, jak1\}$	4	$\{jak1\}$
$\{jak1, stat1\}$	5	$\{stat1\}$
$\{socs1, stat1, tbet\}$	6	$\{tbet\}$

Table 2 Slicing the Positive RS

step	sliced trace
0	$\{tcr, \overline{gata3}, \overline{ifnb}, \overline{fnbr}, \overline{il10r}, \overline{il4}, \overline{il4r}, \overline{jak1}, \overline{stat1}, \overline{stat6}, \overline{tbet}\}$
1	$\{nfat, \overline{gata3}, \overline{ifnbr}, \overline{il4}, \overline{il4r}, \overline{stat1}, \overline{stat3}, \overline{stat6}, \overline{tbet}\}$
2	$\{ifng, \overline{gata3}, \overline{il4}, \overline{il4r}, \overline{stat1}, \overline{stat6}, \overline{tbet}\}$
3	$\{ifngr, \overline{gata3}, \overline{il4r}, \overline{socs1}, \overline{stat6}\}$
4	$\{jak1, \overline{gata3}, \overline{stat6}\}$
5	$\{stat1, \overline{gata3}\}$
6	$\{tbet\}$

The computations have been performed using BioReSolve (Brodo et al. [18]), a tool implementing a process algebraic semantics of RSs and the slicing analysis method briefly described in this section. BioReSolve has been developed in SWI-Prolog and is publicly available with open source license.[2]

5 Predictor analysis

Dynamic slicing is aimed at detecting the causes behind some behaviors in a given computation. On the other hand, predictor analysis delves into understanding how entities can influence each other through reactions. The problem we face is to distill the most general condition on the initial state for a designated entity to be produced after n steps of executing the (closed) Reaction System. This inquiry was originally raised in the paper by Brijder et al. [13], where the authors proposed for the first time the concept of predictor. Predictors single out the sets of entities that could play a role in the production of a target object within a given number of steps. Subsequently, several different notions of predictors were introduced for a not necessarily closed Reaction System (see Barbuti et al. [6, 8]). Later works (see Barbuti et al. [7, 10]) studied the

[2] http://www.di.unipi.it/~bruni/LTSRS/.

definition of predictors for closed Reaction Systems, called ancestors in Dennunzio et al. [27]. Ancestors are initial states that surely lead to the production of the target entity.

In order to be able to determine all the alternative conditions on the initial states for them to be ancestors, the authors introduced the idea of exploiting a Boolean formula characterizing all and only the ancestors states for a given entity.

5.1 Ancestors

Ancestors are sets leading to the production of a given entity in the required number of steps.

Definition 9 (*n*th Ancestors for a)
Let $\mathcal{A} = (S, A)$ be a Reaction System. Given $\mathsf{a} \in S$, a set D_0 is an nth *ancestor* of a if $\mathsf{a} \in res^{(n)}_{\mathcal{A}}(D_0)$.

The same concepts can be naturally extended to sets of entities.

Definition 10 (*n*th Ancestors for $\{\mathsf{a}_1, \ldots, \mathsf{a}_m\}$)
Let $\mathcal{A} = (S, A)$ be a Reaction System. Given $\{\mathsf{a}_1, \ldots, \mathsf{a}_m\} \subseteq S$, a set D_0 is an nth *ancestor* of $\{\mathsf{a}_1, \ldots, \mathsf{a}_m\}$ if $\forall i : 1 \le i \le m$, $\mathsf{a}_i \in res^{(n)}_{\mathcal{A}}(D_0)$.

Ancestors can be succinctly represented by Boolean formulas on the alphabet S. Formally, letting $\mathcal{L} = S \cup \neg S$ be the set of all positive and negative literals, we consider the set $\mathcal{F}_S$ of propositional formulas on S, defined as the least set such that $\mathcal{L} \cup \{true, false\} \subseteq \mathcal{F}_S$ and $f_1 \vee f_2, f_1 \wedge f_2 \in \mathcal{F}_S$ if $f_1, f_2 \in \mathcal{F}_S$. The propositional formulas $f \in \mathcal{F}_S$ are interpreted with respect to sets of entities. Correspondingly, the satisfaction relation is written $C \models f$, where $C \subseteq S$ and $f \in \mathcal{F}_S$. For example, the propositional symbol a is satisfied by any set C such that $\mathsf{a} \in C$. The complete definition of the satisfaction relation is as follows.

Definition 11 (Satisfaction)
Let $C \subseteq S$ for a given set of entities S. Given a propositional formula $f \in \mathcal{F}_S$, the *satisfaction relation* $C \models f$ is inductively defined as follows:

$$\begin{aligned}
&C \models true,\\
&C \models \mathsf{a} \text{ iff } \mathsf{a} \in C,\\
&C \models \neg\mathsf{a} \text{ iff } \mathsf{a} \notin C,\\
&C \models f_1 \vee f_2 \text{ iff } C \models f_1 \text{ or } C \models f_2,\\
&C \models f_1 \wedge f_2 \text{ iff } C \models f_1 \text{ and } C \models f_2.
\end{aligned}$$

Note that, for simplicity but without loss of generality, we restrict the use of negation to literals: the negation of a formula $f \in \mathcal{F}_S$ is representable in $\mathcal{F}_S$ using De Morgan laws. We aim to define a formula characterizing all the initial sets D_0 that lead to the production of a given product $\mathsf{a} \in S$ after exactly n steps. Note that the formula for an nth ancestor of a set of entities $\{\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_m\} \subseteq S$ can be obtained by combining in conjunction all the nth ancestor formulas for each a_i with $i \in \{1, \ldots, m\}$.

Our goal is to characterize all the initial sets leading to the production of entity a in the required number of steps by devising a propositional formula f, called the nth ancestor formula, that all such initial sets have to satisfy (according to the satisfaction relation defined in Definition 11).

Definition 12 (*n*th Ancestor Formula)
Let $\mathcal{A} = (S, A)$ be a Reaction System and $\mathsf{a} \in S$ be an entity of interest. We say that a formula $f \in \mathcal{F}_S$ *is an nth ancestor formula* of a iff the following holds:

$$D_0 \models f \Longleftrightarrow res^{(n)}_{\mathcal{A}}(D_0) \models \mathsf{a}.$$

The causes of an entity in a Reaction System are defined by a propositional formula on the set of entities S. First of all, we define the *applicability predicate* of a reaction r as a propositional logic formula on S describing the requirements for the applicability of reaction r. That is, the fact that all reagents must be present and inhibitors must be absent. These requirements are expressed by the conjunction of all atomic formulas representing the reactants and the negations of all literals representing the inhibitors of the considered reaction.

Definition 13 (Applicability Predicate)
Let $r = (R, I, P)$ be a reaction on entities S. The *applicability predicate* of r, denoted by $ap(r)$, is defined as follows:

$$ap(r) \triangleq \left(\bigwedge_{\mathsf{a}_r \in R} \mathsf{a}_r \right) \wedge \left(\bigwedge_{\mathsf{a}_i \in I} \neg\mathsf{a}_i \right).$$

The *causal predicate* of a given entity a is a propositional formula on S representing the conditions for the production of entity a in one step, that is, we require that at least one reaction having a as a product has to be enabled.

Definition 14 (Causal Predicate)
Let $\mathcal{A} = (S, A)$ be a Reaction System. Given the entity $\mathsf{a} \in S$, the *causal predicate* of a in $\mathcal{A}$, denoted by $cause(\mathsf{a}, \mathcal{A})$ (or $cause(\mathsf{a})$, when $\mathcal{A}$ is clear from the context), is defined as follows:[3]

$$cause(\mathsf{a}) \triangleq \bigvee_{\{(R,I,P) \in A \mid \mathsf{a} \in P\}} ap\left((R, I, P)\right).$$

Moreover, we define the *negation of the causal predicate* of a, called $nocause(\mathsf{a})$, as the DNF formula equivalent to $\neg cause(\mathsf{a})$.

