Natural Computing
https://doi.org/10.1007/s11047-025-10045-6

Experimenting with Reaction Systems using Graph Transformation and GROOVE

Roberto Bruni[1] · **Arend Rensink**[2]

Accepted: 27 July 2025

Abstract
We explore the capabilities of GROOVE, a state-of-the-art toolset based on graph transformation systems, to perform different kinds of analyses of Reaction Systems, ranging from reachability and causal analysis to model checking. Our results are encouraging, as in the presence of large state spaces GROOVE improves the time required for both reachability and causal analyses by an order of magnitude, compared to other available tools. From the point of view of GROOVE, the implementation of Reaction Systems provided some interesting insights on the most convenient way to model certain computational requirements through negative and nested application conditions.

Keywords Reaction Systems · Graph Transformation · GROOVE · Causal analysis · Formal methods

1 Introduction

We explore the capabilities of a state-of-the-art toolset based on graph transformation to perform reachability, causal analysis and verification of complex systems modelled using Reaction Systems.

Reaction systems (RSs) (Ehrenfeucht and Rozenberg 2007) are a computational model inspired by the functioning of biochemical reactions within living cells. RSs focus on the interaction of entities through a set of reactions. Each reaction relies on some reactants, inhibitors, and products to mimic two fundamental mechanisms found in nature: facilitation and inhibition. At each time instant, the next state of the system is determined by the products of all enabled reactions plus some additional entities that are possibly provided by the environment. Unlike traditional models of concurrency, like Petri nets, the theory of RSs is based on three principles: *no permanency*: any entity vanishes unless it is sustained by a reaction; *no competition*: an entity is either available for all reactions, or it is not available at all; and *no counting*: the exact concentration level of available entities is ignored. Moreover, due to the use of inhibitors, RS can exhibit non-monotonic behaviour, in the sense that what can be done with fewer resources is not necessarily replicable with more resources.

While *closed* RSs evolve in isolation, *interactive processes* are dealt with by providing suitable environments that provide a sequence of stimuli at each step: these are sets of entities that can be used to trigger or inhibit some reactions. A common example involves using contexts to analyse how drug administration affects organisms that are modelled as sets of reactions.

Since their introduction, RSs have been successfully applied to the analysis of complex systems in many different fields (Azimi et al. 2014; Corolli et al. 2012; Azimi 2017; Okubo and Yokomori 2016; Ehrenfeucht et al. 2010, 2011). Recent applications concerned, e.g., with the efficacy of medical treatments for comorbidities and with the selection of the best environment to achieve some desired phenomena (Bowles et al. 2024; Brodo et al. 2025), led to experimenting with environments that exhibit nondeterministic and recursive behaviour. As a consequence, performing reachability and causal analysis requires the exploration of large state spaces, for which the prototype tool BioResolve (Brodo et al. 2021) struggled in terms of memory consumption and response time.

✉ Arend Rensink
arend.rensink@utwente.nl

Roberto Bruni
roberto.bruni@unipi.it

[1] CS Department, University of Pisa, Largo B. Pontecorvo, 3, 56127 Pisa, Italy

[2] CS Department, University of Twente, Hallenweg 19, 7522 Enschede, Netherlands

Graph Transformation (GT) (Ehrig et al. 2006; Heckel and Taentzer 2020) is a modelling technique that is widely applicable in problem domains where the objects of study have an inherent graphical structure, and the task at hand is to study their properties and evolution. Besides the graphs themselves, the core concept is that of a (transformation) *rule* capturing a particular change to such a graph. Rules can be used, for instance, to describe the change of a system over time, but can also be instrumental in composing and decomposing graphs and so exposing structural properties. Since RSs can be derived from Boolean network models and visualised themselves as suitable networks of reactions, it is quite natural trying to embed them within the GT framework to take advantage of well established analysis techniques.[1]

Importantly, from a practical point of view, there are a number of (academic) tools supporting the use of GT. The research described here crucially relies on GROOVE (Ghamarian et al. 2012), one of the most prominent tools in this area, which was designed precisely to enable GT-based system analysis of the kind described above. The features of GROOVE that are essential for the purpose of this research are:

1. Nested (i.e., quantified) rules, which capture simultaneous changes in all neighbourhoods that satisfy certain application conditions, rather than only locally in one such neighbourhood at a time;
2. Complete exploration of the set of reachable states (under the given rules) using various strategies;
3. Model checking functionalities that can be used to validate previous findings as well to explore and support the study of new behavioural and structural properties.

The main research question that motivated our study is: *how can GT help in addressing the analysis of Reaction Systems?* To this aim, we encode a given RS as a single graph, upon which a small number of (fixed) rules can simulate the correct semantics.

The core rule describes the simultaneous firing of all **Reaction**s of which no inhibitors and all reactants are present; the firing results in the presence of all products. Simultaneously, all currently present **Entity**s are removed. In GROOVE syntax, the core rule is drawn as in Fig. 1. To parse this, note that

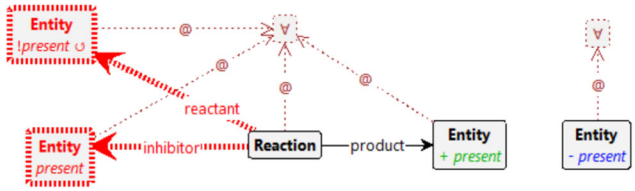


Fig. 1 Rule for reactions firing

(in the GROOVE notation) the red structure must be absent for the rule to apply; moreover, green labels are added and blue ones deleted upon rule application. The *present* flag signals whether an **Entity** is considered to be currently present; hence, creating or deleting that flag comes down to creating or deleting the **Entity**. The ∀-nodes impose the desired quantification, causing a single application of this rule to model the firing of all enabled **Reaction**s, even if there are thousands of them. Similarly for the simultaneous deletion of all **Entity**s.

Our model also supports environments that inject **Entity**s in a controlled manner. This is achieved by encoding the context specification in the initial graph and exploiting a second predefined rule, not shown here (see Fig. 7). A configuration is reachable if it can be constructed by the alternating application of both rules: first the context produces a set of stimuli (possibly upon inspection of the current state, but also nondeterministically or a combination of both), and then all enabled reactions are executed.

After a brief account of RSs (Sect. 2.1) and GROOVE (Sect. 2.2), we present the overall rationale, blueprint and key features of our approach in Sect. 4 through a simple vending machine example (Sect. 3). The subsequent experimentation in Sect. 5 is concerned with revisiting existing RS case studies to assess how GROOVE can enhance their analysis. In particular, we tackle the comorbidity treatment scenario from Bowles et al. (2024) in Sect. 5.1, the protein signaling networks analysis from Ballis et al. (2024) in Sect. 5.2, and the T cell differentiation study from Brodo et al. (2025) in Sect. 5.3. Our results are encouraging: GROOVE is capable to explore large RSs by using roughly one tenth of the time compared to BioResolve for both reachability and causal analyses. More precisely, we are able to find a trace towards unwanted patterns (if they exist) among hundreds of thousands of reachable configurations using different heuristics; then, we can also prune the trace to extract a graphical representation of the causal history of the entities. Using the built-in model checker it is also possible to confirm the results of previous studies, as well as proving new facts. A comparison with related tools is drawn in Sect. 6. Concluding remarks and future directions are discussed in Sect. 7.

[1] It should be noted that there is another, quite unrelated, way in which the concepts of Graph Transformation and Reaction Systems may be combined, namely by extending the latter from pure entities (which are analogous to graph nodes) to entities with relations between them (which can be encoded through graph edges). In Kreowski and Rozenberg (2019) this has inspired a new methodology for Graph Transformation, there called *graph surfing*.

2 Background

2.1 RSs with guarded contexts

First, we briefly account for the classical set theoretic definition of Reaction Systems (RSs) (Ehrenfeucht and Rozenberg 2007). Then, we focus on their process algebraic version (Brodo et al. 2021) and its further extension with guarded contexts in Bowles et al. (2024) that are supported by the RS analysis framework BioResolve.[2]

RS basics. A Reaction System is a pair $A = (S, A)$, where S is the finite set of *entities*, and A is a finite set of *reactions* of the form $a = (R, I, P)$, with $R, I, P \subseteq S$ and $R \cap I = \varnothing$. The sets R, I, P are the sets of *reactants*, *inhibitors*, and *products*, respectively. Without loss of generality, we admit the use of empty sets as reactants, inhibitors, and products. Given the current state $W \subseteq S$, a reaction $a = (R, I, P)$ is enabled in W if all its reactants are present (i.e., $R \subseteq W$) and all its inhibitors absent (i.e., $W \cap I = \varnothing$). The *result* of the reaction a on the current state W is P if a is enabled, and $\varnothing$ otherwise. The result of all reactions A on the current state W, is the union of the results of all reactions. The no-permanency principle of RSs dictates that entities disappear if not sustained by some reaction. Thus, the current state $W = D \cup C$ is determined by the result D of all reactions on the previous state, together with some additional entities C that can be provided by the *context* at each step.

Process algebraic RSs and guarded contexts. Inspired by Plotkin's Structural Operational Semantics approach (Plotkin 2004) and process algebras such as CCS (Milner 1980), the key features of the process algebraic version of RSs are compositionality and the ability to account for a quite general notion of context (guarded, nondeterministic, recursive) using a friendly syntax. This way, we derive a Labelled Transition System (LTS) semantics for RSs by means of inductive inference rules, where LTS states are terms of an algebra, each transition defines a computation step of the RS and its label records the entities involved in that step.

RS processes are defined by the grammar below:

$$\mathsf{P} ::= [\mathsf{M}]$$
$$\mathsf{M} ::= (R, I, P) \mid D \mid \mathsf{K} \mid \mathsf{M}|\mathsf{M}$$
$$\mathsf{K} ::= \mathbf{0} \mid (R, I, C).\mathsf{K} \mid \mathsf{K} + \mathsf{K} \mid X$$

where R, I, P, C, and D are sets of entities (with $R \cap I = \varnothing$) and X is a context identifier drawn from a family of (possibly recursive) definitions $\Delta \triangleq \{X_j = \mathsf{K}_j\}_{j \in J}$, called the *environment*.

Roughly, a RS process P embeds a *mixture* process M obtained as the parallel composition of some reactions (R, I, P), some available entities D (if any), and some *context* process K. A context process K is either: the nil context $\mathbf{0}$ that stops the computation; the guarded context $(R, I, C).\mathsf{K}$ that makes the entities C available to the reactions if the reactants R are present and the inhibitors I are absent, and then will behave as K at the next step; the non-deterministic choice $\mathsf{K}_1 + \mathsf{K}_2$ that can behave as either K_1 or K_2; the context identifier X that behaves as K for $X = \mathsf{K} \in \Delta$. We write $C.\mathsf{K}$ as a shorthand for the trivially guarded process $(\varnothing, \varnothing, C).\mathsf{K}$ and we assume the recursive context $\mathsf{Emp} = \varnothing.\mathsf{Emp}$ is always defined.

We say that P and P' are structurally equivalent, written $\mathsf{P} \equiv \mathsf{P}'$, when they denote the same term up to the laws of Abelian monoids (unit, associativity and commutativity) for parallel composition $\cdot|\cdot$, with $\varnothing$ as the unit, and the laws of idempotent Abelian monoids for choice $\cdot + \cdot$, with $\mathbf{0}$ as the unit. We also assume $D_1|D_2 \equiv D_1 \cup D_2$ for any $D_1, D_2 \subseteq S$. Indexed sums and parallel compositions are denoted, respectively, by $\sum_{i \in I} \mathsf{K}_i$ and $\prod_{i \in I} \mathsf{M}_i$.

The SOS semantics of RS processes is defined by the SOS rules in Fig. 2. A transition label ℓ, written $\langle\langle D \rhd R', I', C\rangle \rhd R, I, P\rangle$, records: the available entities D; the entities C provided by the guarded contexts, assuming all entities in R' are present and those in I' are absent; the set R of entities whose presence enables or disables some reactions; the set I of entities whose absence enables or disables some reactions; and the set P of reaction products. The rules guarantee that, whenever $\mathsf{P} \xrightarrow{\langle\langle D \rhd R', I', C\rangle \rhd R, I, P\rangle} \mathsf{P}'$, it holds that (R', I', C) is enabled in D and that (R, I, P) is enabled in $W \triangleq (D \cup C)$.

The rule (*Ent*) records the set of current entities D. By rule (*Cxt*), a guarded context process $(R, I, C).\mathsf{K}$ makes available the entities in C if the reactants R are present in the current state and the inhibitors I are absent, and then reduces to K. Rules (*Suml*) and (*Sumr*) select a move of either the left or the right context, resp., discarding the other process. By rule (*Rec*), a context identifier X behaves according to its defining process K.

The rule (*Pro*) assumes the reaction (R, I, P) is enabled: it records its reactants, inhibitors, and products in the label, and leaves the reaction available at the next step, together with its products P. The rule (*Inh*) records in the label the reasons why the reaction (R, I, P) should not be executed: possibly some inhibiting entities ($J \subseteq I$) are present or some reactants ($Q \subseteq R$) are missing, with $J \cup Q \neq \varnothing$, as at least one cause is needed.

The rule (*Par*) puts two processes in parallel by pooling their labels and joining all labels components. We write $\ell_1 \cup \ell_2$ for the component-wise union of labels, while the sanity check $\ell_1 \frown \ell_2$ is required to guarantee that labels of reactants and inhibitors are consistent (see definitions in Fig. 2).

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

$$\frac{}{D \xrightarrow{\langle\langle D \rhd \varnothing,\varnothing,\varnothing\rangle \rhd \varnothing,\varnothing,\varnothing\rangle} \varnothing}\ (Ent) \qquad \frac{}{(R,I,C).\mathsf{K} \xrightarrow{\langle\langle \varnothing \rhd R,I,C\rangle \rhd \varnothing,\varnothing,\varnothing\rangle} \mathsf{K}}\ (Cxt)$$

$$\frac{\mathsf{K}_1 \xrightarrow{\ell} \mathsf{K}_1'}{\mathsf{K}_1 + \mathsf{K}_2 \xrightarrow{\ell} \mathsf{K}_1'}\ (Suml) \qquad \frac{\mathsf{K}_2 \xrightarrow{\ell} \mathsf{K}_2'}{\mathsf{K}_1 + \mathsf{K}_2 \xrightarrow{\ell} \mathsf{K}_2'}\ (Sumr) \qquad \frac{X = \mathsf{K} \in \Delta \quad \mathsf{K} \xrightarrow{\ell} \mathsf{K}'}{X \xrightarrow{\ell} \mathsf{K}'}\ (Rec)$$

$$\frac{}{(R,I,P) \xrightarrow{\langle\langle \varnothing \rhd \varnothing,\varnothing,\varnothing\rangle \rhd R,I,P\rangle} (R,I,P)|P}\ (Pro) \qquad \frac{J \subseteq I \quad Q \subseteq R \quad J \cup Q \neq \varnothing}{(R,I,P) \xrightarrow{\langle\langle \varnothing \rhd \varnothing,\varnothing,\varnothing\rangle \rhd J,Q,\varnothing\rangle} (R,I,P)}\ (Inh)$$

$$\frac{\mathsf{M}_1 \xrightarrow{\ell_1} \mathsf{M}_1' \quad \mathsf{M}_2 \xrightarrow{\ell_2} \mathsf{M}_2' \quad \ell_1 \frown \ell_2}{\mathsf{M}_1 \mid \mathsf{M}_2 \xrightarrow{\ell_1 \cup \ell_2} \mathsf{M}_1' \mid \mathsf{M}_2'}\ (Par) \qquad \frac{\mathsf{M} \xrightarrow{\langle\langle D \rhd R',I',C\rangle \rhd R,I,P\rangle} \mathsf{M}' \quad R' \subseteq D \quad R \subseteq D \cup C}{[\mathsf{M}] \xrightarrow{\langle\langle D \rhd R',I',C\rangle \rhd R,I,P\rangle} [\mathsf{M}']}\ (Sys)$$

where $\ell_1 \frown \ell_2$ and $\ell_1 \cup \ell_2$ are defined as follows:

$$\langle\langle D_1 \rhd R_1', I_1', C_1\rangle \rhd R_1, I_1, P_1\rangle \frown \langle\langle D_2 \rhd R_2', I_2', C_2\rangle \rhd R_2, I_2, P_2\rangle \triangleq (\textstyle\bigcup_{i=1,2} D_i \cup R_i') \cap (I_1' \cup I_2') = \varnothing \wedge (\bigcup_{i=1,2} D_i \cup C_i \cup R_i) \cap (I_1 \cup I_2) = \varnothing$$

$$\langle\langle D_1 \rhd R_1', I_1', C_1\rangle \rhd R_1, I_1, P_1\rangle \cup \langle\langle D_2 \rhd R_2', I_2', C_2\rangle \rhd R_2, I_2, P_2\rangle \triangleq \langle\langle D_1 \cup D_2 \rhd R_1' \cup R_2', I_1' \cup I_2', C_1 \cup C_2\rangle \rhd R_1 \cup R_2, I_1 \cup I_2, P_1 \cup P_2\rangle$$

Fig. 2 SOS semantics of the RS processes.