[3] Note that, as a special case, $cause(\mathsf{a}) = false$ if there is no $(R, I, P) \in A$ such that $\mathsf{a} \in P$.

Note that *nocause*(a) is computed (once for all) from *cause*(a) by putting the negations next to the atomic entities using De Morgan's laws and then by nesting all conjunctions within the disjunctions using the distributive law.

In order to devise (effectively compute) the nth ancestor formula we define an operator *Anc*.

Definition 15 (Ancestor Operator)
Let $\mathcal{A} = (S, A)$ be a Reaction System and $\mathsf{a} \in S$ be an entity of interest. We define a function $Anc : S \times \mathbb{N} \to \mathcal{F}_S$ as follows:

$$Anc(\mathsf{a}, n) \triangleq Anc^{(n)}(\mathsf{a}),$$

where the auxiliary function $Anc : \mathcal{F}_S \to \mathcal{F}_S$ is recursively defined as follows:

$$\begin{aligned} Anc(\mathsf{b}) &\triangleq cause(\mathsf{b}),\\ Anc(\neg\mathsf{b}) &\triangleq nocause(\mathsf{b}),\\ Anc(f_1 \vee f_2) &\triangleq Anc(f_1) \vee Anc(f_2),\\ Anc(f_1 \wedge f_2) &\triangleq Anc(f_1) \wedge Anc(f_2),\\ Anc(true) &\triangleq true,\\ Anc(false) &\triangleq false. \end{aligned}$$

We can prove that operator *Anc* computes the nth ancestor formula.

Theorem 1 (Barbuti et al. [6])
*Let $\mathcal{A} = (S, A)$ be a Reaction System. For any entity $\mathsf{a} \in S$, $Anc(\mathsf{a}, n)$ is an n*th ancestor formula *of* a.

The ancestor formula can always be thought as expressing the different alternative conditions to be satisfied by an initial state. To this aim, we can always transform the ancestor formula into disjunctive normal form. This form allows us to compactly represent the formula using a set of sets of literals such as $\{L_1, \ldots, L_n\}$, where the literals in the same set L_i are interpreted as being in conjunction, while the entire formula is obtained as a disjunction of the different L_i. We call σ the map from formulas in disjunctive normal form to set of sets of literals:

$$\sigma(\bigvee_{i \in I} \bigwedge_{j \in I_i} l_{ij}) \triangleq \{L_i \mid i \in I \wedge L_i = \{l_{ij} \mid j \in I_i\}\}.$$

Example 5
For example, the formula $f = (\mathsf{a} \wedge \mathsf{b}) \vee (\mathsf{a} \wedge \mathsf{c} \wedge \neg\mathsf{b})$ can be represented by the set

$$F = \sigma(f) = \{\{\mathsf{a}, \mathsf{b}\}, \{\mathsf{a}, \mathsf{c}, \neg\mathsf{b}\}\}.$$

5.2 Approximated ancestor computation

While the definition of a systematic way to compute ancestor formulas has undoubtedly a theoretical relevance, the complexity of the ancestor computation depends on the number of reactants and inhibitors that are present in each reaction. Moreover, the complexity of the computation increases with the number of steps. In Barbuti et al. [7, 10] it was shown that answering the question if a starting state for the production of a given entity (in the required number of steps) exists or not was, in general, intractable. Indeed, such problem could be solved by simplifying the computed ancestor formula that required, in theory, putting the formula in disjunctive normal form and then finding the minimal simplification. Even if we do not want to simplify the formula to ease its verification, the size and the complexity of the formula makes the calculation feasible only for a limited number of steps.

For these reasons, we need to introduce some kind of approximation. Instead of computing the exact ancestor formula, we take into account just the set of entities that must be present in any ancestor and those that may be present in some ancestor: even if they provide just some partial information, they are computationally less expensive to calculate. The correspondence between the exact ancestor and the approximated version can then be formalized in terms of abstract interpretation as the relation between two domains, the concrete domain with exact information and the abstract domain where the approximation takes place.

Letting $\mathcal{L} = S \cup \neg S$ be the set of all positive and negative literals, our concrete domain is a suitable subset of $\wp(\wp(\mathcal{L}))$, which accounts for disjunctive propositional formulas, as discussed at the end of the previous section (see Example 5). Since the different conjunctions in the ancestor formula describe different initial states, the order on the concrete domain we consider is simply set inclusion of the subsets representing the conjunction.

Definition 16 (Concrete domain)
We let the concrete domain be defined as

$$C \triangleq \{\{L_i\}_i \in \wp(\wp(\mathcal{L})) \mid \forall i. \forall \mathsf{a}. \{\mathsf{a}, \neg\mathsf{a}\} \not\subseteq L_i\}$$

ordered by inclusion.

Note that the domain C does not correspond to that of propositional formulas ordered by implication as it admits noncomparable representations of equivalent propositional formulas. For example, while a is logically equivalent to $(\mathsf{a} \wedge \mathsf{b}) \vee (\mathsf{a} \wedge \neg\mathsf{b})$ the elements $\{\{\mathsf{a}\}\}$ and $\{\{\mathsf{a}, \mathsf{b}\}, \{\mathsf{a}, \neg\mathsf{b}\}\}$ are not comparable. Roughly, this is due to the different role played by literals in the two cases: in a conjunctive formula the absence of a literal means that its truth value is not important, while in the domain C the absence of a literal means that we have no information about its presence or absence.

The idea of the approximation is to consider just two sets of literals: the first element, called `must`, collects all the literals common to all sets and the second, called `maybe`,

lists the literals appearing in at least one set (but not present in $\mathtt{must}$). Hence, the approximation belongs to the abstract domain $\wp(\mathcal{L}) \times \wp(\mathcal{L})$.

Definition 17 (Abstract domain)
We let the abstract domain be defined as

$$\mathcal{M} \triangleq \{(\mathtt{MU},\mathtt{MA}) \in \wp(\mathcal{L}) \times \wp(\mathcal{L}) \mid \forall \mathsf{a}.((\{\mathsf{a},\neg\mathsf{a}\} \cap \mathtt{MU} \neq \emptyset \Rightarrow \{\mathsf{a},\neg\mathsf{a}\} \cap \mathtt{MA} = \emptyset) \wedge \{\mathsf{a},\neg\mathsf{a}\} \not\subseteq \mathtt{MU})\}.$$

The corresponding order on such an abstract domain is defined as follows: given $(\mathtt{MU}_1,\mathtt{MA}_1),(\mathtt{MU}_2,\mathtt{MA}_2) \in \mathcal{M}$, we let $(\mathtt{MU}_1,\mathtt{MA}_1) \sqsubseteq (\mathtt{MU}_2,\mathtt{MA}_2)$ iff $\mathtt{MU}_2 \subseteq \mathtt{MU}_1$ and $\mathtt{MU}_1 \cup \mathtt{MA}_1 \subseteq \mathtt{MU}_2 \cup \mathtt{MA}_2$.

Following an abstract interpretation approach as in Cousot [23], Cousot and Cousot [24, 25], the concrete and abstract domains can be related by an abstraction and concretization functions that form a Galois connection.

Definition 18 (Abstraction Function)
The abstraction function $\alpha : \mathcal{C} \to \mathcal{M}$ is defined as follows: for any $F = \{L_1,\ldots,L_n\} \in \mathcal{C}$,

$$\alpha(F) = (\mathtt{must}(F),\mathtt{maybe}(F)),$$

where $\mathtt{must} : \mathcal{C} \to \wp(\mathcal{L})$ and $\mathtt{maybe} : \mathcal{C} \to \wp(\mathcal{L})$ are two auxiliary functions such that:

$$\mathtt{must}(F) \triangleq \textstyle\bigcap_{i=1}^{n} L_i,$$
$$\mathtt{maybe}(F) \triangleq \textstyle\bigcup_{i=1}^{n} L_i \setminus \mathtt{must}(F).$$

Example 6
The sets $\mathtt{must}(F) = \{\mathsf{a}\}$ and $\mathtt{maybe}(F) = \{\mathsf{c},\mathsf{b},\neg\mathsf{b}\}$ approximate the set F of Example 5.

The following technical lemma will be exploited in the proofs of Propositions 3 and 4.

Lemma 2
For any $F = \{L_1,\ldots,L_n\} \in \mathcal{C}$,

$$\mathtt{must}(F) \cup \mathtt{maybe}(F) = \textstyle\bigcup_{i=1}^{n} L_i\,.$$

Proof
Trivially,

$$\begin{aligned}\mathtt{must}(F) \cup \mathtt{maybe}(F) &= \mathtt{must}(F) \cup (\textstyle\bigcup_{i=1}^{n} L_i \setminus \mathtt{must}(F))\\ &= \mathtt{must}(F) \cup (\textstyle\bigcup_{i=1}^{n} L_i)\\ &= (\textstyle\bigcap_{i=1}^{n} L_i) \cup (\textstyle\bigcup_{i=1}^{n} L_i)\\ &= \textstyle\bigcup_{i=1}^{n} L_i\,. \qquad \square\end{aligned}$$

The concretization function will transform the pair of sets into a set of sets of literals that will include also the original ancestor formula. To construct such sets, we need to include all entities in the must set together with an element of the power set of the maybe set.

Definition 19 (Concretization Function)
The concretization function $\gamma : \mathcal{M} \to \mathcal{C}$ is defined as follows: for any $(\mathtt{MU},\mathtt{MA}) \in \mathcal{M}$,

$$\gamma(\mathtt{MU},\mathtt{MA}) = \{L \mid \mathtt{MU} \subseteq L \subseteq (\mathtt{MU} \cup \mathtt{MA}) \wedge \forall \mathsf{a}.\{\mathsf{a},\neg\mathsf{a}\} \not\subseteq L\}.$$

Proposition 3
Functions α and γ form a Galois insertion.

Proof
It is immediate to check that the maps α and γ are monotone. To show that they form a Galois connection, we need to prove that, for any $F \in \mathcal{C}$ and any $(\mathtt{MU},\mathtt{MA}) \in \mathcal{M}$, one has

$$F \subseteq \gamma(\mathtt{MU},\mathtt{MA}) \Leftrightarrow \alpha(F) \sqsubseteq (\mathtt{MU},\mathtt{MA}).$$

Let $F = \{L_1,\ldots,L_n\}$. We have

$$\begin{aligned}F \subseteq \gamma(\mathtt{MU},\mathtt{MA}) &\Leftrightarrow \forall i.L_i \in \gamma(\mathtt{MU},\mathtt{MA})\\ &\Leftrightarrow \forall i.\mathtt{MU} \subseteq L_i \subseteq \mathtt{MU} \cup \mathtt{MA}\\ &\Leftrightarrow (\mathtt{MU} \subseteq \textstyle\bigcap_{i=1}^{n} L_i) \wedge (\textstyle\bigcup_{i=1}^{n} L_i \subseteq \mathtt{MU} \cup \mathtt{MA})\\ &\Leftrightarrow (\mathtt{MU} \subseteq \mathtt{must}(F))\\ &\qquad \wedge (\mathtt{must}(F) \cup \mathtt{maybe}(F)) \subseteq \mathtt{MU} \cup \mathtt{MA})\\ &\Leftrightarrow (\mathtt{must}(F),\mathtt{maybe}(F)) \sqsubseteq (\mathtt{MU},\mathtt{MA})\\ &\Leftrightarrow \alpha(F) \sqsubseteq (\mathtt{MU},\mathtt{MA}).\end{aligned}$$

Finally, we note that, for any $(\mathtt{MU},\mathtt{MA}) \in \mathcal{M}$, we have:

- $\mathtt{must}(\gamma(\mathtt{MU},\mathtt{MA})) = \mathtt{MU}$, because, by definition of γ, $\mathtt{MU}$ is the smallest set in $\gamma(\mathtt{MU},\mathtt{MA})$ and it is included in any other set $L \in \gamma(\mathtt{MU},\mathtt{MA})$;
- and $\mathtt{maybe}(\gamma(\mathtt{MU},\mathtt{MA})) = \mathtt{MA}$. In fact, to see that

$$\textstyle\bigcup_{L\in\gamma(\mathtt{MU},\mathtt{MA})} L \setminus \mathtt{MU} \subseteq \mathtt{MA},$$

  we note that any $L \in \gamma(\mathtt{MU},\mathtt{MA})$ is included in $\mathtt{MU} \cup \mathtt{MA}$ and thus if $l \in L \setminus \mathtt{MU}$ then $l \in \mathtt{MA}$. Vice versa, if $l \in \mathtt{MA}$, let $L = \mathtt{MU} \cup \{l\}$:[4] since $\mathtt{MU} \subseteq L \subseteq \mathtt{MU} \cup \mathtt{MA}$ then $L \in \gamma(\mathtt{MU},\mathtt{MA})$ and therefore $l \in \bigcup_{L\in\gamma(\mathtt{MU},\mathtt{MA})} L$. Since $\mathtt{MU}$ and $\mathtt{MA}$ are disjoint (by definition of $\mathcal{M}$) and $l \in \mathtt{MA}$, we have $l \in \bigcup_{L\in\gamma(\mathtt{MU},\mathtt{MA})} L \setminus \mathtt{MU}$.

Thus, we conclude $\alpha(\gamma(\mathtt{MU},\mathtt{MA})) = (\mathtt{MU},\mathtt{MA})$. □

[4] We note that $\forall \mathsf{a}.\{\mathsf{a},\neg\mathsf{a}\} \not\subseteq L$ because $(\mathtt{MU},\mathtt{MA}) \in \mathcal{M}$.

Example 7
We can now apply the concretization function to the abstraction obtained in Example 6:

$$\gamma(\{\mathsf{a}\},\{\mathsf{b},\neg\mathsf{b},\mathsf{c}\}):\ \{\ \{\mathsf{a}\},\ \underline{\{\mathsf{a},\mathsf{b}\}},\ \{\mathsf{a},\neg\mathsf{b}\},\ \{\mathsf{a},\mathsf{c}\},\ \underline{\{\mathsf{a},\neg\mathsf{b},\mathsf{c}\}},\ \{\mathsf{a},\mathsf{b},\mathsf{c}\}\ \}$$

Note that we compute an overapproximation of the original set of sets. Indeed, the sets belonging to the concrete set from which we started in Example 5 are underlined here.

Each conjunct of the disjunctive form of the ancestor formula expresses a possible initial state that leads to the production of the desired entity in the required number of steps. The disjunctive form collects all of them. The kind of approximation we propose devises a superset of such states, that includes all conjuncts together with some redundant sets and some other sets that will not produce the required entity.

However, our approximation allows us to define an abstract version of the ancestor operator *Anc*, defined in the previous section. Such an abstract operator takes a pair $(\mathtt{MU},\mathtt{MA})$ and transforms it to a new pair of sets.