Finally, the rule (*Sys*) checks that all the needed reactants are available in the system. Checking the absence of inhibitors is not necessary, thanks to the sanity check in rule (*Par*). Note that, while the enabling of $(R, I, C).\mathsf{K}$ requires the presence of reactants R and the absence of inhibitors I w.r.t. the set of current entities D, in the case of reactions (R, I, P), the check is performed w.r.t. the current state $W = D \cup C$. More importantly, the products C are made available immediately from the context, not at the next step. It is worthy to mention that a conditional prefixed process that is not enabled behaves as the **0** process.

Remark 1 It is worth noting that any RS process can be written in the form $[\mathsf{Rs} \mid \mathsf{Ks} \mid D]$ where $\mathsf{Rs} = \prod_i (R_i, I_i, P_i)$ is the parallel composition of all reactions in the system, $\mathsf{Ks} = \prod_j \mathsf{K}_j$ is the parallel composition of all contexts and D is the set of currently available entities. Moreover, it can be proved that a generic transition has the shape: $[\mathsf{Rs} \mid \mathsf{Ks} \mid D] \xrightarrow{\langle\langle D \rhd R',I',C\rangle \rhd R,I,P\rangle} [\mathsf{Rs} \mid \mathsf{Ks}' \mid P]$, i.e., reactions are always preserved by transitions, the first component D of the transition label is just the set of available entities in the source state and the new result set in the target state is just the product set P observed in the label of the transition. The choices in Ks are taken by considering the set of available entities D (in the case of guarded prefixes) and will determine the context C appearing in the label as well as the continuation Ks' appearing in the target state. Given D and C, the product P is then uniquely determined by Rs. For brevity, we will sometimes draw the LTS, by recording only the strict amount of information in nodes and labels. Thus the above sample transition will be abbreviated as $[\mathsf{Ks} \mid D] \xrightarrow{C} [\mathsf{Ks}' \mid P]$, assuming reactions Rs are known a priori.

A first concrete example of RS that exposes most features is presented in Sect. 3.

2.2 GT and GROOVE

Graph Transformation (GT, sometimes called Graph Rewriting) is a well-established rule-based formalism, the core of which is to specify precisely how graphs may evolve. Each rule embodies a particular change, which can be applied to a given graph (in the simplest form consisting of nodes and binary edges) by establishing where in that graph the preconditions of the rule are met, and then adding and deleting nodes and edges as prescribed.

In this paper, we use the so-called *algebraic approach* to graph transformation (see Ehrig et al. 2006 for a formal exposition and Heckel and Taentzer (2020) for applications in the context of software engineering); moreover, we rely on the particular flavour implemented in the tool GROOVE (Ghamarian et al. 2012; Rensink 2024). Some of the relevant features of the approach and the tool are highlighted below.

Graphs are *simple* and *typed*, meaning that there is at most one edge of a given type between any two nodes and that all nodes and edges are labelled through a morphism to a given (fixed) *type graph*. Edges are *directed* (going from their *source* to their *target*). Besides binary edges, nodes can also have *flags* (which are actually self-loops that act as additional, optional labels on nodes) and *attributes* (which are actually

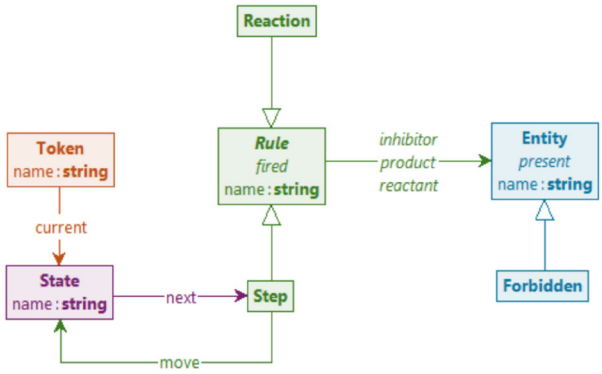


Fig. 3 Core type graph for reaction systems

binary edges whose target node is a data value, e.g., an integer or a string).

Rules, in their simplest form, consist of a left hand side (LHS) and right hand side (RHS). Rule applicability is established by *matching* the LHS to the graph in question, and where a match exists, removing nodes and edges that are in the LHS but not in the RHS, and vice versa, adding nodes and edges that are in the RHS but not in the LHS. In addition, however, GROOVE supports *quantified* rules, which can simultaneously be applied to multiple places in the same graph. An example is shown in Fig. 7 below.

Evolution of a graph is defined on the basis of a graph transformation *system*, which is a set of rules applied to a graph at hand, giving rise to a modified graph to which every rule can be applied again, and so forth. On top of this, GROOVE allows for *control programs* that can specify in what order rules may be applied. By exploring the potential evolution of a graph in all ways allowed by the control program, GROOVE constructs the *state space* of the graph transformation system, in the form of a *labelled transition system* consisting of all reachable graphs and the rule applications between them.

Analysis consists of the exploration of the state space for a given initial graph, rule system and (optional) control program. The exploration can be tuned by built-in strategies for searching and model checking.

The power of graph transformation lies in its generality: many systems naturally lend themselves to be modelled as graphs, and algebraic rules—especially quantified ones—provide a rich framework to specify their evolution. This is in fact our motivation for using it the current paper: reaction systems can straightforwardly be interpreted as graphs. Figure 3 shows the core types for the relevant concepts of that interpretation. (The colours just support the visualisation and have no semantics of their own.)

Note especially the (abstract) supertype **Rule** with subtypes **Reaction** and **Step**: the former is the type for the elements of A in a Reaction System $A = (S, A)$, whereas the latter is used to represent triples (R, I, P) in a context process K. The flag *fired* is used to mark **Rule**s that have triggered in the most recent step. The set S is represented by nodes of type **Entity**; for a given **Rule**, the subsets R, I, and P of S are those **Entity**s to which there is an outgoing edge labelled reactant, inhibitor or product. The subtype **Forbidden** anticipates the principle, demonstrated later in this paper, of identifying undesirable entities and specifically searching for scenarios in which those are produced. The flag *present* is used to label the entities occurring in a state W. Finally, the structure of (guarded) context processes is captured by **State** entities, with next-edges to the **Step**s that can be non-deterministically chosen; the subsequent process after such a **Step** is determined by its outgoing move-edge. **Token** nodes are used to model which **State**s are currently active.

3 Running example: a toy vending machine

To illustrate some basic concepts of RSs and, in the next section, of the proposed GROOVE encoding, we model a system composed of a student and a vending machine as a toy example. The vending machine accepts two different kinds of coins and can dispense either a cappuccino or an espresso when a coffee coin is inserted or a tea if a tea coin is inserted. A cappuccino is dispensed if some milk is available, otherwise espresso is produced. Assuming the powder for preparing coffee and tea are always present, the corresponding process can be written as follows:

$$\begin{aligned}\mathsf{VM} \triangleq\ & (\{\mathsf{ccoin}, \mathsf{cpowder}\}, \{\mathsf{nomilk}\}, \{\mathsf{cappuccino}\})\\ |\ & (\{\mathsf{ccoin}, \mathsf{cpowder}, \mathsf{nomilk}\}, \varnothing, \{\mathsf{espresso}\})\\ |\ & (\{\mathsf{tcoin}, \mathsf{tpowder}\}, \varnothing, \{\mathsf{tea}\})\\ |\ & (\{\mathsf{cpowder}\}, \varnothing, \{\mathsf{cpowder}\})\\ |\ & (\{\mathsf{tpowder}\}, \varnothing, \{\mathsf{tpowder}\})\end{aligned}$$

A refill context process can, nondeterministically, refill the machine with milk.

$$\mathsf{Refill} \triangleq \{\mathsf{nomilk}\}.\mathsf{Refill} + \varnothing.\mathsf{Refill}$$

The student process is very simple: she takes cappuccino in the morning and tea in the afternoon, otherwise she gets angry.

$$\begin{aligned}\mathsf{Student} &\triangleq (\{\mathsf{am}\}, \varnothing, \{\mathsf{ccoin}\}).\mathsf{GetCappuccino}\\ &\quad + (\varnothing, \{\mathsf{am}\}, \{\mathsf{tcoin}\}).\mathsf{GetTea}\\ &\quad + \{\mathsf{idle}\}.\mathsf{Student}\\ \mathsf{GetCappuccino} &\triangleq (\{\mathsf{cappuccino}\}, \varnothing, \varnothing).\mathsf{Student}\\ &\quad + (\{\mathsf{espresso}\}, \varnothing, \{\mathsf{anger}\}).\mathsf{Student}\\ \mathsf{GetTea} &\triangleq (\{\mathsf{tea}\}, \varnothing, \varnothing).\mathsf{Student}\\ &\quad + (\varnothing, \{\mathsf{tea}\}, \{\mathsf{anger}\}).\mathsf{Student}\end{aligned}$$

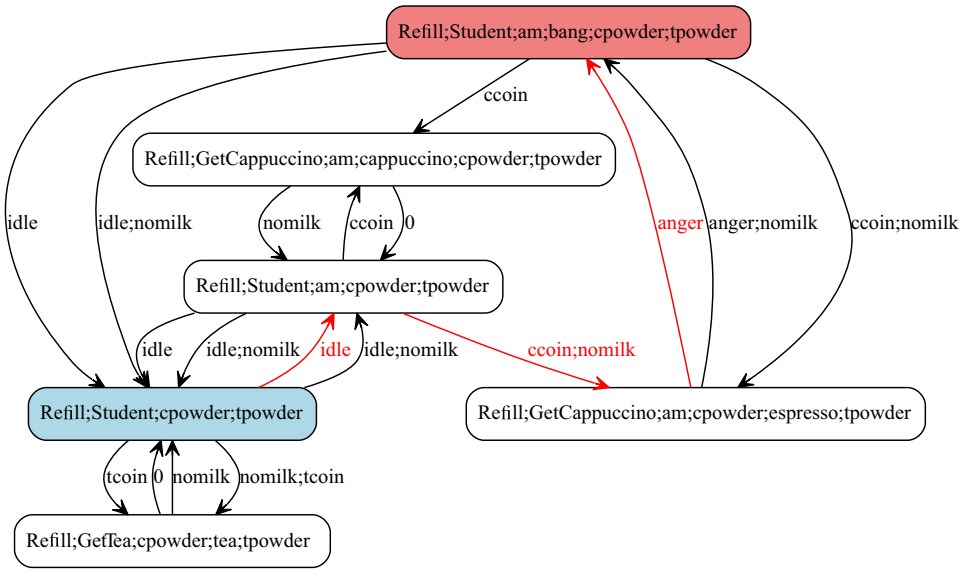


Fig. 4 LTS of the toy example. For brevity we use the notation introduced in Remark 1, where node labels only account for the list of (semicolons separated) current contexts and entities and transition labels carry the entities provided by the context

If the student is angry, she will bang the machine:

$$\mathsf{Anger} \triangleq (\{\mathsf{anger}\}, \varnothing, \{\mathsf{bang}\})$$

Finally, two more reactions model the passage of time (morning vs afternoon) while the student is idle (i.e., not in the process of getting beverages).

$$\begin{aligned}\mathsf{Day} \triangleq\ & (\{\mathsf{idle}\}, \{\mathsf{am}\}, \{\mathsf{am}\}) \\ \mid\ & (\{\mathsf{am}\}, \{\mathsf{idle}\}, \{\mathsf{am}\})\end{aligned}$$

We assume that, initially, both entities cpowder and tpowder are present. So the system can be coded as the guarded RS process

$$[\mathsf{Refill} \mid \mathsf{Student} \mid \{\mathsf{cpowder}, \mathsf{tpowder}\} \mid \mathsf{VM} \mid \mathsf{Anger} \mid \mathsf{Day}].$$

The complete encoding of the above RS in BioResolve syntax is reported in Fig. 19 in the Appendix. Using the BioResolve directive `main_do(digraph)`, we can automatically generate the underlying LTS as in Fig. 4: the initial state is in light blue, while there is also a "bad" state in which the student is banging the machine, shown in light coral. Note that, as the entity am is initially not present, it means that the initial time is in the afternoon. In drawing the transition system, we have used the representational convention explained in Remark 1).

In this particular example, we wish to analyse why bang is produced, i.e., what are its causes. By manual inspection we can recover a trace starting from the initial state and leading to the "bad" state, e.g., $\xrightarrow{\mathsf{idle}}\ \xrightarrow{\mathsf{ccoin;nomilk}}\ \xrightarrow{\mathsf{anger}}$ (highlighted in red in Fig. 4). From this trace, using the dynamic slicing process described in Brodo et al. (2024a), we can reconstruct that the production of bang was due to the prior production of anger, which in turn was caused by the student getting espresso instead of cappuccino, which is because there was nomilk when a ccoin was inserted at am.

4 Encoding of RS in GT

As an alternative to BioResolve, we investigate the use of graph transformation and GROOVE to generate the underlying LTS of a given Reaction System, on the basis of a start graph obtained by transformation from the RS specification. Because the start graph will typically include special **Entity** subtypes, it comes together with an additional type graph where those are specified. Depending on what one wants to analyse, the various strengths and capabilities of GROOVE then come into play.

For instance, one possibility is to use GROOVE's model checking capabilities to check for temporal patterns of entity generation in the transition system. Another way to proceed is to focus on a given trace and build its *occurrence graph*, which contains all the rule occurrences and entity instances present in that trace—analogous, in fact, to the way a Petri net process captures a particular behaviour. If the trace in question leads to a state in which a **Forbidden** entity is present (such as the bang entity in our toy example), we can also *prune* the occurrence graph, again using graph transformation, to keep only those rule occurrences and entity instances that directly contributed to the existence of the forbidden entity.

This gives rise to the tool chain depicted in Fig. 5, the phases of which will be explained in some more detail in the remainder of this section.

Transform. The first step is a text-to-model transformation from a problem specification in BioResolve syntax

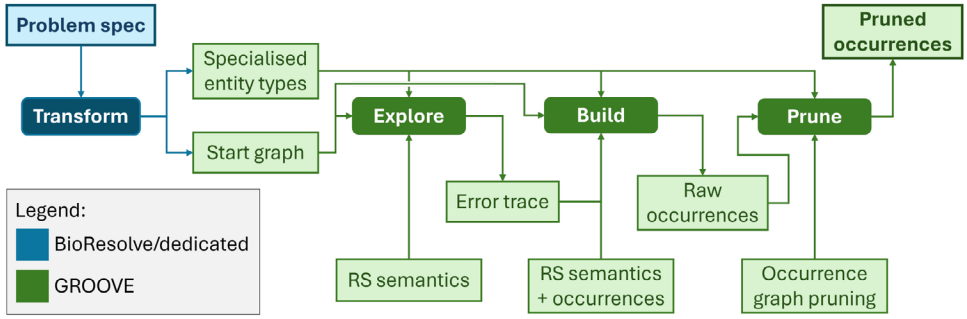


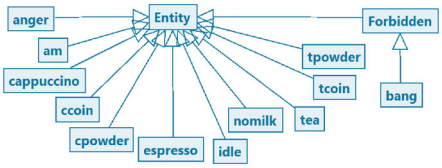
Fig. 5 Reaction system exploration and analysis using GROOVE

into GROOVE syntax. This is achieved by running the `main_do(rs2gts)` directive of BioResolve, which produces two artefacts: firstly, an additional type graph, complementary to the one shown in Fig. 3, which specifies one subtype of **Entity** for each of the entities in the problem at hand (essentially for performance reasons: relying on dedicated types speeds up the matching step of GROOVE); and secondly (more importantly) a start graph in which the entire BioResolve system is encoded as suggested by Fig. 3. For the example system, the additional types as well as two self-explanatory fragments of the start graph are shown in Fig. 6.