Definition 20 (MuMa Operator)
Let $\mathcal{A}=(S,A)$ be a Reaction System and let $\mathsf{a}\in S$. The Must/Maybe ancestor operator $\mathtt{MuMa}: S\times\mathbb{N}\to\mathcal{M}$ is defined as follows:

$$\mathtt{MuMa}(\mathsf{a},n)=\mathtt{MuMax}^{(n)}(\{\mathsf{a}\},\emptyset),$$

where $\mathtt{MuMax}:\mathcal{M}\to\mathcal{M}$ is an auxiliary function defined inductively as follows:

$$\mathtt{MuMax}(\mathtt{MU},\mathtt{MA})\triangleq(mu_{\mathrm{x}}(\mathtt{MU}),ma_{\mathrm{x}}(\mathtt{MU},\mathtt{MA})),$$

where

$$\begin{aligned} mu_{\mathrm{x}}(\mathtt{MU}) &\triangleq \textstyle\bigcup_{l\in\mathtt{MU}}\mathtt{must}(F_l),\\ ma_{\mathrm{x}}(\mathtt{MU},\mathtt{MA}) &\triangleq \bigcup_{l\in\mathtt{MA}\cup\mathtt{MU}}(\mathtt{must}(F_l)\cup\mathtt{maybe}(F_l))\setminus mu_{\mathrm{x}}(\mathtt{MU}),\\ F_l &\triangleq \sigma(Anc(l)). \end{aligned}$$

The definition of the abstract operator MuMa can be better understood by reading the above equations bottom-up. If l is a positive literal a, then F_l is just (the image via σ of) the causal predicate of a. If l is a negative literal a, then we need instead to take the image of the negated causal predicate. Note that $\alpha(F_l)=(\mathtt{must}(F_l),\mathtt{maybe}(F_l))$ collects the must–maybe causes for a single literal in a single step.

The function mu_{x} takes a set MU of must causes, for each literal in MU computes its must set for a single step, and then returns the union of all such must sets. The function ma_{x} deals with maybe sets: it takes a set MU of must causes and a set MA of maybe causes, then for each literal $l\in\mathtt{MU}\cup\mathtt{MA}$ computes the union $\mathtt{must}(F_l)\cup\mathtt{maybe}(F_l)$ of its must and maybe causes and collects them all, finally to return the set of maybe causes it removes the must causes $mu_{\mathrm{x}}(\mathtt{MU})$.

Finally, the function $\mathtt{MuMax}^{(n)}$ just iterates the calculation of must–maybe causes for n steps.

The following example clarifies the above procedure.

Example 8
Let $\mathcal{A}=\{\mathsf{a},\mathsf{b},\mathsf{c}\}$ be the Reaction System with the following reactions:

$$\begin{aligned} r_1&=(\{\mathsf{a}\},\{\mathsf{b}\},\{\mathsf{b}\}), & r_2&=(\{\mathsf{b},\mathsf{c}\},\emptyset,\{\mathsf{a}\}),\\ r_3&=(\{\mathsf{a},\mathsf{c}\},\emptyset,\{\mathsf{b}\}), & r_4&=(\{\mathsf{b}\},\{\mathsf{c}\},\{\mathsf{c}\}).\\ r_5&=(\{\mathsf{a},\mathsf{b}\},\emptyset,\{\mathsf{c}\}). \end{aligned}$$

The must and maybe sets for all the literals are given below:

$$\begin{aligned} \alpha(F_{\mathsf{a}})&=(\{\mathsf{b},\mathsf{c}\},\emptyset), & \alpha(F_{\neg\mathsf{a}})&=(\emptyset,\{\neg\mathsf{b},\neg\mathsf{c}\}),\\ \alpha(F_{\mathsf{b}})&=(\{\mathsf{a}\},\{\mathsf{c},\neg\mathsf{b}\}), & \alpha(F_{\neg\mathsf{b}})&=(\emptyset,\{\neg\mathsf{a},\mathsf{b},\neg\mathsf{c}\}),\\ \alpha(F_{\mathsf{c}})&=(\{\mathsf{b}\},\{\mathsf{a},\neg\mathsf{c}\}), & \alpha(F_{\neg\mathsf{c}})&=(\emptyset,\{\neg\mathsf{a},\neg\mathsf{b},\mathsf{c}\}). \end{aligned}$$

Assume we want to approximate the ancestor for the entity c in 3 steps. We need to compute $\mathtt{MuMa}(\mathsf{c},3)$:

$$\begin{aligned} \mathtt{MuMa}(\mathsf{c},3)&=\mathtt{MuMax}^{(3)}(\{\mathsf{c}\},\{\})\\ &=\mathtt{MuMax}^{(2)}(\{\mathsf{b}\},\{\mathsf{a},\neg\mathsf{c}\})\\ &=\mathtt{MuMax}(\{\mathsf{a}\},\\ &\qquad(\{\mathsf{a}\}\cup\{\mathsf{c},\neg\mathsf{b}\}\cup\{\mathsf{b},\mathsf{c}\}\\ &\qquad\cup\{\neg\mathsf{a},\neg\mathsf{b},\mathsf{c}\})\setminus\{\mathsf{a}\})\\ &=\mathtt{MuMax}(\{\mathsf{a}\},\{\neg\mathsf{a},\mathsf{b},\neg\mathsf{b},\mathsf{c}\})\\ &=(\{\mathsf{b},\mathsf{c}\},\{\mathsf{a},\neg\mathsf{a},\neg\mathsf{b},\neg\mathsf{c}\}). \end{aligned}$$

Note that the (exact) ancestor of c in 3 steps is

$$\sigma(Anc(\mathsf{c},3))=\{\{\mathsf{a},\mathsf{b},\mathsf{c}\}\},$$

expressing the fact that after 3 steps we can reach a state where entity c is present if and only if we start in an initial state D_0 containing a, b, and c.

Observe that if one were to concretize $\mathtt{MuMa}(\mathsf{c},3)$, then one would obtain an overapproximation of $\mathtt{Anc}(\mathsf{c},3)$, namely

$$\sigma(Anc(\mathsf{c},3))\subseteq\gamma(\mathtt{MuMa}(\mathsf{c},3))=\{\{\mathsf{a},\mathsf{b},\mathsf{c}\},\{\neg\mathsf{a},\mathsf{b},\mathsf{c}\}\}.$$

Since not all the states belonging to the application of operator MuMa lead to the production of the entity of interest, once we have computed the abstraction, we need to check which initial states in the concretization actually lead to the desired outcome. This can be done by running the Reaction System (for the given number of steps) starting with the different initial states identified by the abstraction. Note that

the presence of inhibitors causes the reasoning to be non-monotonic: to detect all the states that lead to the desired outcome, we need to check *all* the initial states satisfying the abstraction, because we cannot rely on minimal sets only.

Example 9
Consider again the RS in Example 8. Assume we are interested in the production of $\mathtt{b}$ in 1 step. The corresponding ancestor formula is

$$Anc(\mathtt{b},1) = (\mathtt{a} \wedge \neg\mathtt{b}) \vee (\mathtt{a} \wedge \mathtt{c})$$

while using the approximation we compute

$$\mathtt{MuMa}(\mathtt{b},1) = \mathtt{MuMax}(\{\mathtt{b}\},\emptyset) = (\{\mathtt{a}\},\{\neg\mathtt{b},\mathtt{c}\}).$$

If we are interested in all the initial states that lead to the production of $\mathtt{b}$ in 1 step, we have to check the sets $\{\mathtt{a}\}$, $\{\mathtt{a},\mathtt{c}\}$, but also $\{\mathtt{a},\mathtt{b}\}$ and $\{\mathtt{a},\mathtt{c},\mathtt{b}\}$, because the fact that some sets satisfying $\mathtt{MuMa}(\mathtt{b},1)$ (in this case $\{\mathtt{a}\}$ and $\{\mathtt{a},\mathtt{c}\}$) lead to the production of $\{\mathtt{b}\}$ in 1 step does not guarantee that all their supersets will do the same. Indeed, $\{\mathtt{a},\mathtt{b},\mathtt{c}\}$ does, while $\{\mathtt{a},\mathtt{b}\}$ does not.

We conclude this section by showing that the $\mathtt{MuMax}$ operator is a correct approximation of the ancestor function Anc.

Proposition 4 (Soundness)
For any f in disjunctive normal form,

$$\alpha(\sigma(Anc(f))) \sqsubseteq \mathtt{MuMax}(\alpha(\sigma(f))).$$

Proof
First note that for any $F \in C$, we have

$$F = \{(U \cup B_1),\ldots,(U \cup B_k)\},$$

where $U = \mathtt{must}(F)$ and $\bigcup_{i=1}^{k} B_i = \mathtt{maybe}(F)$: in the following, we tacitly assume this standard representation for the elements of C. In particular, we let:

$$\sigma(f) = \{(M \cup L_1),\ldots,(M \cup L_n)\},$$
$$F_f \triangleq \sigma(Anc(f)) = \{(M^f \cup L_1^f),\ldots,(M^f \cup L_{m_f}^f)\}.$$

Therefore,

$$\alpha(\sigma(f)) = (M, \textstyle\bigcup_{i=1}^{n} L_i),$$
$$\alpha(F_f) = (M^f, \textstyle\bigcup_{j=1}^{m_f} L_j^f).$$

Moreover, we let $(\mathtt{Mu},\mathtt{Ma}) = \mathtt{MuMax}(\alpha(\sigma(f)))$, i.e.,

$$\mathtt{Mu} = mu_{\mathrm{x}}(M) = \textstyle\bigcup_{p\in M} \mathtt{must}(\sigma(Anc(p))) = \bigcup_{p \in M} M^p,$$
$$\mathtt{Ma} = ma_{\mathrm{x}}(M, \textstyle\bigcup_{i=1}^{n} L_i).$$

To prove the claim, by definition of $\sqsubseteq$, we need to show that:

1. $\mathtt{Mu} \subseteq M_f$; and
2. $M^f \cup \bigcup_{j=1}^{m_f} L_j^f \subseteq \mathtt{Mu} \cup \mathtt{Ma}$.

For the former, taking a generic literal $l \in \mathtt{Mu}$, we need to show that $l \in M_f$. Since $l \in \mathtt{Mu}$, it must be the case that $l \in M^p$ for some $p \in M$. Since $p \in M$, then $p \in M \cup L_i$ for all $i \in [1,n]$. Therefore $M^p \subseteq M^f$, because the ancestors of p are taken into account in any disjunct of f, and we conclude $l \in M^f$.

For the latter, we prove the stronger property

$$M^f \cup \textstyle\bigcup_{j=1}^{m_f} L_j^f = \mathtt{Mu} \cup \mathtt{Ma}.$$

Note that, by definition,

$$M^f \cup \textstyle\bigcup_{j=1}^{m_f} L_j^f = \bigcup_{L \in F_f} L.$$

Exploiting Lemma 2, and using the shorthand $l \mathrel{\widehat{\in}} f$ in place of the more verbose $l \in \bigcup_{L\in\sigma(f)} L$ to mean that l is a literal that appears in the formula f, we can show that

$$\begin{aligned}\mathtt{Mu} \cup \mathtt{Ma} &= \textstyle\bigcup_{l \widehat{\in} f} \mathtt{must}(\sigma(Anc(l))) \cup \mathtt{maybe}(\sigma(Anc(l)))\\ &= \textstyle\bigcup_{l \widehat{\in} f} \bigcup_{L \in F_l} L,\end{aligned}$$

where, as usual, $F_l \triangleq \sigma(Anc(l))$. Finally, the equality

$$\textstyle\bigcup_{L \in F_f} L = \bigcup_{l \widehat{\in} f} \bigcup_{L \in F_l} L$$

can be proved to hold for any formula f by structural induction on f. The base cases, where f is a Boolean value or $f = l$ for some literal l are trivial. For the inductive cases, if $f = f_1 \vee f_2$, we have

$$\begin{aligned}\textstyle\bigcup_{L \in F_f} L &= \textstyle\bigcup_{L \in \sigma(Anc(f_1) \vee Anc(f_2))} L\\ &= \textstyle\bigcup_{L \in \sigma(Anc(f_1)) \cup \sigma(Anc(f_2))} L\\ &= (\textstyle\bigcup_{L \in F_{f_1}} L) \cup (\bigcup_{L \in F_{f_2}} L)\\ &= (\textstyle\bigcup_{l \widehat{\in} f_1} \bigcup_{L \in F_l} L) \cup (\bigcup_{l \widehat{\in} f_2} \bigcup_{L \in F_l} L)\\ &= \textstyle\bigcup_{l \widehat{\in} (f_1 \vee f_2)} \bigcup_{L \in F_l} L.\end{aligned}$$

Finally, if $f = f_1 \wedge f_2$, we have

$$\begin{aligned}\textstyle\bigcup_{L \in F_f} L &= \textstyle\bigcup_{L \in \sigma(Anc(f_1) \wedge Anc(f_2))} L\\ &= \textstyle\bigcup_{L_1 \in \sigma(Anc(f_1)), L_2 \in \sigma(Anc(f_2))} L_1 \cup L_2\\ &= (\textstyle\bigcup_{L_1 \in F_{f_1}} L_1) \cup (\bigcup_{L_2 \in F_{f_2}} L_2)\\ &= (\textstyle\bigcup_{l \widehat{\in} f_1} \bigcup_{L_1 \in F_l} L_1) \cup (\bigcup_{l \widehat{\in} f_2} \bigcup_{L_2 \in F_l} L_2)\\ &= \textstyle\bigcup_{l \widehat{\in} (f_1 \wedge f_2)} \bigcup_{L \in F_l} L.\end{aligned}$$

□

5.3 Applying the abstraction to PRSs

Of course, we can reason on Positive Reaction Systems in the same way we did for ordinary Reaction Systems. Therefore, the abstraction proposed in the previous section can be applied also to Positive Reaction Systems.

However, when reactions are without inhibitors, an important advantage in the verification stage comes out because we can exploit monotonicity. In particular, when checking for each formula in the concretization whether this formula leads to the production of the entity of interest, we can reason on minimal sets only.

Proposition 5
Given a propositional formula $f \in \mathcal{F}_S$ that does not contain any negative literals. For any superset $C' \supseteq C$, we have that $C \models f \implies C' \models f$.

Proof
We proceed, by structural induction on f.

The base cases $f = true$ and $f = false$ are trivial.

If $f = \mathsf{a}$ for some literal $\mathsf{a} \in S$ and $C \models f$, then it means that $\mathsf{a} \in C \subseteq C'$ and thus $C' \models \mathsf{a}$.

The case $f = \neg\mathsf{a}$ is not admissible, as f does not contain any negative literals by hypothesis.

If $f = f_1 \vee f_2$ for some formulas f_1 and f_2 without negative literals and $C \models f$, then it means that $C \models f_i$ for some $i \in \{1, 2\}$. Then, by inductive hypothesis, $C' \models f_i$ and thus $C' \models f$.

If $f = f_1 \wedge f_2$ for some formulas f_1 and f_2 without negative literals and $C \models f$, then it means that $C \models f_1$ and $C \models f_2$. Then, by inductive hypothesis, $C' \models f_1$ and $C' \models f_2$, thus $C' \models f$. □

This property assures us that in the case of Positive Reaction Systems, once we have determined a set in the concretization of the MuMa abstraction, we can avoid checking all the supersets of such a set because we are sure they will lead to the desired outcome as well.