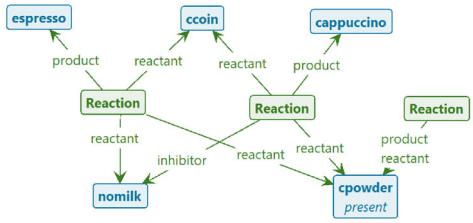
We claim that this transformation is semantics-preserving; Appendix A gives a sketch of the argument. A fully formal statement and proof of semantic correspondence, however, is outside of the scope of this paper.

Explore. The dynamics of Reaction Systems is encoded as a combination of two rules, context and react, which are scheduled to fire in alternation. The rule context encodes the simultaneous firing of all context processes (nondeterministically selecting an enabled **Step** from every **State** with a **Token**), whereas react encodes the (deterministic) simultaneous firing of all enabled **Reaction**s, while simultaneously erasing all **Entity**s that were not just produced. The production or erasure of an **Entity** is encoded through the creation or deletion of a *present* flag on a (persistent) **Entity** node, *not* by the creation or deletion of the node itself. In addition, to keep track of which nondeterministic choices were actually taken, the context rule marks the **Step**s that were selected with a *fired* flag, which is subsequently erased by the react rule.

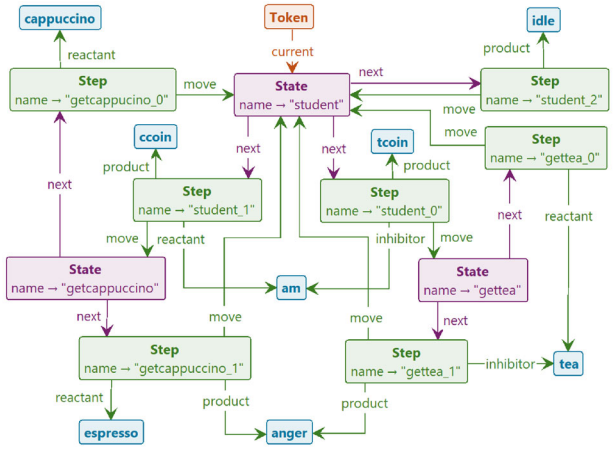
Figure 7 shows the first (and most intricate) of these rules, viz. the one for the context firing. This is a quantified rule, which can be read as follows: For all **State**s with a **Token**, there is a next **Step** such that for all inhibitors there is no *present* flag whereas for all reactants there is a *present* flag; moreover, when the rule is applied, all products of the selected **Step**s receive a *present* flag, the **Step**s themselves receive a *fired* flag, and all **Token**s move to the successor **State**s. Colour coding is used in the visual representation to distinguish the quantifier nodes ∀ and ∃ (both in purple), as well as the mandatory absence (red), deletion (blue) and creation (green) of edges and flags.[3]



(a) Specialised entity types



(b) Start graph fragment: Three reactions from VM



(c) Start graph fragment: The Student context process

Fig. 6 Graph representation of running example

[3] This colour coding is GROOVE-specific and entirely separate from the problem-specific colouring of the graph nodes in Figs. 3 and 6; in fact, to avoid confusion, the problem-specific colouring is *not* used in the rule view.

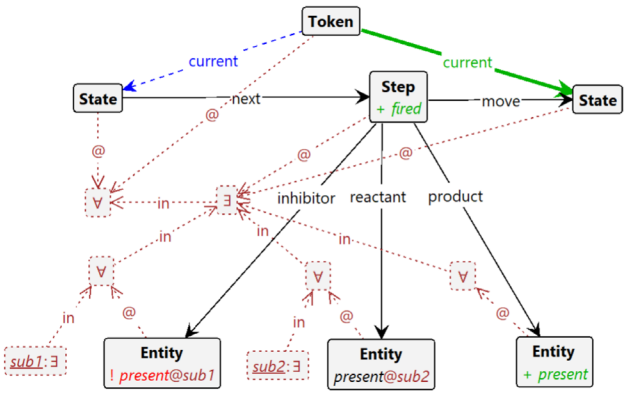


Fig. 7 Rule for context firing

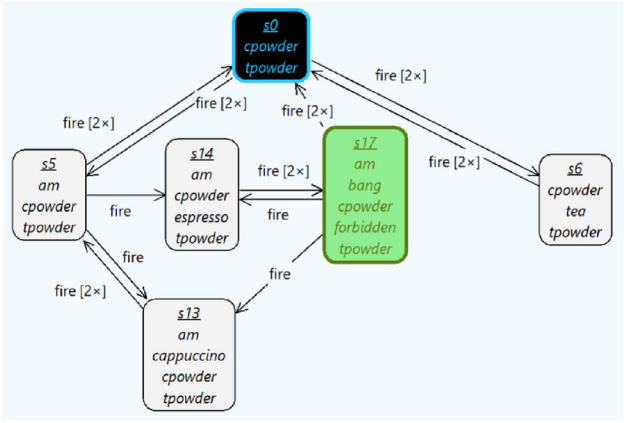


Fig. 8 GROOVE LTS of the toy example

To mimic the BioResolve semantics as closely as possible, we can instruct GROOVE to regard every pair of context- and react-transitions as an atomic transaction, corresponding precisely to a transition in BioResolve (though not with the same label), and then generate the entire state space. This is achieved through a control program of the form

```
recipe fire() {
  context; react;
}
```

where a "recipe" is the keyword for a transaction wrapping the body. With this in place, the GUI-based version of GROOVE produces the transition system displayed in Fig. 8 (which can also be exported to a range of standard formats), which is easily (visually) checked to be essentially isomorphic to the one in Fig. 4.

Alternatively, we can (for instance) ask GROOVE to search the for the first reachable state in which a **Forbidden** entity appears, using breadth-first search. (In fact, the state property for which GROOVE searches is itself determined by a *rule*, which in this case merely tests for the presence of a **Forbidden** entity). When found, the trace to the forbidden state can be saved as a control program that drives the next stage of GROOVE exploration. In particular, using the alternating application of the context and react rules (rather than the transactional variant used for Fig. 8) this control program also records the fired-applications that tell which **Step**s have fired: this completely determines how the non-determinism in the context process has been resolved in order to arrive at the forbidden state. Here is the control program for the shortest trace to state *s17* in our running example:

```
context;
fired("student_2");
fired("refill_1");
react;
context;
fired("student_1");
fired("refill_0");
react;
context;
fired("refill_1");
fired("getcappuccino_1");
react;
```

Here student_2, student_1 and getcappuccino_1 are the **Step**s of the Student process visualised in Fig. 6; refill_1 and refill_0 are the steps of the Refill process given in Sect. 3.

Build. The purpose of this phase is to build an occurrence graph that explains how **Forbidden** was produced, by collecting its (transitive) dependencies (Brodo et al. 2024b). Concretely, we record the following dependencies:

- From each non-initial **Entity** instance to the **Rule** occurrence of which it is the product;
- From each **Rule** occurrence to all its reactant **Entity** instances;
- From each **Step** occurrence to all directly preceding **Step** occurrences.

Figure 9 shows the occurrence type graph.

With respect to the slicing algorithm used in, e.g., Brodo et al. (2025), Brodo et al. (2024a), the difference is that our dependencies are based on instances, and that we explicitly include the rule occurrences and their predecessors. Each of these different kinds of dependencies is visualised both through a specific edge (product, reactant or predecessor) between the relevant instance- and occurrence-nodes, and (for the sake of a more uniform treatment during the next step of *pruning*) through an auxiliary depends-edge on the level of **TreeNode**, of which all others are subtypes.

Note that all this is restricted to *positive* dependencies. In fact, we have constructed our example so that there are no inhibitors in the **Rule**s that fire in the trace above; if we would rely on an entity milk that inhibits the production of espresso, rather than on nomilk as a reactant, the occurrence graph for bang would not include milk; and likewise if we

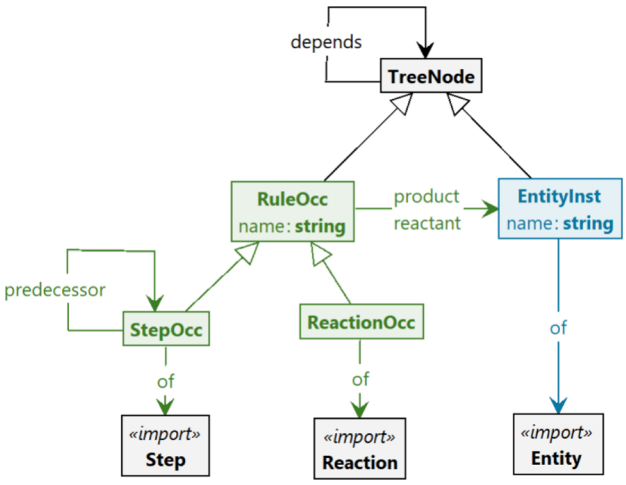


Fig. 9 Occurrence type graph

would use cappuccino as an inhibitor for anger rather than espresso as a reactant for it. The representation of negative dependencies is a research question in its own, and is outside the scope of this paper (although a possibility to capture negative dependencies is to focus the analysis on the Positive RS that results from the transformation defined in Brodo et al. (2024b)).

As indicated in Fig. 5, the occurrence graph is produced by another GROOVE rule system, using the same start graph but driven by the control program previously created in the explore phase, which encodes (as we have seen) the trace to the undesirable state. The occurrence graph semantics consists of rules with the same names (react, context and fired), but different functionality: in particular, rather than manipulating *present* flags, the react rule now creates **RuleOcc**- and **EntityInst**-nodes together with their dependencies. This is a non-trivial procedure that in fact itself requires several successive stages. Though the details of these stages are not of sufficient interest to include in this paper, we want to point out that breaking down a single rule (react, in this case) into multiple stages would seem to contradict the tenet of algebraic graph transformation that a rule embodies a single, atomic change to a graph. This contradiction is solved by relying once more on GROOVE recipes: in the occurrence graph semantics, react is actually not a rule but a recipe, defined as

```
recipe react() {
  entities-age;
  react-produce;
  merge;
}
```

of which the three atomic steps perform the necessary bookkeeping to correctly produce the occurrence graph.

Prune. The occurrence graph built by the rule system described above is too large to be useful: it contains *all* entity instances and rule occurrences produced by the trace, not just the dependencies of the undesired **Forbidden** entity. Moreover, the entire start graph is also (still) present. Therefore, in a third phase, all redundant information is pruned. This is achieved by first marking all transitive backward dependencies, and then removing all unmarked nodes. Since this is straightforward, and of no particular interest in the context of this paper, we omit the details of the GROOVE rule system for this phase. Its outcome for our running example is shown in Fig. 10 (where we have elided the auxiliary depends-edges).

The pruned occurrence graph visualises the causal effect chain already explained informally at the end of Sect. 3: the presence of a ccoin, which itself depends on am, combined with cpowder and nomilk causes the production of espresso, after which the student produces anger and then bang, which is **Forbidden**.

5 Experimentation

Here we consider three larger case studies whose RS specifications have already appeared in the recent literature. For each case study, we briefly describe its main features and then show how the methodology outlined in the previous sections can be applied for carrying out some fruitful experimentation with GROOVE.

All GROOVE experiments were carried out using GROOVE version 7.4.3 on a Dell Precision 3551 laptop with an Intel i7 CPU running at 2.6 GHz; GROOVE was run in a Java 24 JVM with 12GB of memory. No attempt was made to measure running time with precision, and repeated experiments have shown that the reported durations can deviate up to 25%. In order to facilitate replication of the experiments, we have included supplementary materials with this paper, including the required rule systems and start graphs and instructions for invoking GROOVE; see Sect. C.5.

5.1 Comorbidity treatment analysis

This case was studied in Bowles et al. (2024), where guarded contexts were introduced to handle key features of medical treatments. It concerns the risk mitigation of medication harm in the treatment of patients with comorbidities; i.e., patients with two or more long-term chronic conditions (such as diabetes, hypertension, cardiovascular diseases, chronic kidney disease, cancer, chronic obstructive pulmonary disease, among many others), who are therefore subject to follow several treatment plans simultaneously, called *clinical guidelines* (Feder et al. 1999; Woolf et al. 1999). Since clinical guidelines address a single disease, comorbidities easily lead to *polypharmacy*, where five or more medications must be

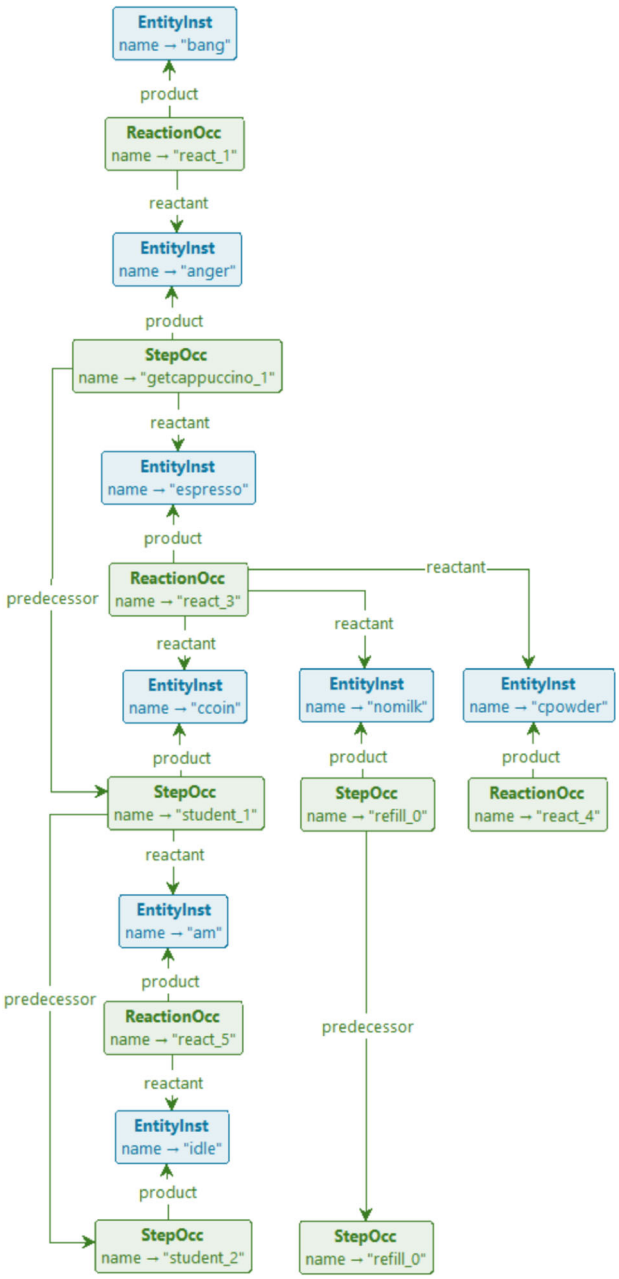


Fig. 10 Pruned occurrence graph of the running example

administered, increasing the risk of adverse drug reactions, or of making certain drugs less effective when combined (Hughes et al. 2013).

Analysis goals. The goal of the analysis is to explore the combination of clinical guidelines in the presence of comorbidities and for different patient profiles to detect if major risks can arise from the treatments and which profiles are exposed at severe risks. Using formal methods for risk mitigation intends to help doctors choose between alternative treatment options as well as to point out missing conditions that could be helpful to revise and update clinical guidelines.

Features of interest. In this case study, reachability and causal analysis are key issues. Specifically, reachability is used to address questions such as *can the combination of clinical guidelines expose the patient at serious risks because of drugs interference?* Then, in the affirmative case, causal analysis can help to detect which medical decisions would be directly responsible for causing serious harm as well as to point out which alternative treatment would be available, if any. We selected this case study because the use of guarded contexts introduces new challenges for the causal analysis of RSs: while existing approaches typically focus on identifying combinations of drugs administered within the context that may have caused harm, they often fail to highlight the medical decisions that led to their administration.

Experimental set up. The RS encoding proposed in Bowles et al. (2024) relies on a formal representation of patient profiles, medical guidelines and adverse drug reactions.

For each drug d that appears in the therapies, we consider three corresponding entities get_d, stop_d and d: the first represents the prescription of d by the doctor, the second the removal of d from the current treatment and the third the intake of the drug by the patient. For handling multiple drugs of the same class c, we exploit analogous entities stop_c and c.