Example 10
Let us revisit the RS presented in Examples 8–9 by encoding it into a Positive one. We have:

$$A^+_{pos} = \{\ r'_1 = (\mathsf{a}\bar{\mathsf{b}}, \mathsf{b}), r'_2 = r_2, r'_3 = r_3, r'_4 = (\mathsf{b}\bar{\mathsf{c}}, \mathsf{c}), r'_5 = r_5\ \},$$

$$A^+_{neg} = \{\ r'_6 = (\bar{\mathsf{b}}, \bar{\mathsf{a}}), r'_7 = (\bar{\mathsf{c}}, \bar{\mathsf{a}}), r'_8 = (\bar{\mathsf{a}}, \bar{\mathsf{b}}), r'_9 = (\mathsf{b}\bar{\mathsf{c}}, \bar{\mathsf{b}}), r'_{10} = (\bar{\mathsf{b}}, \bar{\mathsf{c}}), r'_{11} = (\bar{\mathsf{a}}\mathsf{c}, \bar{\mathsf{c}})\ \}.$$

As before, assume we are interested in the production of b in 1 step. Our abstraction describing all the initial states leading to b in 1 step in A^+ is

$$\begin{aligned} \mathtt{MuMa}(\mathsf{b}, 1) &= \mathtt{MuMax}^1(\{\mathsf{b}\}, \{\}) \\ &= \mathtt{MuMax}^0(\{\mathsf{a}\}, \{\bar{\mathsf{b}}, \mathsf{c}\}) \\ &= (\{\mathsf{a}\}, \{\bar{\mathsf{b}}, \mathsf{c}\}). \end{aligned}$$

Note that this time the set $\{\mathsf{a}\}$ does not lead to the production of b in 1 step. The minimal sets that produce b are $\{\mathsf{a}, \bar{\mathsf{b}}\}$ and $\{\mathsf{a}, \mathsf{c}\}$. Once we have determined them, we are guaranteed that also any of their supersets will work as initial state. As, for example, $\{\mathsf{a}, \bar{\mathsf{b}}, \bar{\mathsf{c}}\}$.

5.4 MuMa tool

We developed a tool to compute the ancestor computation based on the MuMa operator following the procedure presented in the previous two sections. The tool is written in Java and available on GitHub,[5] together with installation and usage instructions. In particular, we used SwingX framework for the graphical user interface and Guava and BitSet libraries respectively to implement the initial logical computation and to improve performances based on set computations. The tool works for RSs in general, but it is designed to take some advantages in the case of PRSs.

MuMa requires to load a Reaction System. This can be done incrementally, by adding one reaction at a time, or all at once by uploading a file containing all reactions together, written in a suitable plain text format. Once the entire Reaction System is loaded, we can ask for the computation of the ancestor of a single entity or any conjunction of entities for an arbitrary number of steps. The tool will then perform the computation of the approximation based on the MuMa operator as defined in Definition 20. After such a computation, the results can be visualized and the two final stages that consist of the concretization of the abstraction and the checking of the real solutions can be executed on request. At the end, a table will show all the verified formulas that are legal initial states for the required outcome, grouped by set inclusion. A sample screenshot of MuMa graphical interface is shown in Fig. 1.

We studied the performance of our tool with standard and Positive Reaction Systems. Indeed, one possible problem of the proposed abstraction resides in the verification phase. When dealing with standard Reaction Systems, the explosion in the size of the configurations that need to be checked can be a serious bottleneck. The complexity of this operation depends on the number of formulas generated by concretization, the number of initial states satisfying the formulas, and the number of steps performed. We tested our tool on a machine with 8-core AMD Ryzen7 5800H processor, with 16 processing threads and 16 GB of RAM. To challenge the tool, we used a carefully crafted Reaction System consisting of 20 entities and 33 different reactions (also available on the GitHub page). We asked for the ancestors of a given entity in 4 steps. In the analyzed case, despite the fact that the resulting formulas had a very simple structure, the verification was very difficult because it required checking 2^{16} different configurations. For these reasons, we also implemented a

[5] https://github.com/valquake/MuMa-Predictor.

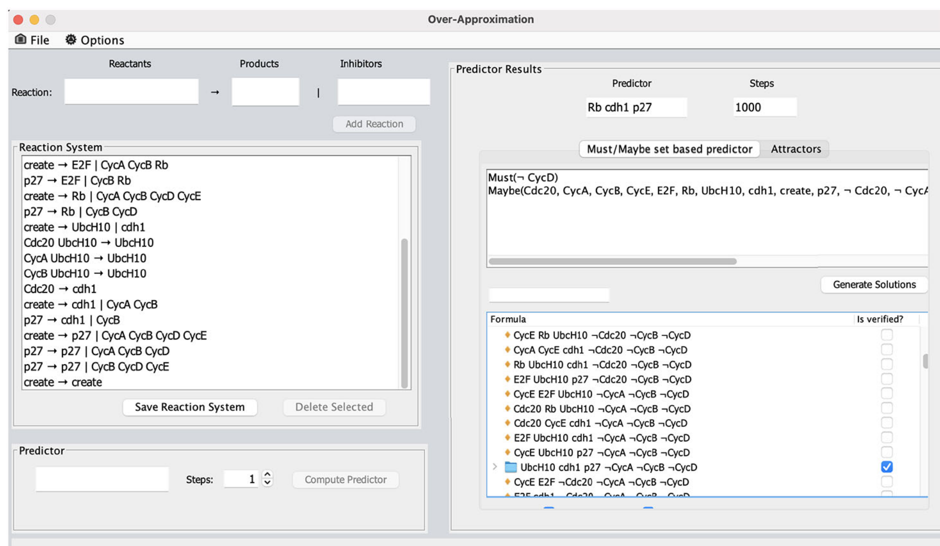


Fig. 1 MuMa tool graphical interface

Table 3 Comparing MuMa performances on standard and Positive Reaction Systems

Steps	RS	Positive RS	Ancestors
1	35.64 s	4 ms	1
2	130.03 s	4 ms	5
3	undef	17 ms	255
4	undef	31 ms	3071

translation into PRSs and systematically compared the efficiency of the computations of the ancestors for standard and (equivalent) Positive RSs. Clearly, also the translation of an ordinary reaction system into a positive one has a cost but this is required just once, at the beginning.

Table 3 shows the comparison between the performance of our method in case of a standard and (equivalent) Positive Reaction System. In the first column we reported the number of steps. In the second column there is the time required to detect all the ancestors starting with the standard Reaction System. In the third column there is the time required to detect all the ancestors starting with the equivalent Positive Reaction System, and finally in the fourth column we reported the number of ancestors found.

As we can see, thanks to the translation into Positive Reaction Systems, ancestor problems that seem to be intractable starting with a standard Reaction System become easier to solve when starting with the corresponding Positive Reaction System. This is because, thanks to Proposition 5 in the case of Positive Reaction Systems, we can exploit its monotonic behavior to avoid checking all the sets in the concretization.

However, also the concretization can be expensive because we must deal with the powerset of the maybe set, an operation known to be exponential in the number of elements in the set. Of course, this number strictly depends on the different reactants and inhibitors that constitute each reaction considered for a single product.

In order to further improve our tool, we implemented a multithread version of the concretization and verification steps. Indeed, we parallelized the computation of the power set. In more details, the presence or absence of all literals that form a reaction is encoded using binary sequences. These sequences are generated and processed in parallel with a fork/join mechanism, in which only noncontradictory sequences are kept and collected. The final list will contain a filtered power set. We also gave an additional parallel solution for the verification procedure. The list of possible formulas is split between a fixed number of thread in a thread pool. Each thread will work on a list partition and will insert the result for all the formulas in the task in a shared list, that will be displayed in the user interface.

We evaluated the multithread benefits by using two classical metrics, namely scalability and speedup. Scalability is the ratio between the throughput of the workload on a multiprocessor and the throughput on a comparable uniprocessor. The speedup is the ratio between the performance obtained by computing the task with enhancement and the one without enhancement.

The Reaction System used as a benchmark has 23 reactions and 10 entities. The reactions were chosen in such a way that the abstraction was imprecise; in fact, the result is a maybe set containing all positive and negative entities. Therefore, the resulting concretization is a list containing all possible noncontradictory formulas. In this way, the parallel solution can express its full capabilities. Once again, we compared the computation starting with standard and corresponding Positive RSs.