For medical guidelines, it takes in input the event structure modelling of therapies introduced in Bowles and Caminati (2017). Roughly, to each event e there is an identifier E_e defined as a sum of processes, one for each outgoing arc of e. If some guard is attached to the arc, then the corresponding alternative is also guarded. The prescription of a drug d is modelled by the provision of the entity get_d. Similarly, if the therapy requires stopping the drug d, the entity stop_d is produced.

The patient profile is determined by the conditions that trigger the treatment (e.g., headache, hypertension) and by the conditions that appear in the arc labels of the event structure (e.g., pregnant, asthma). We call them *features*. Correspondingly, there is one context $\mathsf{K}_f = \{f\}.\mathsf{Emp}$ for each feature f, and a patient profile is just a combination of some features $\mathsf{K}_{f_1} \mid \cdots \mid \mathsf{K}_{f_n}$. Once the profile is determined by the context, it is preserved during the rest of the computation by reactions of the form $(\{f\}, \varnothing, \{f\})$, one for each feature. Accounting for all possible patient profiles within a single model can be done by considering the context $\prod_f(\mathsf{K}_f + \mathsf{Emp})$.

For each drug d of class c, there will be the following reactions: $(\{\mathsf{get}\_d\}, \{\mathsf{stop}\_d, \mathsf{stop}\_c\}, \{d, c\})$ modelling the intake of the drug d as for doctor prescription, and $(\{d\}, \{\mathsf{stop}\_d, \mathsf{stop}\_c\}, \{d, c\})$ modelling the prosecution of the therapy. Adverse drug reactions are provided in the form of so-called ADR tables. Each row corresponds to a set of medications M, a textual description of their side effects and

risks when used in combination, and a severity level m (e.g., minor, moderate, major). Each row translates to a reaction $(M, \varnothing, \{m\})$.

The whole RS specification can be found in the Appendix: the list of reactions is in Fig. 20 and context processes are defined in Fig. 21.

Previous approach. The approach outlined in Bowles et al. (2024) has been used to synthesise the patient profiles that are more at risk, as a support for dynamic guideline revision: by refining guarded contexts to prevent severe effects for specific patients, we can readily check the efficacy of the changes.

GROOVE experimentation. The benefit of using GROOVE in this case study is that, besides identifying situations where a risk is found, for any risk so identified the corresponding occurrence graph of a risk can be generated, using the process outlined in Sect. 4. This provides a means for medical experts to more easily analyse root causes: for any risk that has been identified, what is the causal structure of the steps and entities leading up to it?

GROOVE can be used for full state space generation, for instance to count the number of ways a minor, moderate or major risk can arise. Some statistics can be found in Fig. 11.

The first two lines show that depth-first exploration strategy (DFS) is generally more convenient than breadth-first (BFS), so we rely on the former for the subsequent entries. At first sight, the number of major (and moderate) risks reported in Fig. 11 seems impossibly large, and evidently implies that there are patient profiles that give rise to many risks. However, this should be interpreted with care: the count refers to the total number of configurations containing a **Forbidden** (i.e., major or minor) entity, and there may very well be entities whose presence or absence does not causally contribute to that **Forbidden** entity—in other words, which would not appear in its occurrence graph. Configurations counted as separate risks may well reduce to the same causation. To analyse this further, one would have to construct (and prune) the occurrence graphs for all risk configurations, and compare them on that basis. Though this is beyond the scope of this paper, such an analysis is in principle straightforward to carry out in GROOVE—it is a matter of combining the three steps in Fig. 5 into a single rule system.

The line of Fig. 11a headed "Maj&Mod risks" reports the number of configurations at which *both* a major and a moderate entity appear during the same step; hence, these are counted as both major and moderate risks (partially explaining their high numbers).

By exploring only up to a certain depth, we can get some idea of the number of steps after which a risk typically appears, which in turn indicates the complexity of the context in which it appears. Figure 11b shows how many major risks occur after a fixed number of reaction steps, and also how much of the total state space is involved. Note that, in this case, the total state count (reached after 12 steps) stays below that of Fig. 11b; this is because in the experiments of Fig. 11b we stop exploration at states where a risk has been found. Figure 11c shows the same data in a graphical form.

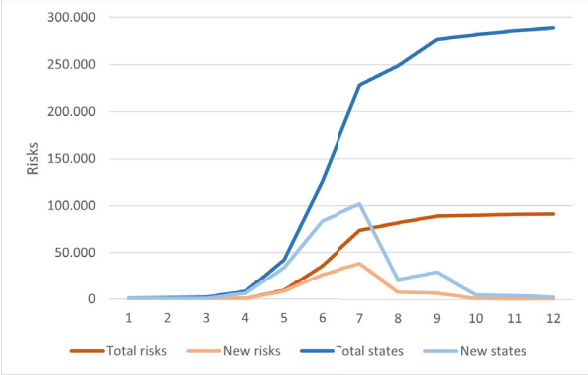
Alternatively, we can stop exploring after having found a predetermined number of risks. By setting the exploration strategy to breadth-first search, it is guaranteed that the risks found are those reached after the shortest number of steps, meaning they are the easiest to analyse visually. For instance,

Measurement	Search method	Count	Time (s)
Total states	BFS	309 798	3 368
Total states	DFS	309 798	2 464
Major risks	DFS	91 113	2 447
Moderate risks	DFS	97 805	1 534
Maj&Mod risks	DFS	61 976	2 727
Minor risks	DFS	24	2 218

(a) Full exploration

Explore depth	Total risks	New risks	Total states	New states
1	0	0	769	769
2	0	0	1 345	576
3	0	0	1 873	528
4	716	716	8 133	6 260
5	9 317	8 601	42 293	34 160
6	35 899	26 582	126 012	83 719
7	73 591	37 693	227 866	101 854
8	81 691	8 100	248 420	20 554
9	88 631	6 940	276 924	28 504
10	89 733	1 102	281 854	4 930
11	90 573	840	286 054	4 200
12	91 133	560	288 854	2 800

(b) Bounded BFS exploration of major risks (tabular)



(c) Bounded BFS exploration of major risks (chart)

Fig. 11 States, risks and execution time

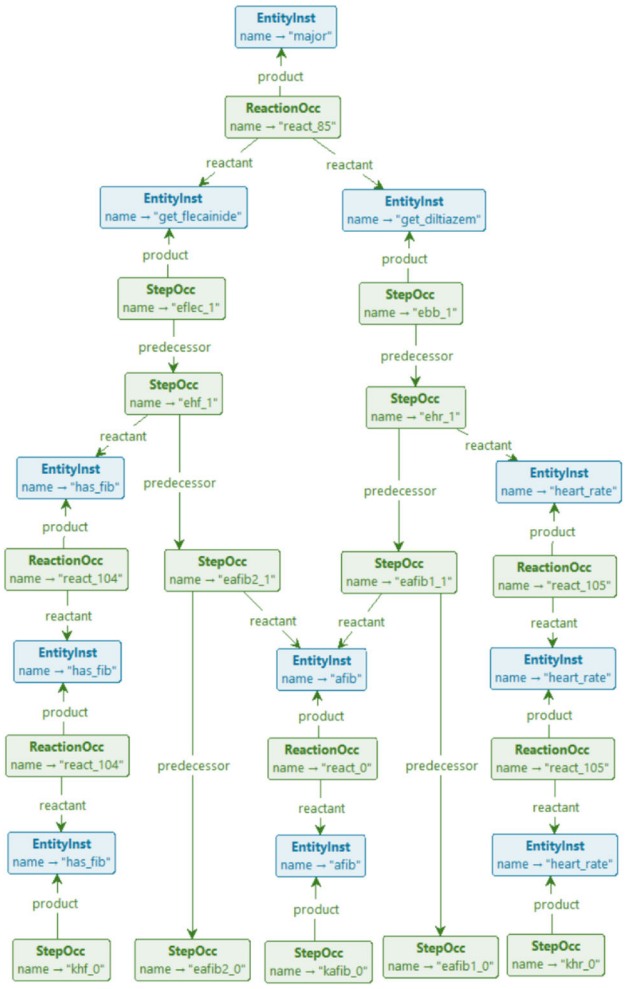


Fig. 12 Occurrence graph of a major risk

Fig. 12 shows the occurrence graph of the first major risk found in this way.

The patient configuration in question is a combination of `has_fib`, `afib` and `heart_rate`; the combination of the first two leads to the prescription of `fleacinide` and the second to the prescription of `diltiazem`, the combination of which should, however, be avoided. By counting the longest chain of **StepOcc**-nodes, it is confirmed that it indeed takes 4 steps to establish this risk.

We can also use GROOVE to replicate the findings of Bowles etal., (2024, Fig. 6) in terms of the relation between patient profiles and risks, using model checking. Recall that the reaction system starts by having the context produce initial entities, and in this particular case study, the first move of the context is to select a patient profile; hence the initial state has $2^9 = 512$ outgoing transitions, whose target state corresponds to the chosen profile. Moreover, the rule forbidden tests for the presence of a **Forbidden** entity in a state. Therefore, a formula of the shape $\mathsf{AX}(\bigwedge_i f_i \rightarrow \mathsf{EF}\,\mathsf{forbidden})$, where each of the f_i specifies the presence or absence of a patient feature, specifies whether all patient profiles with that combination of features contain a potential risk.

Concretely, in the case study at hand, Bowles et al. (2024, Fig. 6) contains necessary and sufficient criteria for patient profiles to contain major, moderate and minor risks. The part of the table pertaining to major risks is reproduced in Fig. 13. This should be read as: *precisely* in the combination of features where the green ones are present and the red ones absent, a major risk may occur.

In order to replicate these results, we can use CTL model checking. For instance, the relevant CTL property for the major risks is also shown in Fig. 13. Running the CTL model checker built into GROOVE, it reports that this is indeed satisfied, as are the corresponding characteristic properties for the moderate and minor risks. The following table reports the time taken for these checks (where the precision of our time measurement is such that the apparent difference between the model checking times is not significant):

Model generation:	2236 s
Major risk check:	67 s
Moderate risk check:	71 s
Minor risk check:	65 s

Note that the time taken for model generation is consistent with that reported in Fig. 11b.

Discussion. Compared to the prior results in Bowles et al. (2024), the advantages of using GROOVE lie in performance and flexibility:

- The time needed to analyse the entire state space is around 41 min for GROOVE; while not particularly fast, this still compares very favourably to the 5 h needed by BioResolve for LTS generation.
- Finding shortest paths to risk configurations and computing occurrence graphs is part of the core functionality of GROOVE—given, of course, suitable rule systems that encode the chosen notion of causal dependency.
- The CTL check can be used to immediately confirm the outcome of the slicing algorithm.

5.2 Protein signalling networks analysis

This case was studied in Ballis et al. (2024), where it was encoded into the Maude[4] ecosystem (Clavel et al. 2007) to take advantage of their built-in LTL and CTL model checker facilities. It is based on a biological case study from der Heyde et al. (2014), aimed to identify the best drug treatment for three different breast cancer representative cell lines: BT474, SKBR3 and HCC1954. This is achieved by

[4] https://maude.cs.illinois.edu.

	Patient profiles possibily leading to a "major" adverse reaction									
	afib	has_fib	heart_rate	consensus_acei	over75	below55	diabetes	doac_int	hyper	origin
1	TRUE		TRUE	TRUE	FALSE			FALSE	TRUE	
2	TRUE		TRUE	TRUE	FALSE		TRUE	FALSE		
3	TRUE		TRUE	TRUE	TRUE	FALSE		FALSE		
4	TRUE	TRUE		TRUE				FALSE	TRUE	
5	TRUE	TRUE							TRUE	
6	TRUE	TRUE	TRUE							

```
AX((afib           & heart_rate & consensus_acei & !over75                    & !doac_int & hyper |
    afib           & heart_rate & consensus_acei & !over75          & diabetes & !doac_int         |
    afib           & heart_rate & consensus_acei &  over75 & !below55          & !doac_int         |
    afib & has_fib              & consensus_acei                               & !doac_int & hyper |
    afib & has_fib                                                                        & hyper |
    afib & has_fib & heart_rate                                                                  )
   <-> EF forbidden)
```

Fig. 13 CTL encoding of the major risk profiles found in Bowles et al., (2024, Fig. 6)

studying the behaviour of the protein signalling networks for the HER2-positive breast cancer subtype in the presence of different combinations of monoclonal antibody drugs. In a nutshell, Maude is a high-performance reflective language and system based on equational and rewriting logic specification. The encoding of RSs is made possible by setting up a specific rewrite theory, called **ccReact**, which is expressive enough to capture the relevant aspects of the protein signalling networks. The analysis conducted in Ballis et al. (2024) matches previous findings, and makes it possible to readily inspect new hypotheses.

Analysis goals. The goal of the analysis is to validate or refute some behavioural hypotheses of RSs.

Features of interest. Besides reachability analysis, mostly concerned with the possibility to reach certain attractors, the distinguishing feature of this case study is the possibility to model check RSs with guarded contexts against behavioural properties written in LTL and CTL.

Experimental set up. The technique in Ballis et al. (2024) starts directly from a RS specification, which is manually coded in **ccReact** and queried using Maude state exploration techniques and built-in model checkers. Likewise, here we can just exploit the direct translation of RSs (with guarded contexts) to GROOVE presented in the previous section, i.e., no preprocessing is necessary. The BioResolve specification is in Fig. 22 in the Appendix. The following properties have been experimented with:

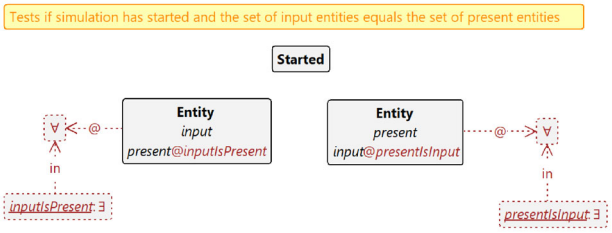
1. searching for the the presence/absence of the attractor `akt` in steady states of the BT747 cell line, where the context `[k,ket]` is considered;
2. in order to observe the interactions when either erlotinib or pertuzumab are supplied, the context `[{e,egf,hrg}.korep]` is considered and Maude reports that there exists at least one path where that treatment is successful, but not all paths avoid a steady state where `akt` is present;
3. using the context `[k,korept]`, it is shown that, regardless the drug used, once `pdk1` is present, inevitably the steady state includes `akt`; and that `pdk1` never appears before `erbb1` is produced (which basically means that `pdk1` is a product of the activation of the `erbb1` receptor);
4. finally, using the context `[k,kge]`, it is shown that by permanently providing the drug erlotinib and the stimulus (`egf` and `hrg`), the attractor `akt` is never produced. Moreover, Maude checks that the production of `akt` can be also inhibited by providing erlotinib only when receptors `erbb1` and `erbb2` are active.

Previous approach. **ccReact** allowed to perform reachability analysis directly exploiting the `search` command of Maude. The formal verification of temporal formulas has been made possible by relying on a general interface to different model checkers for Maude models, called the Unified Maude Model-Checking tool (`umaudemc`) Rubio et al. (2021). Some examples of verified temporal formulas are those expressing properties such as: *Does there exist at least one path where that treatment is successful? Do all paths prevent reaching a steady state in which a AKT is present?*
GROOVE experimentation.

Like Maude, GROOVE has built-in model checking capabilities for both LTL and CTL properties; below, we show how to replicate the results of Ballis et al. (2024), for the four scenarios listed above.

The main challenge in replicating the results is that some of the properties to be checked are formulated in terms of *steady states* of the reaction system, which are essentially

(a) Rule testStart (b) Rule markInput



(c) Condition steady

Fig. 14 Additional rules and condition for detecting steadiness

one-state attractors, that is, states in which the context and reactions together reproduce exactly the entities of that state again. Though GROOVE detects such a loop as a matter of course, it is a structural property of the LTS and not a state property available for model checking. In order to be able to reason about steadiness, we have to remember *input entities*, i.e., entities that were present in the source state, and compare them to *present entities*, i.e., those that have been produced in the target state. Moreover, we should not accidentally mark the start state as steady even if it has neither inputs nor present entities. Figure 14 shows the additional rules that achieve this, together with the modified recipe defined by

```
recipe fire() {
  try testStart; markInput; context; react;
}
```

The resulting steady condition is given (using quantifier syntax) in Fig. 14c.

1. Given the context `[k,ket]`, GROOVE confirms the status of the following LTL properties:
   - `FG(steady -> akt)` is not satisfied; GROOVE produces a counter-example.
   - `G(erbb2 -> X(erbb2))` is satisfied.
2. Given the context `[{e,egf,hrg}.korep]`, GROOVE confirms the status of the following CTL properties:
   - `EF (steady & !akt)` is satisfied;
   - `AF (steady & !akt)` is not satisfied.
3. Given the context `[k,korept]`, GROOVE confirms the status of the following LTL properties:
   - `G(pdk1 -> FG(steady -> akt))` is satisfied;
   - `erbb1 R !pdk1` is satisfied.
4. Given the context `[k,kge]`, GROOVE confirms the status of the following CTL property:
   - `EG EF (steady -> !akt)` is satisfied.

   However, we want to point out that this property does not actually provide any useful guarantees, because the predicate `steady` (both in Ballis et al. (2024) and in our encoding explained above) only tests for *single-state* attractors. If the reaction system ends up in a multi-state loop, `steady` will never be satisfied and hence the implication `steady -> !akt` is *always* satisfied, irregardless of whether or not `akt` holds. Indeed, the state space of this scenario, visually reproduced in Fig. 15, has such a multi-state attractor (consisting of states *s15* and *s18*); in both of those states the predicate `akt` holds, yet the state space as a whole satisfies the CTL property above.

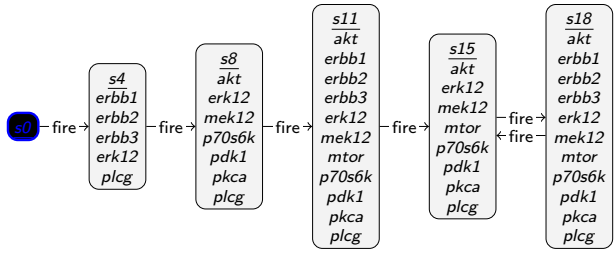


Fig. 15 GTS for cancer scenario 4

In replicating the results from Ballis et al. (2024), we have had to make a few adjustments. The LTL formulas reported in Ballis et al., (2024, Page 14) for scenarios 1 and 3 are actually *not* literally the ones above, but use the predicate `io-state` rather than `isSteady`. We believe that the use of `isSteady` (or, in our case `Steady`) is more informative and probably the intended version.

As a final observation, we note that the numbers of states in all these scenarios is actually quite small. We have already shown the 6-state scenario 4 in Fig. 15; the size of the others is given by the following table.

Nr.	Context	States
1	`[k,ket]`	4
2	`[{e,egf,hrg}.korep]`	10
3	`[k,korept]`	32
4	`[k,kge]`	6

Discussion. Compared to the prior results in Ballis et al. (2024), the advantages of using GROOVE lie in the combination of visual inspection and automatic model checking.

Fig. 16 Graphical representation of the Boolean network model of T cell differentiation from Puniya et al. (2018)

Not only were we able to confirm the findings of the original paper using exactly the same encoding of reaction system as for the previous case, but the ability to inspect and visualise the state spaces also gives additional insights, such as the observation above that steadiness as formalised there does not actually capture the intended notion of being an attractor.

5.3 T cell differentiation analysis

The paper (Brodo et al. 2025) (being the full and corrected version of Brodo et al. (2025), see also Footnote 5) exploits Reaction Systems to analyse T cell differentiation in the immune system, a widely studied biological phenomenon. The starting point for the analysis is a Boolean network model; several of those are available as a Saez-Rodriguez et al. (2007); Thakar and Albert (2010); Puniya et al. (2018), among which the one in Puniya et al. (2018) was selected. The model encompasses reactions enabling T cells to manifest various phenotypes in response to environmental stimuli, and describes a realistic regulation system that is involved in many diseases (Lafaille 1998; Hirahara and Nakayama 2016; Meng et al. 2016).

The Boolean network model is graphically represented as shown in Fig. 16, where the 9 orange nodes represent different environmental stimuli that the T cell can receive; all the other nodes represent so-called *transcription factors* and have an associated Boolean update formula that specifies when they are triggered. T cells can manifest four phenotypes, represented by the four transcription factors `tbet`, `gata3`, `rorgt` and `foxp3`, respectively. There exists experimental and computational evidence that a T cell can manifest more than one phenotype (Luckheeram et al. 2012; Puniya et al. 2018).

Analysis goals. The reachability analysis must take into account the different combinations of phenotypes that a T cell can express, called a *target* (hence, $2^4 = 16$ targets overall). For example, for the target containing the combination of transcription factors {`tbet`, `gata3`}, we must select an attractor that includes at least one state in which `tbet` is present, at least one state (possibly the same) in which `gata3` is present, and no state in which either `rorgt` or `foxp3` are present. The causal analysis aims to collect the combinations of environmental stimuli that are responsible for leading to that target.

Features of interest. We have selected this case study because it shows the applicability of our method to Boolean networks models, like those available in the public database on the CellCollective platform (Helikar et al. 2012). For these models, the most relevant viewpoints are often reachability (e.g., *which phenotypes are reachable?*) and causality (*what is the effect of environmental conditions?*) analyses. Their corresponding RSs always use a special kind of nondeterministic persistent context, where at the beginning of the experiment a subset of external stimuli is chosen and then provided at each subsequent step, inevitably causing the RS to end up in a loop (called an attractor).

Experimental set up. The translation from Boolean networks to RS consists in turning every update formula into disjunctive normal form. Then, every clause of the disjunction produces a reaction in which (i) reactants are the positive atoms, (ii) inhibitors are the negated atoms and (iii) the updated variable forms a singleton product. The translation from Boolean network to BioResolve syntax is done using the directive `main_do(bn2rs)`. For the readers' convenience, all update formulas and the resulting reactions are reported, respectively, in Fig. 22 and in Fig. 24 in the Appendix. For example, the update formula for `IL12R` is

```
(IL12&NFAT) | (STAT4&¬GATA3)
  | Tbet | (TCR&¬GATA3)
```

which yields the four reactions[5]

```
({il12, nfat}, ∅, {il12r})   ({stat4}, {gata3}, {il12r})
  ({tbet}, ∅, {il12r})       ({tcr}, {gata3}, {il12r})
```

The RS context can choose any combination of environmental stimuli that will then persist, i.e., for each possible stimulus s we define the context processes $\mathsf{X}_s \triangleq \{s\}.\mathsf{X}_s$ and then take the context $\prod_s (\mathsf{X}_s + \mathsf{Emp})$. The resulting LTS has an initial branching into 2^9 different states, because there are 9 possible stimuli to be considered. Subsequently, each of the 2^9 states originates a deterministic computation, leading to some attractors.

Previous approach. The paper (Brodo et al. 2025) presents a toolchain (BioResolve, SWI-Prolog, Python and Python-to-Prolog binding facilitated by the `swiplserver` Python

[5] The specification analysed in Brodo et al. (2025), on which this paper is based, repairs a minor typo in the original conference version (Brodo et al. 2025). Specifically, the product set of the reaction (`stat4`, `gata3`, `il12r`) was mistakenly written as `il2r`, omitting the digit `1`. Unfortunately, since `il2r` was also a valid entity, the error was difficult to detect. Though the mistake mildly affected the original results, it turns out that for the analysis reported in this paper there is no difference at all.:

package) to study the Boolean network model. Roughly, after translating the Boolean network model to RS specifications the whole LTS is constructed according to any combination of persistent stimuli that can be provided by the context. BioResolve returns the LTS as a graph in dot format, which is then loaded by a Python script. Then, attractors related with a target of interest are identified by looking for cycles in the LTS, and a slicing algorithm performs some form of causal analysis, to simplify each computation trace by preserving only the relevant causes of those target entities. The generation of the LTS is often the bottleneck of the approach, both in terms of time (Prolog performance), but also in terms of space, because BioResolve can require to allocate a large stack limit size to succeed.

GROOVE experimentation. The capabilities of GROOVE called upon for this case study are very similar to those in Sect. 5.1; the main difference lies in the specific interest in attractors. Indeed, in contrast to the situation for comorbidities, here after the initial selection of a profile, the context does not cause any more nondeterminism; hence every profile eventually ends up in such an attractor.

A trace ending in a loop, sometimes called a "lollipop", is in fact precisely what LTL properties are checked over; hence an LTL property violation takes the form of a lollipop. This means that we can find attractors with specific properties by formulating their non-existence in LTL, and then finding a counter-example through model checking. For instance, the following formulas deny the reachability of an attractor in which `tbet` and `gata3` are expressed:

- `!G(F gata3 & F tbet)` (separate expression)
- `!GF (gata3 & tbet)` (simultaneous expression)

Using the start graph derived from BioResolve using the process outlined in Fig. 5, both of these formulas yield counterexamples, meaning that `tbet` and `gata3` *can* in fact be (recurrently) expressed simultaneously. Using a variation of the process outlined in Sect. 4, we can once more visualise a trace leading to such a recurrent state. The variation lies in the fact that, this time, we do not want to show the causal history of a *single* forbidden entity, but rather of the combination of two distinct entities. Fortunately, this is just a matter of creating another rule, `gata3-tbet`, which applies precisely when `gata3 & tbet` holds. On this basis we can go through the steps outlined in Fig. 5, resulting in the occurrence graph displayed in Fig. 17.

This complements the observation embodied in Brodo et al., (2025, Fig. 7) that for the combination of `tbet` and `gata3`, the context has to provide the stimuli `ifnge` (produced here by the **StepOcc** named `x81_0`), `tcr` (repeatedly produced by **StepOcc**s named `x91_0`) and `il4e` (also repeatedly produced, by **StepOcc**s named `x51_0`). In more detail, we see that `tbet` derives, in a linear sequence of four **ReactionOcc**s, from `ifnge` and `tcr`, whereas `gata3` derives, in 3 successive combinations of simultaneous **ReactionOcc**s, from `tcr` and `il4e`. Moreover, the genes produced along the way are precisely the ones reported in Brodo et al., (2025, Fig. 8(a)), using the slicing algorithm of that paper, as being relevant for the expression of `tbet` and `gata3`.

Besides the production of such occurrence graphs for specific cases, GROOVE can also be used once more to directly confirm the findings of Brodoet al., (2025, Figs. 7 and 8), by expressing them as CTL formulas similar to the one reported in Fig. 13. In this context, it is relevant to report that, in contrast to BioResolve, where (as reported above) time and space performance were a bottleneck in the analysis of this model, GROOVE can fully explore the state space in approximately 3 s.

Discussion Compared to the results in Brodo et al. (2025), the advantages of GROOVE are threefold (reiterating the observations made for the previous two cases):

- LTL model checking allows to express, in a flexible manner, the scenarios one wants to investigate;
- The occurrence graph visualisation offers analysis possibilities beyond the outcome of the slicing algorithm;
- The performance of GROOVE is an order of magnitude better than that of BioResolve

6 Comparison with existing tools

The approach presented in this paper can provide several advantages over existing tools in the literature.[6]

- `brsim`[7] (Basic Reaction System Simulator, written in Haskell and distributed under the terms of GNU GPLv3 license) Azimi et al. (2015) was the first RS simulator to be made publicly available. Given the reactions of the RS and a context sequence, `brsim` is capable of computing the resulting sequence and generating additional annotations for each computation step, such as the enabled reactions. Alternatively, `brsim` can be executed in an interactive mode, allowing the user to manually provide the context to be used at each step.
- WebRSim[8] is a basic RS simulator that makes all functionalities of `brsim` available through a friendly web interface (Ivanov et al. 2018);

[6] See, e.g., the list of Reaction Systems Computer Environments at https://www.reactionsystems.org/about-reaction-systems.

[7] Available at https://github.com/scolobb/brsim/.

[8] Available at https://github.com/scolobb/brsim.

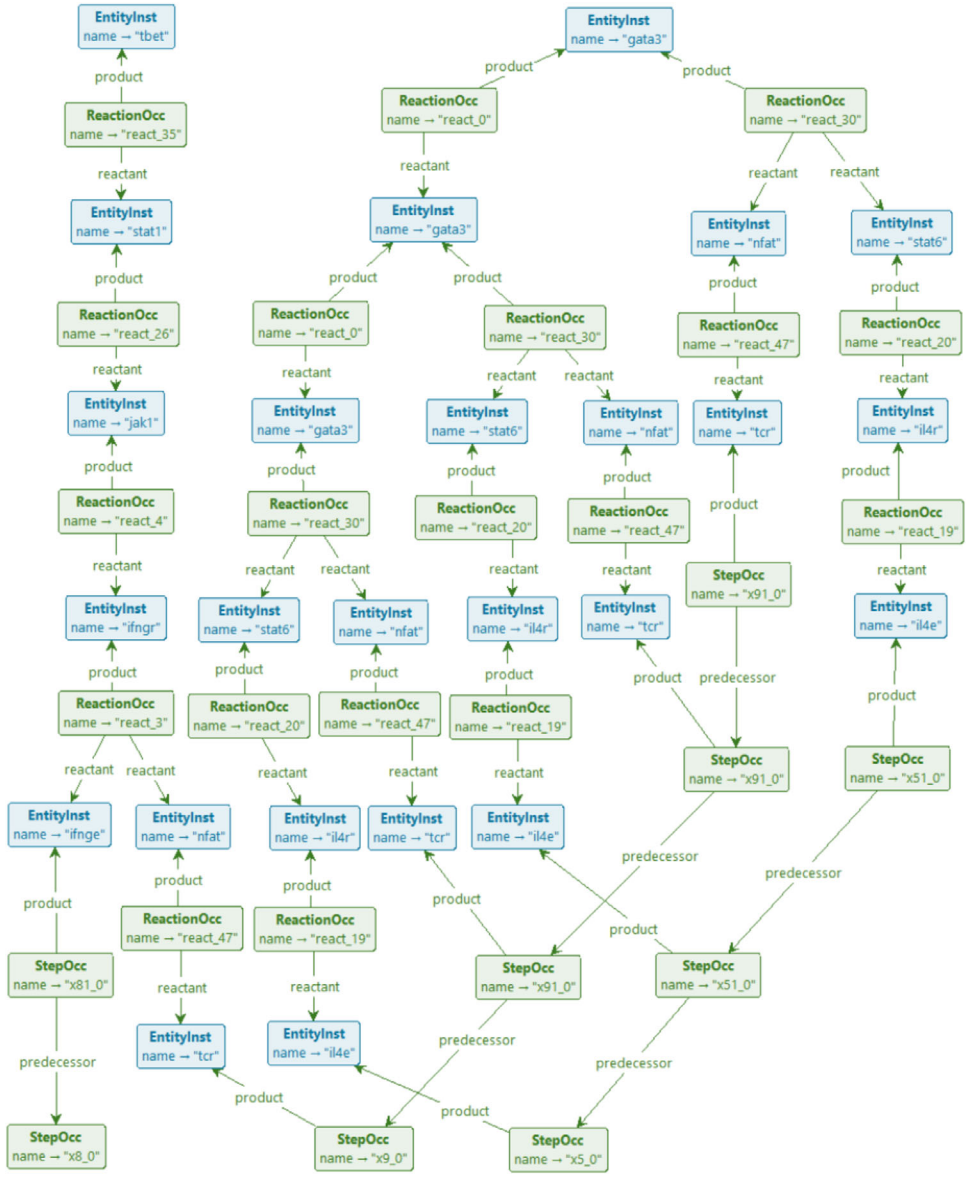


Fig. 17 Occurrence graph for the simultaneous expression of `gata3` and `tbet`

- HERESY[9] is a Highly Efficient REaction SYstem GPU-based simulator, developed using CUDA (Nobile et al. 2017). It features a user-friendly GUI and is designed to exploit the high degree of parallelism offered by modern GPUs to handle very large-scale RSs simulations.
- `cl-rs`[10] is an optimised Common Lisp simulator for RSs (Ferretti et al. 2020) that can exhibit performances comparable with the GPU-based simulator HERESY. This is achieved by discarding all reactions that cannot produce effects and by encoding RS evolution in terms of matrix–vector multiplications and vector additions.
- BioResolve[11] is a Prolog interpreter for Reaction System analysis, first proposed in Brodo et al. (2021) and later extended in a series of papers to deal with enhanced features, like delays, duration, monitoring, slicing and guarded contexts (Brodo et al. 2023, 2024a; Bowles et al. 2024). Many capabilities of BioResolve have been discussed at length in the previous sections.