From Fig. 2, we can notice that results are similar. This is due to the fact that the execution time of sequential code is

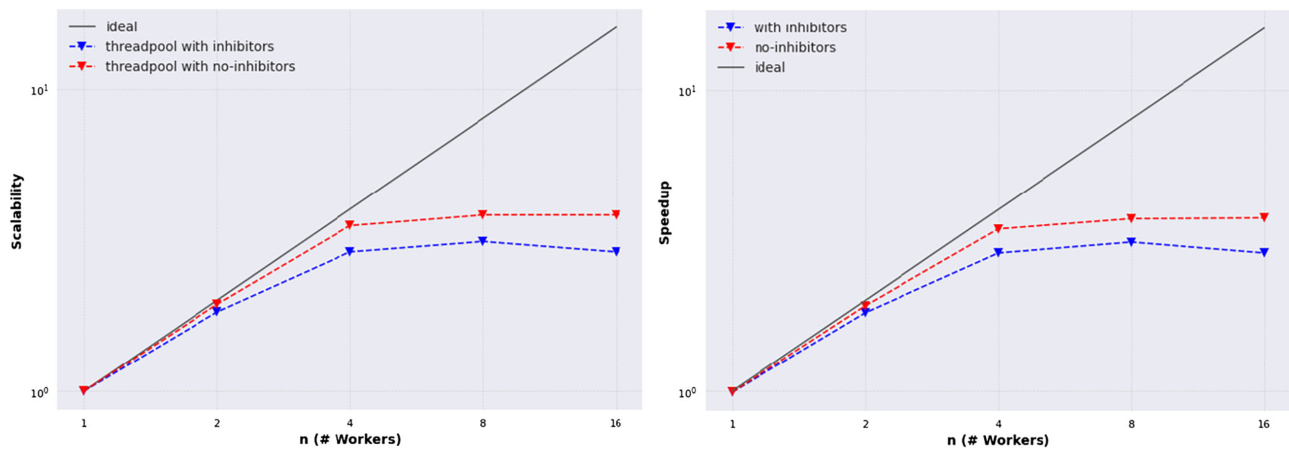


Fig. 2 Scalability vs Speedup for a 10,000 steps execution

close to the execution time of a single thread. The tendency observed is that even when the number of threads is greater than 4 the threadpool behaves like when it worked with 4 threads, for this example. Moreover, also the experimentation with multithreads gives a better performance when Positive Reaction Systems are considered than with standard Reaction Systems.

6 Conclusions and future work

This paper centers on formal cause/effect analyses for Reaction Systems (RSs), a computational formalism inspired by biochemical reactions in living cells. The presence of inhibitors in the reactions of an RS gives rise to a computationally nonmonotonic behavior, which makes the identification of the causes of the production of an entity more difficult. This may cause an exponential blow up. To enhance scalability and to expose the causal dependencies from inhibitors, we have introduced a transformation from standard RSs to Positive RSs. This transformation preserves the behavior of the original RS while replacing all inhibitors with new entities, referred to as 'negative' entities, representing the absence of their original counterparts. At the cost of introducing several new reactions for bookkeeping negative entities, the transformed system exhibits a monotonic behavior.

We have then considered two complementary analyses. A slicing framework which allows us to focus on the causes of a given set of entities in a specific computation, and a predictor analysis that aims at determining a Boolean formula, called ancestor formula, which expresses the exact causes of the introduction of a set of entities in a given number of steps. Both methods benefit from the transformation to Positive RSs. Indeed, the slicer in Brodo et al. [18] did not previously consider the role of inhibitors. In this paper we have shown that we can now explain what are the minimal entities which must be present or absent (inhibitors) in a computation for determining the production of a target set of entities. Regarding the ancestor formula computation, we have introduced an overapproximation of propositional formulas via abstract interpretation, making the computation tractable and scalable. Finally, we have presented MuMa, a tool based on such abstraction, and have conducted several experiments with our prototype. The results confirm that, thanks to positive reactions and the must/maybe abstraction, the performances of the tool are quite improved and can scale up.

The cause/effect analysis methods we have considered in this paper are closely related with responsibility analysis approaches described by Deng and Cousot [26]. In that context, methods are classified into *dependency analysis*, *counterfactual causality*, and *actual causality*. The analysis approaches we considered in this paper belong to the dependency analysis class, namely they allow identifying entities that contribute to some effect without measuring how decisive they are. However, it is usually possible to conceive experiments in which the application of these methods to different variants of the RS under study provides information typical of counterfactual or actual causality. For instance, in (Brodo et al. [19]), where we studied a gene regulatory network, we repeated slicing analysis to an exhaustive set of environmental conditions in order to determine how decisive is each gene in the achievement of the behavior of interest (similarly to actual causality). Moreover, the analysis can be executed on a set of modified RSs in which system components are selectively knocked-out to determine which of them are nondecisive (as in counterfactual causality).

Several research directions are left to future work. First, Positive RSs make it feasible to develop a theory of forward causal analysis, where each entity carries its history of causes during the computation, so that when an erroneous state is reached the set of causes is immediately available for inspection.

Second, we plan to combine slicing with predictor analysis. Essentially, the idea is that the slicer should not merely return a plain set of causes, but rather a potentially disjunctive ancestor formula. Disjuncts would provide a more detailed account of the various alternatives ultimately responsible for the production of a specific entity. For example, given the (positive) reactions (ab, d) and (ac, d), the short result sequence abc, d, and the undesired production of d, the current slicer would just return the entire set of causes abc, while a more accurate ancestor formula would be $\mathsf{ab} \vee \mathsf{ac}$.

Third, the theory of abstract interpretation could be further exploited in MuMa by devising abstract domains that are more precise with respect to must/maybe sets, possibly exploiting some ad hoc knowledge about the particular RS under inspection to exploit correlations between entities in the design of the abstract domain.

Fourth, we plan to study interactive RSs, where the environment (called "context") can provide a set of positive entities to the RS at each computation step.