[9] Available at https://github.com/aresio/HERESY.

[10] Available at https://github.com/mnzluca/cl-rs.

[11] Available at https://www.di.unipi.it/~bruni/LTSRS/.

- **ccReact**[12] is an interacting language for Reaction Systems based on Maude 3.2.1 (Ballis et al. 2024), whose key features have been illustrated in Sect. 5.2.
- ReactICS[13] is a Reaction Systems Verification Toolkit that consists of two main modules for model checking temporal properties expressed in logical languages tailored to Reaction Systems: one that exploits binary decision diagrams (BDD) and bounded model checking; the other that translates verification problems in rsLTL (Meski et al. 2015) into satisfiability modulo theories (SMT) (Meski et al. 2024).

Modelling capabilities. The GROOVE-based method supports a rich and expressive encoding of RSs, including the most recent features such as the handling of *guarded, recursive, and nondeterministic contexts*. Among the other tools, such features are only supported by BioResolve, which, however, relies on a Prolog back-end that limits scalability and requires external scripting for improving the performance of many analyses whenever large state generation and exploration is necessitated. Tools such as HERESY, WebRSim, and `cl-rs` provide lightweight RS simulators but are limited to basic semantics, lacking support for more advanced interactions with the context or advanced verification features. **ccReact** supports temporal logic model checking (LTL/CTL), but not recursive contexts. Moreover, the encoding of RSs in **ccReact** is manual and less suited to visual inspection or dynamic causal analysis. In addition to (bounded) model checking of custom temporal logics specifically designed for Reaction Systems, ReactICS also supports context automata (a slightly less general notion of contexts than the one considered here), reactions with concentration levels, parameter and reaction synthesis, as well as parameter optimisation.

Performance and scalability. The ability of GROOVE to explore large state spaces efficiently is central to our method. Through configurable exploration strategies and a rule-based control mechanism, GROOVE handles complex RS instances that involve thousands of reachable configurations. Our experiments demonstrate a substantial improvement in analysis time compared to BioResolve, often reducing execution time by an order of magnitude. Furthermore, the performance of BioResolve is strongly influenced by the nature of Prolog evaluation strategies, which can lead to excessive memory and time consumption in large case studies. With respect to **ccReact**, the paper (Ballis et al. 2024) on which we based our experiments does not provide information on performance, and indeed the system studied there is so small that no useful comparison could be made on that basis. Since Maude, underlying **ccReact**, is a long-standing and mature tool, it would be interesting to investigate this in more detail; however, we leave this to future work. In contrast, other tools either consider linear executions only and do not scale to large models or lack optimisation strategies necessary for handling non-trivial state spaces.

Contrary to GROOVE, which is a general purpose, explicit-state tool, not optimised for Reaction Systems, ReactICS can take advantage of symbolic representations to abstract much larger state-spaces. For instance, for the mutual exclusion protocol reported in Meski et al. (2015), Nobile et al. (2017), ReactICS can cope with 37 to 54 processes (depending on the precise analysis) in a matter of hours; in contrast, in comparable time GROOVE can exhaustively generate the state space of up to 13 processes. Even though the model checking queries in ReactICS probably do not require full state space exploration, and hence the comparison is skewed, the dedicated methods of ReactICS clearly pay off.

Causal analysis and verification. A key distinguishing feature of our approach is the ability to perform graph-based *causal slicing*. By automatically generating and pruning *occurrence graphs*, GROOVE provides detailed and visual explanations of how specific states, such as those involving undesirable or forbidden entities, are reached. This form of causal reasoning is not available in high-performance tools such as HERESY, WebRSim, `cl-rs`, or ReactICS and is only partially addressed in **ccReact**, where the focus is primarily on reachability and temporal properties. GROOVE's integrated support for CTL and LTL model checking further extends its applicability to behavioural verification, enabling the specification and validation of complex temporal properties.

Summary. In conclusion, the combination of expressive modelling, efficient state space exploration, and integrated causal analysis makes GROOVE a powerful and versatile full-fledged platform for the study of Reaction Systems. It not only generalises and extends existing tools, but also opens the door to new forms of analysis that were previously impractical or unsupported.

7 Conclusion and future work

In this work, we have demonstrated how Reaction Systems can be effectively encoded and analysed within the GROOVE framework, so to reuse the expressiveness and efficiency of graph transformation techniques. By exploiting quantified rules, the encoding consists of a direct translation from RS specification to a typed graph, which is made automatic in BioResolve. Then, GROOVE enables both exhaustive state space exploration, the extraction of causal information

[12] Available at https://depot.lipn.univ-paris13.fr/olarte/reaction-systems-maude.

[13] Available at https://github.com/arturmeski/reactics/.

through occurrence graphs and property-based verification based on model checking.

We have used GROOVE to revisit several case studies from the literature and our experimental results, although preliminary, are promising: GROOVE not only supports complex RS features such as guarded, nondeterministic and recursive contexts but also significantly improves performance and flexibility compared to existing RS tools. Moreover, the use of GROOVE's recipes and model-checking capabilities opens the door to sophisticated analyses that were previously impractical.

A number of interesting avenues for future research remain open, among which we mention the possibility to extend the methodology to support quantitative RS variants with durations or weights (Brodo et al. 2023); investigate alternative notions of causality and their representation in graph-based semantics, like dependencies drawn from inhibitors rather than reactants[14]; apply the GROOVE toolset to further biological case studies, like those available in the CellCollective public repository (Helikar et al. 2012), possibly exploiting an automated pipeline for analysis.

Overall, the results show that graph transformation, and GROOVE in particular, provide a robust and scalable foundation for the specification, execution, and analysis of Reaction Systems.

One option that we have ignored throughout the paper deserves a brief mention here. The slicing algorithms reported in (Bowles et al. 2024; Brodo et al. 2025) on the basis of BioResolve are actually implemented as stand-alone scripts that operate on a `.dot`-formatted LTS. Given that GROOVE can also produce LTSs as `.dot` files, an alternative way to benefit from its superior performance might be merely to replace BioResolve in the tool chain used previously. For this to be possible, the LTS labelling information produced by GROOVE should conform to the requirements of the slicing algorithm, following the principles outlined in Remark 1. Though that is currently not the case (compare GROOVE's Fig. 8 to BioResolve's Fig. 4), we believe that this requires only a minor adjustment of the rule system.

From the perspective of GROOVE development, carrying out the experiments described in this paper has led to many small improvements as well as inspiration for new features. For instance, the various strategies for "ordinary" state space exploration (DFS- or BFS based, bounded or not, conditional or not) on the one hand hand and the model checking capabilities on the other are not well-integrated. Queries such as "count the number of states of max depth n where a given CTL property holds" or "find all prefixes of length n of paths satisfying a given LTL property," which would enhance the capabilities for analysing graph-based models such as the ones studied here, currently cannot be posed in a straightforward manner. Another useful extension would be the ability to automatically chain different transformations, such as the three steps of explore–build–prune in Fig. 5, which currently have to be invoked separately. We plan to investigate these extensions in the future.

[14] In this respect, we could, e.g., exploit the semantic-preserving transformation from RSs to Positive RSs proposed in Brodo et al. (2024b).

Supplementary information

For replication of the GROOVE experiments, we provide an archive with supplementary material, described in Sect. C.5.

Appendix A Semantic correspondence

In this paper, we claim, but do not prove, that the GROOVE encoding correctly captures the operational Reaction Systems semantics recalled in Sect. 2.1. A complete formal proof is outside the scope of this paper, which focusses on experimental results; however, in this section we provide a sketch of such a proof.

As described in Sect. 4, each GROOVE state is a graph consisting of a fixed part (which is the same in every state) and a variable part. The fixed part contains an encoding of (i) the reaction system A itself, with the entities S as **Entity**-typed nodes, and (ii) the automata of the context processes, with individual processes corresponding to **Token**-typed nodes and states as **State**-typed nodes.

The variable part consisting of *present* flags on **Entity** nodes and a current-edge from every **Token** to a **State**. As observed in Remark 1, any RS process [M] can be written in the form [Rs | Ks | D] where: (i) $\mathsf{Rs} = \prod_i (R_i, I_i, P_i)$ is the parallel composition of all reactions in the system, and never changes along the computation; (ii) $\mathsf{Ks} = \prod_j \mathsf{K}_j$ is the parallel composition of all contexts, and (iii) D is the set of currently present entities.

A GROOVE state G is equivalent to an RS process [Rs | Ks | D] if: (i) an entity e is present in D if and only if its **Entity** node is flagged as *present* in G, and (ii) there is a one-to-one correspondence of the available contexts K_j in the RS process and the **Token**-nodes in G, such that the current state of that **Token** is the start state of the automaton for K_j.

We claim that this equivalence establishes a bisimulation between the GROOVE and RS state spaces. (It is not an isomorphism, because the RS semantics for K "consumes" the process (rules *(Cxt)–(Rec)* in Fig. 2) whereas G keeps the automaton intact, merely moving the current pointer.)

To prove this claim, we have to show correspondence of the transitions. Concretely, to any transition carrying the label $\langle\langle D \rhd R', I', C\rangle \rhd R, I, P\rangle$, there corresponds a single

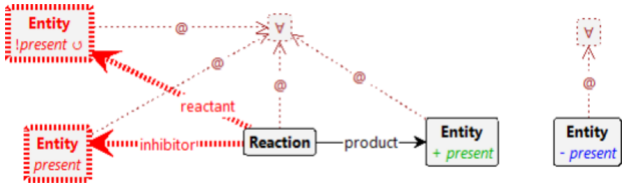


Fig. 18 Rule for reaction firing

application of the recipe fire, which in turn consists of applications of rules context (see Fig. 7) followed by react (see Fig. 18).

Both of these rules are universally quantified. Roughly speaking, referring to Fig. 2, context simultaneously encodes all sub-transitions of the form

$$D \xrightarrow{\langle\langle D \triangleright \emptyset,\emptyset,\emptyset\rangle \triangleright \emptyset,\emptyset,\emptyset\rangle} D' \quad \text{(rule } (Ent)\text{)}$$

$$\mathsf{K} \xrightarrow{\langle\langle \emptyset \triangleright R',I',C\rangle \triangleright \emptyset,\emptyset,\emptyset\rangle} \mathsf{K}' \quad \text{(rules } (Cxt)\text{–}(Rec)\text{)}$$

as well as their composition and filtering (rules (Par)–(Sys)). react in turns simultaneously encodes all sub-transitions of the form

$$(R, I, P) \xrightarrow{\langle\langle \emptyset \triangleright \emptyset,\emptyset,\emptyset\rangle \triangleright R',I',P'\rangle} \mathsf{M}' \quad \text{(rules } (Pro)\text{–}(Inh)\text{)}$$

together with their composition with the context-transitions (rule (Par)) and filtering (rule (Sys)). (The *fired*-flags on the **Step**-nodes, added by context and removed again by react, play no role in this correspondence: for the purpose of the equivalence of the GROOVE and RS semantics, they might be omitted entirely.)

Appendix B Mutual exclusion

Table 1 shows the GROOVE performance in generating the full state space of the mutual exclusion example of Meski et al. (2015), Nobile et al. (2017), discussed in Sect. 6.

Table 1 State space generation for mutual exclusion

Case	States		Time		Memory	
#	#	Ratio	s	Ratio	MB	Ratio
2	11		0.2		1	
3	27	2.5	0.4	1.9	2	2.2
4	63	2.3	0.7	2.0	4	2.1
5	143	2.3	2.0	2.8	13	3.0
6	319	2.2	5.6	2.7	34	2.6
7	703	2.2	17.5	3.2	96	2.8
8	1535	2.2	56.6	3.2	262	2.7
9	3327	2.2	175.0	3.1	700	2.7
10	7167	2.2	595.3	3.4	1,929	2.8
11	15359	2.1	1,890.3	3.2	4,764	2.5
12	32767	2.1	5,146.1	2.7	8,588	1.8
13	69631	2.1	15,647.3	3.0	13,640	1.6

```
myentities([cpowder,tpowder]). % initial set D0

myreactions([                  % list of reactions
    react([idle],[am],[am]),
    react([am],[idle],[am]),
    react([ccoin,cpowder],[nomilk],[cappuccino]),
    react([ccoin,cpowder,nomilk],[],[espresso]),
    react([tcoin,tpowder],[],[tea]),
    react([cpowder],[],[cpowder]),
    react([tpowder],[],[tpowder]),
    react([anger],[],[bang]) ]).

mycontext("[refill,student]"). % context processes

myenvironment("[               % context definitions
    refill = ({nomilk}.refill
            + {}.refill),
    student = (?{},{am},{tcoin}?.gettea
             + ?{am},{},{ccoin}?.getcappuccino
             + {idle}.student),
    gettea = (?{tea},{},{}?.student
            + ?{},{tea},{anger}?.student),
    getcappuccino = (?{cappuccino},{},{}?.student
                   + ?{espresso},{},{anger}?.student) ]").
```

Fig. 19 BioResolve implementation of the vending machine RS from Sect. 3. The question marks ? are used to delimit guarded prefixes in context processes

Appendix C Auxiliary material

C.1 Auxiliary material for the toy running example

The BioResolve specification for the toy running example about the interaction between the student and the vending machine is reported in Fig. 19. The corresponding RS has been described in Sect. 3 and it has been used to illustrate some key features of the GROOVE encoding in Sect. 4.

Feats ≜ ({hyper}, ∅, {hyper}) | ({afib}, ∅, {afib}) | ({has_fib}, ∅, {has_fib}) | ({heart_rate}, ∅, {heart_rate}) | ({consensus_acei}, ∅, {consensus_acei})
| ({over75}, ∅, {over75}) | ({below55}, ∅, {below55}) | ({diabete}, ∅, {diabete}) | ({origin}, ∅, {origin})
| ({doac_int}, ∅, {doac_int}) | ({hyper}, ∅, {diseases}) | ({diabete}, ∅, {diseases})

Drugs ≜ ({get_diltiazem}, {stop_cbb}, {diltiazem, cbb}) | ({diltiazem}, {stop_cbb}, {diltiazem, cbb}) | ({get_verapamil}, {stop_cbb}, {verapamil, cbb})
| ({verapamil}, {stop_cbb}, {verapamil, cbb}) | ({diltiazem, verapamil}, {stop_cbb}, {alert_dup}) | ({get_propranolol}, {stop_nsbb}, {propranolol, nsbb})
| ({propranolol}, {stop_nsbb}, {propranolol, nsbb}) | ({get_carvedilol}, {stop_nsbb}, {carvedilol, nsbb}) | ({carvedilol}, {stop_nsbb}, {carvedilol, nsbb})
| ({propranolol, carvedilol}, {stop_nsbb}, {alert_dup}) | ({get_bisoprolol}, {stop_sbb}, {bisoprolol, sbb}) | ({bisoprolol}, {stop_sbb}, {bisoprolol, sbb})
| ({get_atenolol}, {stop_sbb}, {atenolol, sbb}) | ({atenolol}, {stop_sbb}, {atenolol, sbb}) | ({bisoprolol, atenolol}, {stop_sbb}, {alert_dup})
| ({get_flecainide}, {stop_flec}, {flecainide}) | ({flecainide}, {stop_flec}, {flecainide}) | ({get_warfarin}, {stop_warf}, {warfarin})
| ({warfarin}, {stop_warf}, {warfarin}) | ({get_apixaban}, {stop_doac}, {apixaban, doac}) | ({apixaban}, {stop_doac}, {apixaban, doac})
| ({get_dabigatran}, {stop_doac}, {dabigatran, doac}) | ({dabigatran}, {stop_doac}, {dabigatran, doac}) | ({apixaban, dabigatran}, {stop_doac}, {alert_dup})
| ({get_vkant}, {stop_vkant}, {vkant}) | ({vkant}, {stop_vkant}, {vkant}) | ({get_benazepril}, {stop_acei}, {benazepril, acei})
| ({benazepril}, {stop_acei}, {benazepril, acei}) | ({get_captopril}, {stop_acei}, {captopril, acei}) | ({captopril}, {stop_acei}, {captopril, acei})
| ({benazepril, captopril}, {stop_acei}, {alert_dup}) | ({get_olmesortan}, {stop_arb}, {olmesortan, arb}) | ({olmesortan}, {stop_arb}, {olmesortan, arb})
| ({get_irbesartan}, {stop_arb}, {irbesartan, arb}) | ({irbesartan}, {stop_arb}, {irbesartan, arb}) | ({olmesortan, irbesartan}, {stop_arb}, {alert_dup})
| ({get_indapamide}, {stop_td}, {indapamide, td}) | ({indapamide}, {stop_td}, {indapamide, td}) | ({get_chlorothiazide}, {stop_td}, {chlorothiazide, td})
| ({chlorothiazide}, {stop_td}, {chlorothiazide, td}) | ({indapamide, chlorothiazide}, {stop_td}, {alert_dup}) | ({doac}, {doac_ok, doac_fail}, {doac_test})
| ({doac_ok}, {doac_fail}, {doac_ok}) | ({doac_fail}, {doac_ok}, {doac_fail}) | ({doac}, {doac_fail, stop_doac}, {doac_danger})
| ({doac}, {doac_danger, stop_doac}, {danger})

ADR ≜ ({get_apixaban, get_diltiazem}, ∅, {moderate}) | ({get_apixaban, diltiazem}, ∅, {moderate}) | ({apixaban, get_diltiazem}, ∅, {moderate})
| ({apixaban, diltiazem}, ∅, {moderate}) | ({get_apixaban, get_verapamil}, ∅, {moderate}) | ({get_apixaban, verapamil}, ∅, {moderate})
| ({apixaban, get_verapamil}, ∅, {moderate}) | ({apixaban, verapamil}, ∅, {moderate}) | ({get_dabigatran, get_diltiazem}, ∅, {moderate})
| ({get_dabigatran, diltiazem}, ∅, {moderate}) | ({dabigatran, get_diltiazem}, ∅, {moderate}) | ({dabigatran, diltiazem}, ∅, {moderate})
| ({get_dabigatran, get_verapamil}, ∅, {major}) | ({get_dabigatran, verapamil}, ∅, {major}) | ({dabigatran, get_verapamil}, ∅, {major})
| ({dabigatran, verapamil}, ∅, {major}) | ({get_dabigatran, get_carvedilol}, ∅, {moderate}) | ({get_dabigatran, carvedilol}, ∅, {moderate})
| ({dabigatran, get_carvedilol}, ∅, {moderate}) | ({dabigatran, carvedilol}, ∅, {moderate}) | ({get_warfarin, get_benazepril}, ∅, {minor})
| ({get_warfarin, benazepril}, ∅, {minor}) | ({warfarin, get_benazepril}, ∅, {minor}) | ({warfarin, benazepril}, ∅, {minor})
| ({get_warfarin, get_indapamide}, ∅, {minor}) | ({get_warfarin, indapamide}, ∅, {minor}) | ({warfarin, get_indapamide}, ∅, {minor})
| ({warfarin, indapamide}, ∅, {minor}) | ({get_warfarin, get_chlorothiazide}, ∅, {minor}) | ({get_warfarin, chlorothiazide}, ∅, {minor})
| ({warfarin, get_chlorothiazide}, ∅, {minor}) | ({warfarin, chlorothiazide}, ∅, {minor}) | ({get_warfarin, get_propranolol}, ∅, {minor})
| ({get_warfarin, propranolol}, ∅, {minor}) | ({warfarin, get_propranolol}, ∅, {minor}) | ({warfarin, propranolol}, ∅, {minor})
| ({get_flecainide, get_diltiazem}, ∅, {major}) | ({get_flecainide, diltiazem}, ∅, {major}) | ({flecainide, get_diltiazem}, ∅, {major})
| ({flecainide, diltiazem}, ∅, {major}) | ({get_flecainide, get_verapamil}, ∅, {major}) | ({get_flecainide, verapamil}, ∅, {major})
| ({flecainide, get_verapamil}, ∅, {major}) | ({flecainide, verapamil}, ∅, {major}) | ({get_flecainide, get_bisoprolol}, ∅, {moderate})
| ({get_flecainide, bisoprolol}, ∅, {moderate}) | ({flecainide, get_bisoprolol}, ∅, {moderate}) | ({flecainide, bisoprolol}, ∅, {moderate})
| ({get_flecainide, get_atenolol}, ∅, {moderate}) | ({get_flecainide, atenolol}, ∅, {moderate}) | ({flecainide, get_atenolol}, ∅, {moderate})
| ({flecainide, atenolol}, ∅, {moderate}) | ({get_flecainide, get_propranolol}, ∅, {moderate}) | ({get_flecainide, propranolol}, ∅, {moderate})
| ({flecainide, get_propranolol}, ∅, {moderate}) | ({flecainide, propranolol}, ∅, {moderate}) | ({get_flecainide, get_carvedilol}, ∅, {moderate})
| ({get_flecainide, carvedilol}, ∅, {moderate}) | ({flecainide, get_carvedilol}, ∅, {moderate}) | ({flecainide, carvedilol}, ∅, {moderate})
| ({major}, ∅, {major}) | ({moderate}, ∅, {moderate}) | ({minor}, ∅, {minor}) | ({alert_dup}, ∅, {alert_dup}) | ({danger}, ∅, {danger})

Fig. 20 Reactions for the comorbidity case study in 5.1.

C.2 Auxiliary material for the comorbidity case study

The RS specification for the comorbidity case study presented in Bowles et al. (2024) is reported in Fig. 20 (set of reactions) and Fig. 21 (context processes definitions), where we assume the initial state is $D_0 = \varnothing$. The corresponding experimentation with GROOVE has been discussed in Sect. 5.1.

C.3 Auxiliary material for the protein signaling networks case study

The BioResolve specification for the protein signaling networks case study presented in Ballis et al. (2024) is reported in Fig. 22. The corresponding experimentation with GROOVE has been discussed in Sect. 5.2.

```
eafib1 ≜ (∅, {afib}, ∅).eafib1 + ({afib}, ∅, ∅).ehr
ehr ≜ (∅, {heart_rate}, ∅).ehr + ({heart_rate}, ∅, ∅).ebb
ebb ≜ ∅.ebb + e_cbb + e_nsbb + e_sbb
e_cbb ≜ (∅, {verapamil}, {get_diltiazem}).empty + (∅, {diltiazem}, {get_verapamil}).empty
e_nsbb ≜ (∅, {carvedilol}, {get_propranolol}).empty + (∅, {propranolol}, {get_carvedilol}).empty
e_sbb ≜ (∅, {atenolol}, {get_bisoprolol}).empty + (∅, {bisoprolol}, {get_atenolol}).empty
eafib2 ≜ (∅, {afib}, ∅).eafib2 + ({afib}, ∅, ∅).ehf
ehf ≜ (∅, {has_fib}, ∅).ehf + ({has_fib}, ∅, ∅).eflec
eflec ≜ ∅.eflec + e_flec
e_flec ≜ {get_flecainide}.empty
eafib3 ≜ (∅, {afib}, ∅).eafib3 + ({afib}, ∅, ∅).econs
econs ≜ (∅, {heart_rate, has_fib}, ∅).econs + (∅, {consensus_acei}, ∅).econs + ({consensus_acei, heart_rate}, ∅, ∅).estroke + ({consensus_acei, has_fib}, ∅, ∅).estroke
estroke ≜ (∅, {diseases, over75}, ∅).ewarf + ({over75}, {doac_fail, doac_int}, ∅).edoac + ({diseases}, {doac_fail, doac_int}, ∅).edoac + ({over75, doac_fail}, ∅, ∅).evkant
        + ({over75, doac_int}, ∅, ∅).evkant + ({diseases}, {doac_fail, ∅, ∅).evkant + ({diseases}, {doac_int, ∅, ∅).evkant
ewarf ≜ ∅.ewarf + e_warf
e_warf ≜ {get_warfarin}.empty
edoac ≜ ∅.edoac + e_doac
e_doac ≜ (∅, {dabigatran}, {get_apixaban}).e_doacfail + (∅, {apixaban}, {get_dabigatran}).e_doacfail
e_doacfail ≜ ({doac_fail}, ∅, {stop_doac}).evkant + (∅, {doac_fail}, ∅).e_doacfail
evkant ≜ ∅.evkant + e_vkant
e_vkant ≜ {get_vkant}.empty
ghyper ≜ (∅, {hyper}, ∅).ghyper + ({hyper}, ∅, ∅).g1
g1 ≜ ({diabete}, ∅, ∅).g2 + ({below55}, {diabete, origin}, ∅).g2 + (∅, {below55, diabete}, ∅).g3 + ({origin}, {diabete}, ∅).g3
g2 ≜ ∅.g2 + (∅, {captopril}, {get_benazepril}).g4 + (∅, {benazepril}, {get_captopril}).g4 + (∅, {irbesartan}, {get_olmesortan}).g5
   + (∅, {olmesortan}, {get_irbesartan}).g5
g3 ≜ ∅.g3 + (∅, {verapamil}, {get_diltiazem}).g6 + (∅, {diltiazem}, {get_verapamil}).g6
g4 ≜ ∅.g4 + (∅, {verapamil}, {get_diltiazem}).g7 + (∅, {diltiazem}, {get_verapamil}).g7 + (∅, {chlorothiazide}, {get_indapamide}).g8
   + (∅, {indapamide}, {get_chlorothiatzide}).g8
g5 ≜ ∅.g5 + (∅, {verapamil}, {get_diltiazem}).g9 + (∅, {diltiazem}, {get_verapamil}).g9 + (∅, {chlorothiazide}, {get_indapamide}).g10
   + (∅, {indapamide}, {get_chlorothiatzide}).g10
g6 ≜ ∅.g6 + (∅, {captopril}, {get_benazepril}).g7 + (∅, {benazepril}, {get_captopril}).g7 + (∅, {irbesartan}, {get_olmesortan}).g9
   + (∅, {olmesortan}, {get_irbesartan}).g9 + (∅, {chlorothiazide}, {get_indapamide}).g11 + (∅, {indapamide}, {get_chlorothiatzide}).g11
g7 ≜ ∅.g7 + (∅, {irbesartan}, {get_olmesortan}).etd + (∅, {olmesortan}, {get_irbesartan}).etd + (∅, {chlorothiazide}, {get_indapamide}).earb
   + (∅, {indapamide}, {get_chlorothiatzide}).earb
g8 ≜ ∅.g8 + (∅, {irbesartan}, {get_olmesortan}).ecbb + (∅, {olmesortan}, {get_irbesartan}).ecbb + (∅, {verapamil}, {get_diltiazem}).earb
   + (∅, {diltiazem}, {get_verapamil}).earb
g9 ≜ ∅.g9 + (∅, {captopril}, {get_benazepril}).etd + (∅, {benazepril}, {get_captopril}).etd + (∅, {chlorothiazide}, {get_indapamide}).eacei
   + (∅, {indapamide}, {get_chlorothiatzide}).eacei
g10 ≜ ∅.g10 + (∅, {captopril}, {get_benazepril}).ecbb + (∅, {benazepril}, {get_captopril}).ecbb + (∅, {chlorothiazide}, {get_indapamide}).eacei
    + (∅, {indapamide}, {get_chlorothiatzide}).eacei
g11 ≜ ∅.g11 + (∅, {captopril}, {get_benazepril}).earb + (∅, {benazepril}, {get_captopril}).earb + (∅, {irbesartan}, {get_olmesortan}).eacei
    + (∅, {olmesortan}, {get_irbesartan}).eacei
ecbb ≜ ∅.ecbb + e_cbb
eacei ≜ ∅.eacei + e_acei
e_acei ≜ (∅, {captopril}, {get_benazepril}).empty + (∅, {benazepril}, {get_captopril}).empty
earb ≜ ∅.earb + e_arb
e_arb ≜ (∅, {irbesartan}, {get_olmesortan}).empty + (∅, {olmesortan}, {get_irbesartan}).empty
etd ≜ ∅.etd + e_td
e_td ≜ (∅, {chlorothiazide}, {get_indapamide}).empty + (∅, {indapamide}, {get_chlorothiatzide}).empty
k_doac ≜ ({doac_test}, ∅, {doac_ok}).empty + ({doac_test}, ∅, {doac_fail}).empty + (∅, {doac_test}, ∅).k_doac
empty ≜ ∅.empty
kafib ≜ {afib}.empty + empty
khf ≜ {has_fib}.empty + empty
khr ≜ {heart_rate}.empty + empty
kcons ≜ {consensus_acei}.empty + empty
kage ≜ {over75}.empty + {below55}.empty + empty
kdiabete ≜ {diabete}.empty + empty
kdoacint ≜ {doac_int}.empty + empty
khyper ≜ {hyper}.empty + empty
korigin ≜ {origin}.empty + empty
```

Fig. 21 Context process definitions for the comorbidity case study in 5.1. The initial context is given by the parallel composition of therapies eafib1 | eafib2 | eafib3 | ghyper and the parallel composition of features kafib | khf | khr | kcons | kage | kdiabete | kdoacint | khyper | korigin | k_doac.

C.4 Auxiliary material for the T cell differentiation case study

The BioResolve specification derived from the Boolean network model (available at Puniya (2024), see Fig. 24) of the T cell differentiation case study from Puniya et al. (2018), and exploited in Brodo et al. (2025) is reported in Fig. 24. The corresponding experimentation with GROOVE has been discussed in Sect. 5.3.

C.5 Auxiliary material for the GROOVE experiments

To replicate the GROOVE experiments reported in Sects. 4 (for the toy running example) and 5, we have included the following supplementary resources with this submission:

- The rule systems described in Sect. 4;
- The start graphs derived from the BioResolve specifications in this appendix (C.1–C.4);
- Instructions for calling the GROOVE generator so as to reproduce all the exploration runs, occurrence graphs and model checking results (using GROOVE version 7.4.3).

```
myentities([]).

myreactions([
    react([akt],[],[akt]),
    react([erbb3],[],[akt]),
    react([mtor],[],[akt]),
    react([pdk1],[],[akt]),
    react([erbb1],[e,p],[erbb1]),
    react([egf],[e,p],[erbb1]),
    react([plcg],[e,p],[erbb1]),
    react([erbb2],[e,t,p],[erbb2]),
    react([egf],[e,t,p],[erbb2]),
    react([erbb3],[e,t,p],[erbb2]),
    react([erbb3],[e,p],[erbb3]),
    react([hrg],[e,p],[erbb3]),
    react([erk12],[],[erk12]),
    react([egf],[],[erk12]),
    react([p],[],[erk12]),
    react([mek12],[],[erk12]),
    react([mek12],[],[mek12]),
    react([erbb1],[],[mek12]),
    react([erbb2],[],[mek12]),
    react([erbb3],[],[mek12]),
    react([mtor],[],[mtor]),
    react([p],[],[mtor]),
    react([akt],[],[mtor]),
    react([p70s6k],[],[p70s6k]),
    react([akt],[],[p70s6k]),
    react([mtor],[],[p70s6k]),
    react([erk12],[],[p70s6k]),
    react([pdk1],[],[pdk1]),
    react([erbb1],[],[pdk1]),
    react([erbb2],[],[pdk1]),
    react([erbb3],[],[pdk1]),
    react([mek12],[],[pdk1]),
    react([pkca],[],[pkca]),
    react([plcg],[],[pkca]),
    react([plcg],[],[plcg]),
    react([egf],[],[plcg]),
    react([erbb1],[],[plcg]),
    react([erbb2],[],[plcg]),
    react([erbb3],[],[plcg]) ]).

myenvironment("[
    k = {egf,hrg}.k,
    ket = {e,t}.ket,
    korep = ({e}.korep + {p}.korep),
    korept = ({e}.korept + {p}.korept + {t}.korept),
    kge = (?{erbb1},{},{e}?.kge
          + ?{erbb2},{},{e}?.kge
          + ?{},{erbb1,erbb2},{}?.kge) ]").
```

Fig. 22 BioResolve implementation of the protein signaling network case study from 5.2.

```
 Jak1 = IFNgR and not SOCS1
 IL21 = STAT3 and NFAT
IL18R = IL18 and IL12 and not STAT6
SOCS1 = STAT1 or Tbet
  IL6 = RORgt
STAT5 = IL2R
 IL17 = (RORgt and not STAT1)
        or (STAT3 and IL17 and IL23R and not STAT1 and not STAT5)
STAT4 = IL12R and IL12 and not GATA3
IFNgR = (IFNg_e and NFAT) or (IFNg and NFAT)
STAT6 = IL4R and not IFNg and not SOCS1
GATA3 = (STAT6 and NFAT and not TGFb and not RORgt and not Foxp3
        and not Tbet) or (GATA3 and not Tbet)
        or (STAT5 and not TGFb and not RORgt and not Foxp3 and not Tbet)
  IL4 = GATA3 and NFAT and not STAT1
 NFkB = IRAK and not Foxp3
  IL2 = NFAT and NFkB and not Tbet
IL23R = (IL23 and STAT3 and not Tbet) or STAT3
 Tbet = (STAT4 and not RORgt and not Foxp3)
        or (STAT1 and not RORgt and not Foxp3)
        or (Tbet and not IL12 and not IFNg and not RORgt and not Foxp3)
TGFbR = TGFb and NFAT
RORgt = TGFbR and ((STAT3 and IL21R) or (STAT3 and IL6R)) and not Tbet
        and not GATA3 and not Foxp3
 IL6R = IL6 or IL6_e
IL21R = IL21
Foxp3 = (TGFbR and not (IL6R and STAT3) and not IL21R and not GATA3)
        or (STAT5 and not (IL6R and STAT3) and not IL21R and not GATA3)
 IRAK = IL18R
IL12R = (IL12 and NFAT) or (STAT4 and not GATA3) or Tbet or (TCR and not GATA3)
 IL2R = IL2 and NFAT
STAT3 = IL21R or IL23R or IL6R
 IFNg = NFkB or (STAT4 and NFkB and NFAT and not STAT3 and not STAT6)
        or (Tbet and not STAT3)
 NFAT = TCR and not Foxp3
STAT1 = (IL27 and NFAT) or Jak1
 IL4R = (IL4 and not SOCS1) or IL4_e
```

Fig. 23 Boolean updates of the T Cell differentiation model from Puniya et al. (2018), available at Puniya (2024).