Funding Open access funding provided by Università degli Studi di Sassari within the CRUI-CARE Agreement. This research has been partially supported by the Italian MUR PRIN 2022 project "MEDICA" (2022RNTYWZ), by the Next Generation EU programme project PNRR ECS00000017 – "THE - Tuscany Health Ecosystem" – Spoke 3 – CUP B63C22000680007, and Spoke 6 – CUP I53C22000780001, by the Italian MUR PRIN PNRR 2022 project "DELICE" *Decentralized Ledgers in Circular Economy* (P20223T2MF), by the Italian MUR PRIN 2022 PNRR project *Resource Awareness in Programming: Algebra, Rewriting, and Analysis* (P2022HXNSC), and by the INdAM–GNCS project CUP_E53C22001930001.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Alpuente, M., Ballis, D., Espert, J., Romero, D.: Backward trace slicing for rewriting logic theories. In: Proc. of CADE'11. Lecture Notes in Computer Science, vol. 6803, pp. 34–48. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-22438-6_5
2. Alpuente, M., Ballis, D., Frechina, F., Romero, D.: Using conditional trace slicing for improving Maude programs. Sci. Comput. Program. **80**, 385–415 (2014). https://doi.org/10.1016/j.scico.2013.09.018
3. Alpuente, M., Ballis, D., Frechina, F., Sapiña, J.: Debugging Maude programs via runtime assertion checking and trace slicing. J. Log. Algebraic Methods Program. **85**, 707–736 (2016). https://doi.org/10.1016/j.jlamp.2016.03.001
4. Azimi, S.: Steady states of constrained Reaction Systems. Theor. Comput. Sci. **701**(C), 20–26 (2017). https://doi.org/10.1016/j.tcs.2017.03.047
5. Azimi, S., Iancu, B., Petre, I.: Reaction System models for the heat shock response. Fundam. Inform. **131**(3–4), 299–312 (2014). https://doi.org/10.3233/FI-2014-1016
6. Barbuti, R., Gori, R., Levi, F., Milazzo, P.: Investigating dynamic causalities in Reaction Systems. Theor. Comput. Sci. **623**, 114–145 (2016). https://doi.org/10.1016/j.tcs.2015.11.041
7. Barbuti, R., Bernasconi, A., Gori, R., Milazzo, P.: Computing preimages and ancestors in Reaction Systems. In: Proc. of TPNC 2018. Lecture Notes in Computer Science, vol. 11324, pp. 23–35. Springer, Berlin (2018). https://doi.org/10.1007/978-3-030-04070-3_2
8. Barbuti, R., Gori, R., Levi, F., Milazzo, P.: Generalized contexts for Reaction Systems: definition and study of dynamic causalities. Acta Inform. **55**(3), 227–267 (2018). https://doi.org/10.1007/s00236-017-0296-3
9. Barbuti, R., Gori, R., Milazzo, P., Nasti, L.: A survey of gene regulatory networks modelling methods: from differential equations, to Boolean and qualitative bioinspired models. J. Membr. Comput. **2**, 207–226 (2020). https://doi.org/10.1007/s41965-020-00046-y
10. Barbuti, R., Bernasconi, A., Gori, R., Milazzo, P.: Characterization and computation of ancestors in Reaction Systems. Soft Comput. **25**(3), 1683–1698 (2021). https://doi.org/10.1007/s00500-020-05300-0
11. Barbuti, R., Gori, R., Milazzo, P.: Encoding Boolean networks into reaction systems for investigating causal dependencies in gene regulation. Theor. Comput. Sci. **881**, 3–24 (2021). https://doi.org/10.1016/j.tcs.2020.07.031
12. Bodei, C., Gori, R., Levi, F.: Causal static analysis for Brane Calculi. Theor. Comput. Sci. **587**, 73–103 (2015). https://doi.org/10.1016/j.tcs.2015.03.014
13. Brijder, R., Ehrenfeucht, A., Rozenberg, G.: A note on causalities in Reaction Systems. ECEASST 30 (2010) https://doi.org/10.14279/tuj.eceasst.30.429
14. Brijder, R., Ehrenfeucht, A., Main, M.G., Rozenberg, G.: A tour of Reaction Systems. Int. J. Found. Comput. Sci. **22**(7), 1499–1517 (2011). https://doi.org/10.1142/S0129054111008842
15. Brodo, L., Bruni, R., Falaschi, M.: Enhancing reaction systems: a process algebraic approach. In: Alvim, M., Chatzikokolakis, K., Olarte, C., Valencia, F. (eds.) The Art of Modelling Computational Systems. LNCS, vol. 11760, pp. 68–85. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-31175-9_5
16. Brodo, L., Bruni, R., Falaschi, M.: A logical and graphical framework for Reaction Systems. Theor. Comput. Sci. **875**, 1–27 (2021). https://doi.org/10.1016/j.tcs.2021.03.024
17. Brodo, L., Bruni, R., Falaschi, M., Gori, R., Levi, F., Milazzo, P.: Exploiting modularity of SOS semantics to define quantitative extensions of Reaction Systems. In: Proc. of TPNC 2021. Lecture Notes in Computer Science, vol. 13082, pp. 15–32. Springer, Berlin (2021). https://doi.org/10.1007/978-3-030-90425-8_2
18. Brodo, L., Bruni, R., Falaschi, M.: Dynamic slicing of Reaction Systems based on assertions and monitors. In: Proc. of PADL 2023. Lecture Notes in Computer Science, vol. 13880, pp. 107–124. Springer, Berlin (2023). https://doi.org/10.1007/978-3-031-24841-2_8
19. Brodo, L., Bruni, R., Falaschi, M., Gori, R., Milazzo, P.: Attractor and slicing analysis of a T Cell differentiation model based on reaction systems Proceedings of DataMod 2023 Springer, Berlin, LNCS. (In press)
20. Busi, N.: Causality in membrane systems. In: Proc. of WMC 2007, pp. 160–171 (2007). https://doi.org/10.1007/978-3-540-77312-2_10
21. CellCollective Org: CD4+ T cell differentiation model webpage on the CellCollective platform (2018). https://cellcollective.org/

#module/2901:1/t-cell-differentiation/1. Last accessed: 18 Sept. 2023
22. Corolli, L., Maj, C., Marinia, F., Besozzi, D., Mauri, G.: An excursion in Reaction Systems: from computer science to biology. Theor. Comput. Sci. **454**, 95–108 (2012). https://doi.org/10.1016/j.tcs.2012.04.003
23. Cousot, P.: Principles of Abstract Interpretation. MIT Press, Cambridge (2021)
24. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of POPL 1977, pp. 238–252. ACM Press, New York (1977). https://doi.org/10.1145/512950.512973
25. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proc. of POPL 1979, pp. 269–282. ACM Press, New York (1979). https://doi.org/10.1145/567752.567778
26. Deng, C., Cousot, P.: The systematic design of responsibility analysis by abstract interpretation. ACM Trans. Program. Lang. Syst. **44**(1), 3:1–3:90 (2022). https://doi.org/10.1145/3484938
27. Dennunzio, A., Formenti, E., Manzoni, L., Porreca, A.E.: Ancestors, descendants, and gardens of Eden in Reaction Systems. Theor. Comput. Sci. **608**, 16–26 (2015). https://doi.org/10.1016/j.tcs.2015.05.046
28. Dennunzio, A., Formenti, E., Manzoni, L., Porreca, A.E.: Preimage problems for Reaction Systems. In: Dediu, A., Formenti, E., Martín-Vide, C., Truthe, B. (eds.) Proc. of LATA 2015. Lecture Notes in Computer Science, vol. 8977, pp. 537–548. Springer, Berlin (2015). https://doi.org/10.1007/978-3-319-15579-1_42
29. Ehrenfeucht, A., Rozenberg, G.: Reaction Systems. Fundam. Inform. **75**(1–4), 263–280 (2007). http://content.iospress.com/articles/fundamenta-informaticae/fi75-1-4-15
30. Falaschi, M., Gabbrielli, M., Olarte, C., Palamidessi, C.: Dynamic slicing for concurrent constraint languages. Fundam. Inform. **177**(3–4), 331–357 (2020). https://doi.org/10.3233/FI-2020-1992
31. Formenti, E., Manzoni, L., Porreca, A.E.: Fixed points and attractors of Reaction Systems. In: Proc. of CiE 2014. Lecture Notes in Computer Science, vol. 8493, pp. 194–203. Springer, Berlin (2014). https://doi.org/10.1007/978-3-319-08019-2_20
32. Gori, R., Levi, F.: Abstract interpretation based verification of temporal properties for BioAmbients. Inf. Comput. **208**(8), 869–921 (2010). https://doi.org/10.1016/j.ic.2010.03.004
33. Korel, B., Laski, J.: Dynamic program slicing. Inf. Process. Lett. **29**(3), 155–163 (1988). https://doi.org/10.1016/0020-0190(88)90054-3
34. Mendoza, L., Xenarios, I.: A method for the generation of standardized qualitative dynamical systems of regulatory networks. Theor. Biol. Med. Model. **3**(13) (2006). https://doi.org/10.1186/1742-4682-3-13
35. Ochoa, C., Silva, J., Vidal, G.: Dynamic slicing of lazy functional programs based on redex trails. High.-Order Symb. Comput. **21**(1–2), 147–192 (2008). https://doi.org/10.1007/s10990-008-9023-7
36. Okubo, F., Yokomori, T.: The computational capability of chemical reaction automata. Nat. Comput. **15**(2), 215–224 (2016). https://doi.org/10.1007/s11047-015-9504-7
37. Salomaa, A.: Functional constructions between Reaction Systems and propositional logic. Int. J. Found. Comput. Sci. **24**(1), 147–160 (2013). https://doi.org/10.1142/S0129054113500044
38. Salomaa, A.: Minimal and almost minimal Reaction Systems. Nat. Comput. **12**(3), 369–376 (2013). https://doi.org/10.1007/s11047-013-9372-y
39. Silva, J.: A vocabulary of program slicing-based techniques. ACM Comput. Surv. **44**(3), 12:1–12:41 (2012). https://doi.org/10.1145/2187671.2187674
40. Weiser, M.: Program slicing. IEEE Trans. Softw. Eng. **10**(4), 352–357 (1984). https://doi.org/10.1109/TSE.1984.5010248

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.