Acknowledgements We thank Paolo Milazzo for the technical help in double checking the coherence of large state space generated by Python scripts based on BioResolve and those generated by GROOVE.

Author Contributions Both authors contributed equally to the paper, in writing and reviewing all sections.

Funding Research partially supported by the PRIN PNRR 2022 project *Resource Awareness in Programming* (RAP, P2022HXNSC), by the project *SEcurity and RIghts In the CyberSpace* (SERICS, PE00000014 - CUP H73C2200089001), under the National Recovery and Resilience Plan (NRRP) funded by the European Union - NextGenerationEU, by the INdAM-GNCS Projects *Reversibilità In SIstemi COncorrenti: analisi Quantitative e Funzionali* (RISICO, CUP E53C22001930001) and *Modelli e Analisi per sistemi reversibili e quantistici* (MARQ, CUP E53C24001950001), and by the University of Pisa PRA project *Formal methods for the healthcare domain based on spatial information* (FM4HD, PRA_2022_99).

Data Availability No datasets were generated or analysed during the current study.

Code Availability GROOVE is an open-source tool, available at https://github.com/nl-utwente-groove/code. For replication of the GROOVE experiments reported in this paper, we provide supplementary material described in Sect. C.5.

Declarations

Ethical approval Not applicable.

Conflict of interest The authors declare no Conflict of interest.

References

Ehrenfeucht A, Rozenberg G (2007) Reaction systems Fundam Informaticae 75(1–4):263–280

```
myreactions([
    react([stat5],[gata3,il21r,il6r],[foxp3]),
    react([stat5],[gata3,il21r,stat3],[foxp3]),
    react([tgfbr],[gata3,il21r,il6r],[foxp3]),
    react([tgfbr],[gata3,il21r,stat3],[foxp3]),
    react([gata3],[tbet],[gata3]),
    react([nfat,stat6],[foxp3,rorgt,tbet,tgfb],[gata3]),
    react([stat5],[foxp3,rorgt,tbet,tgfb],[gata3]),
    react([nfat,nfkb,stat4],[stat3,stat6],[ifng]),
    react([nfkb],[],[ifng]),
    react([tbet],[stat3],[ifng]),
    react([ifng,nfat],[],[ifngr]),
    react([ifnge,nfat],[],[ifngr]),
    react([il12,nfat],[],[il12r]),
    react([stat4],[gata3],[il12r]),
    react([tbet],[],[il12r]),
    react([tcr],[gata3],[il12r]),
    react([il17,il23r,stat3],[stat1,stat5],[il17]),
    react([rorgt],[stat1],[il17]),
    react([il12,il18],[stat6],[il18r]),
    react([nfat,nfkb],[tbet],[il2]),
    react([nfat,stat3],[],[il21]),
    react([il21],[],[il21r]),
    react([il23,stat3],[tbet],[il23r]),
    react([stat3],[],[il23r]),
    react([il2,nfat],[],[il2r]),
    react([gata3,nfat],[stat1],[il4]),
    react([il4],[socs1],[il4r]),
    react([il4e],[],[il4r]),
    react([rorgt],[],[il6]),
    react([il6],[],[il6r]),
    react([il6e],[],[il6r]),
    react([il18r],[],[irak]),
    react([ifngr],[socs1],[jak1]),
    react([tcr],[foxp3],[nfat]),
    react([irak],[foxp3],[nfkb]),
    react([il21r,stat3,tgfbr],[foxp3,gata3,tbet],[rorgt]),
    react([il6r,stat3,tgfbr],[foxp3,gata3,tbet],[rorgt]),
    react([stat1],[],[socs1]),
    react([tbet],[],[socs1]),
    react([il27,nfat],[],[stat1]),
    react([jak1],[],[stat1]),
    react([il21r],[],[stat3]),
    react([il23r],[],[stat3]),
    react([il6r],[],[stat3]),
    react([il12,il12r],[gata3],[stat4]),
    react([il2r],[],[stat5]),
    react([il4r],[ifng,socs1],[stat6]),
    react([stat1],[foxp3,rorgt],[tbet]),
    react([stat4],[foxp3,rorgt],[tbet]),
    react([tbet],[foxp3,ifng,il12,rorgt],[tbet]),
    react([nfat,tgfb],[],[tgfbr]) ]).
```

Fig. 24 BioResolve implementation of the T cell case study from 5.3.

Azimi S, Iancu B, Petre I (2014) Reaction system models for the heat shock response. Fund Inform 131(3–4):299–312. https://doi.org/10.3233/FI-2014-1016

Corolli L, Maj C, Marini F, Besozzi D, Mauri G (2012) An excursion in reaction systems: From computer science to biology. Theor Comput Sci 454:95–108. https://doi.org/10.1016/j.tcs.2012.04.003

Azimi S (2017) Steady states of constrained reaction systems. Theor Comput Sci 701((C)):20–26. https://doi.org/10.1016/j.tcs.2017.03.047

Okubo F, Yokomori T (2016) The computational capability of chemical reaction automata. Nat Comput 15(2):215–224. https://doi.org/10.1007/s11047-015-9504-7

Ehrenfeucht A, Main MG, Rozenberg G (2010) Combinatorics of life and death for reaction systems. Int J Found Comput Sci 21(3):345–356. https://doi.org/10.1142/S0129054110007295

Ehrenfeucht A, Main MG, Rozenberg G (2011) Functions defined by reaction systems. Int J Found Comput Sci 22(1):167–178. https://doi.org/10.1142/S0129054111007927

Bowles J, Brodo L, Bruni R, Falaschi M, Gori R, Milazzo P (2024) Enhancing reaction systems with guards for analysing comorbidity treatment strategies. In: Gori, R., Milazzo, P., Tribastone, M. (eds.) Computational Methods in Systems Biology - 22nd International Conference, CMSB 2024, Pisa, Italy, September 16-18, Proceedings. Lecture Notes in Computer Science, vol. 14971, pp. 27–44. Springer, ???. https://doi.org/10.1007/978-3-031-71671-3_3

Brodo L, Bruni R, Falaschi M, Gori R, Milazzo P (2025) Attractor and slicing analysis of a t cell differentiation model based on reaction systems. In: From Data to Models and Back - 11th International Symposium, DataMod 2023, Eindhoven, The Netherlands, November 6-7, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14618, pp. 69–89. Springer, ???. https://doi.org/10.1007/978-3-031-87217-4_4

Brodo L, Bruni R, Falaschi M (2021) A logical and graphical framework for reaction systems. Theor Comput Sci 875:1–27. https://doi.org/10.1016/J.TCS.2021.03.024

Ehrig H, Ehrig K, Prange U, Taentzer G (2006) Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. An EATCS Series. Springer, ???. https://doi.org/10.1007/3-540-31188-2

Heckel R, Taentzer G (2020) Graph Transformation for Software Engineers - With Applications to Model-Based Development and Domain-Specific Language Engineering. Springer, ??? . https://doi.org/10.1007/978-3-030-43916-3

Kreowski H, Rozenberg G (2019) Graph transformation through graph surfing in reaction systems. J. Log. Algebraic Methods Program. **109**. https://doi.org/10.1016/J.JLAMP.2019.100481

Ghamarian AH, Mol M, Rensink A, Zambon E, Zimakova M (2012) Modelling and analysis using GROOVE. Int J Softw Tools Technol Transf 14(1):15–40. https://doi.org/10.1007/S10009-011-0186-X

Ballis D, Brodo L, Falaschi M, Olarte C (2024) Process calculi and rewriting techniques for analyzing reaction systems. In: Gori, R., Milazzo, P., Tribastone, M. (eds.) Computational Methods in Systems Biology - 22nd International Conference, CMSB 2024, Pisa, Italy, September 16-18, 2024, Proceedings. Lecture Notes in Computer Science, vol. 14971, pp. 1–18. Springer, ???. https://doi.org/10.1007/978-3-031-71671-3_1

Plotkin GD (2004) A structural approach to operational semantics. J Log Algebraic Methods Program 60–61:17–139

Milner R (1980) A Calculus of Communicating Systems. LNCS, vol. 92. Springer, ???

Rensink A(2024) The GROOVE tool set, https://github.com/nl-utwente-groove/code/releases/tag/release-7_0_1version 7.0.1. Available at https://github.com/nl-utwente-groove/code

Brodo L, Bruni R, Falaschi M (2024) A framework for monitored dynamic slicing of reaction systems. Nat Comput 23(2):217–234. https://doi.org/10.1007/S11047-024-09976-3

Brodo L, Bruni R, Falaschi M, Gori R, Milazzo P, Montagna V, Pulieri P (2024) Causal analysis of positive reaction systems. Int J Softw Tools Technol Transf 26(4):509–526. https://doi.org/10.1007/S10009-024-00757-Y

Feder G, Eccles M, Grol R, Griffiths C, Grimshaw J (1999) Using clinical guidelines Bmj 318(7185):728–730. https://doi.org/10.1136/bmj.318.7185.728

Woolf SH, Grol R, Hutchinson A, Eccles M, Grimshaw J (1999) Potential benefits, limitations, and harms of clinical guidelines. BMJ 318(7182):527–530. https://doi.org/10.1136/bmj.318.7182.527

Hughes LD, McMurdo MET, Guthrie B (2013) Guidelines for people not for diseases: the challenges of applying UK clinical guidelines to people with multimorbidity. Age and Ageing 42, 62–69. https://doi.org/10.1093/ageing/afs100

Bowles JKF, Caminati MB (2017) A flexible approach for finding optimal paths with minimal conflicts. In: Proceedings of ICFEM 2017. LNCS, vol. 10610, pp. 209–225. Springer, ???. https://doi.org/10.1007/978-3-319-68690-5_13

Clavel M, Durán F, Eker S, Lincoln P, Martí-Oliet N, Meseguer J, Talcott CL (eds.) (2007) All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. Lecture Notes in Computer Science, vol. 4350. Springer, ???. https://doi.org/10.1007/978-3-540-71999-1

Heyde SV, Bender C, Henjes F (2014) Boolean erbb network reconstructions and perturbation simulations reveal individual drug response in different breast cancer cell lines. BMC Systems Biology 8, 75. https://doi.org/10.1186/1752-0509-8-75

Rubio R, Martí-Oliet N, Pita I, Verdejo A (2021) Strategies, model checking and branching-time properties in maude. J Log Algebraic Methods Program 123:100700. https://doi.org/10.1016/J.JLAMP.2021.100700

Brodo L, Bruni R, Falaschi M, Gori R, Milazzo P (2025) Slicing analyses for negative dependencies in reaction systems modeling gene regulatory networks. Natural Computing. Under review; will hopefully appear in same volume as this paper

Saez-Rodriguez J, Simeoni L, Lindquist JA, Hemenway R, Bommhardt U, Arndt B, Haus U-U, Weismantel R, Gilles ED, Klamt S (2007) A logical model provides insights into T cell receptor signaling. PLoS Comput Biol 3(8):163. https://doi.org/10.1371/journal.pcbi.0030163

Thakar J, Albert R (2010) Boolean models of within-host immune interactions. Curr Opin Microbiol 13(3):377–381. https://doi.org/10.1016/j.mib.2010.04.003

Puniya BL, Todd RG, Mohammed A, Brown DM, Barberis M, Helikar T (2018) A mechanistic computational model reveals that plasticity of CD4+ T cell differentiation is a function of cytokine composition and dosage. Frontiers in physiology 9, 878. https://doi.org/10.3389/fphys.2018.00878

Lafaille JJ (1998) The role of helper T cell subsets in autoimmune diseases. Cytokine & growth factor reviews 9(2):139–151. https://doi.org/10.1016/s1359-6101(98)00009-4

Hirahara K, Nakayama T (2016) CD4+ T-cell subsets in inflammatory diseases: beyond the T h 1/T h 2 paradigm. Int Immunol 28(4):163–171. https://doi.org/10.1093/intimm/dxw006

Meng X, Yang J, Dong M, Zhang K, Tu E, Gao Q, Chen W, Zhang C, Zhang Y (2016) Regulatory T cells in cardiovascular diseases. Nat Rev Cardiol 13(3):167–179. https://doi.org/10.1038/nrcardio.2015.169

Luckheeram RV, Zhou R, Verma AD, Xia B (2012) CD4+ T cells: differentiation and functions. Clinical and developmental immunology **2012**. https://doi.org/10.1155/2012/925135

Helikar T, Kowal B, McClenathan S, Bruckner M, Rowley T, Madrahimov A, Wicks B, Shrestha M, Limbu K, Rogers JA (2012) The cell collective: toward an open and collaborative approach to systems biology. BMC Syst Biol 6(1):1–14. https://doi.org/10.1186/1752-0509-6-96

Azimi S, Gratie C, Ivanov S, Petre I (2015) Dependency graphs and mass conservation in reaction systems. Theor Comput Sci 598:23–39. https://doi.org/10.1016/j.tcs.2015.02.014

Ivanov S, Rogojin V, Azimi S, Petre I (2018) WEBRSIM: A web-based reaction systems simulator. In: Díaz, C.G., Riscos-Núñez, A., Paun, G., Rozenberg, G., Salomaa, A. (eds.) Enjoying Natural Computing - Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday. Lecture Notes in Computer Science, vol. 11270, pp. 170–181. Springer, ???. https://doi.org/10.1007/978-3-030-00265-7_14

Nobile MS, Porreca AE, Spolaor S, Manzoni L, Cazzaniga P, Mauri G, Besozzi D (2017) Efficient simulation of reaction systems on graphics processing units. Fundam Informaticae 154(1–4):307–321. https://doi.org/10.3233/FI-2017-1568

Ferretti C, Leporati A, Manzoni L, Porreca AE (2020) The many roads to the simulation of reaction systems. Fundam Informaticae 171(1–4):175–188. https://doi.org/10.3233/FI-2020-1878

Brodo L, Bruni R, Falaschi M, Gori R, Levi F, Milazzo P (2023) Quantitative extensions of reaction systems based on SOS semantics. Neural Comput Appl 35(9):6335–6359. https://doi.org/10.1007/S00521-022-07935-6

Meski A, Penczek W, Rozenberg G (2015) Model checking temporal properties of reaction systems. Inf. Sci. 313, 22–42.https://doi.org/10.1016/J.INS.2015.03.048

Meski A, Koutny M, Mikulski L, Penczek W (2024) Reaction mining for reaction systems. Nat Comput 23(2):323–343. https://doi.org/10.1007/S11047-024-09989-Y

Puniya BL (2024) CD4+ T cell Differentiation model webpage on the CellCollective platform. Accessed: 18 March. https://research.cellcollective.org/?dashboard=true#module/6678:1/cd4-t-cell-differentiation/1