Slicing Analyses for Negative Dependencies in Reaction Systems modeling Gene Regulatory Networks

Linda Brodo[1*], Roberto Bruni[2*], Moreno Falaschi[3*], Roberta Gori[2*] and Paolo Milazzo[2*]

[1]Dipartimento di Scienze Economiche e Aziendali, University of Sassari, Via Muroni 25, Sassari, 07100, Italy.
[2]Department of Computer Science, University of Pisa, Largo B. Pontecorvo, 3, Pisa, 56127, Italy.
[3]Department of Information Engineering and Mathematics, University of Siena, Via Roma 56, Siena, 53100, Italy.

*Corresponding author(s). E-mail(s): brodo@uniss.it; roberto.bruni@unipi.it; moreno.falaschi@unisi.it; roberta.gori@unipi.it; paolo.milazzo@unipi.it;

Abstract

Reaction Systems (RSs) are a qualitative model inspired by biochemical processes, where the dynamics of complex systems is modelled by a collection of local reactions. Each reaction comprises a set of reactants that triggers a set of products unless hindered by the presence of some inhibitors. The use of inhibitors introduces non-monotonic behaviours that are difficult to analyze. This work focuses on the explainability of local phenomena, like the production of certain products or the reachability of certain attractors, by separating the causes responsible for reaching them from the irrelevant elements of a possibly much larger, global statespace. The main novelty of our approach is the ability to derive sufficient conditions that combine positive dependencies (e.g., requesting the presence of some entities at a certain stage, as already done in the literature) with negative ones (e.g., requesting the absence of some entities). This is achieved by combining and extending previous "static" constructions, like the transformation to Positive RSs and the minimization of RSs with "dynamic" techniques, like the process algebraic evolution of RSs, the slicing of computation and the on-the-fly generation of negative dependencies. We compare many different combinations of the above approaches, discussing their respective benefits and trade-offs in order to identify the most convenient analysis. We demonstrate our methodology on a case study involving T cell protein interactions, showing how it can reveal critical stimulus combinations and pinpoint potential drug targets by explaining phenotype emergence. Our analysis offers new insights and greater explanatory power than existing approaches.

Keywords: Systems Biology, Positive Reaction Systems, LTS, Slicing, Attractors

1 Introduction

Reaction Systems (RSs) [1, 2] are a successful computational framework inspired by biochemical systems. They offer a broad and flexible platform for constructing qualitative models, abstracting away the specific count of modeled entities. A RS comprises a finite set of entities S, called the *background set*, together with a finite set of reactions (R, I, P), with reactants R, inhibitors I, and products P drawn from S. The theory of RSs is based on three assumptions: *no permanency*, any entity vanishes unless it is sustained by a reaction; *no counting*, it abstracts away from the exact quantity of each entity; and *threshold nature of resources*, an entity is either available for all reactions, or it is not available at all.

A computation in a RS begins with an initial state (a finite set of entities) and, at each step, repeatedly generates new entities by applying all enabled reactions to the current state. Each reaction (R, I, P) contributes to the new state only if all reactants in R and no inhibitor in I is present in the current state. If these two conditions are met, the products P are included in the next state. The union of the products of all reactions determines the new state.

RSs are an interactive model of computation that can be fed with an ordered sequence of stimuli (sets of entities) called *context sequence*. Then, the i-th state in a computation will be determined by the result of the application of the reactions to the previous state, together with the set of entities in the i-th set of the context sequence.

RSs turned out to be particularly suitable to analyze Gene Regulatory Networks (GRNs) [3], namely networks describing how genes in a cell influence each other in order to regulate cell functions. GRNs are often expressed in terms of Boolean networks [3], which can be translated into RSs [4]. Such a translation allows analysis methods developed for RSs to be applied to investigate dynamical properties of genes. In this paper we will show that causality properties can be properly verified in the RS representation of the network. Discovering causality relationships between genes in a GRN is very important in order to better understand the related biological phenomena and to identify drug targets in case of diseases in which some of the genes do not behave as expected.

Objective. In this paper, we present a methodology for analyzing the attractors of a RS model, which represent the set of states towards which the system converges, and for identifying the biological entities responsible for their attainment.

Exploiting the operational semantics defined in [5], our method starts by computing a Labeled Transition System (LTS) that describes all the possible behaviors of the RS model. Then, the idea is to employ automated techniques, known as dynamic slicing methods (see [6]), to analyze all kinds of causal dependencies between the entities involved in the LTS and the attractors.

While verifying the entities that must be present for the achievement of an attractor can be done by simply applying slicing to the LTS (see, for example, [7]), this methodology does not allow us to detect which entities must necessarily be absent to reach that attractor. However, this kind of information is crucial since the distinctive feature of RSs is the inhibitory capability.

Methodology. In this paper, we extend the approach proposed in [7] with the ability to track both positive causal dependencies (entities that must necessarily be present) and negative causal dependencies (entities that must necessarily be absent).

To this aim, we initially follow a two steps approach. First, we apply to the RS under investigation an encoding into a Positive RS, as described in [8]. Roughly speaking, in Positive RSs, the absence of an entity is explicitly encoded with a new symbol, allowing us to treat it similarly to a standard entity. New reactions are added to produce these new "negative" entities (representing entity absence), and specific properties of the state need to be enforced so that the obtained state is meaningful. Of course, this transformation increases the set of reactions.

Once we obtain a Positive RS system, as second step we apply the slicing method to it. Running the slicing method on a Positive RS allows us to precisely track negative dependencies, represented by negative entities in the sliced trace. Moreover, this two-step methodology inspired us the definition of a new slicing algorithm, called Dynamic Negative Slicing, which is able to track negative dependencies directly from the original RS without encoding it into Positive RS form. By starting from an execution trace of the RS, in which only standard entities are present, the idea of the new algorithm is to infer negative dependencies by inspecting the reactions that, at each step, were not applied and caused the missing production (i.e., the absence) of an entity of interest.

The new method is in principle more efficient for the analysis of a single trace, since it avoids the encoding into Positive RS which could require generating a very large number of new reactions. However, we will show that it can be slightly less precise than the approach based on the Positive RS representation. Indeed, the encoding into Positive RS allows the set of reactions expressing the absence of an entity to be simplified by applying logical minimization, and this prevents identifying non relevant dependencies that are actually due to redundancies in the reactions. The Dynamic Negative Slicing, for reasons of efficiency, do not simplify reactions and therefore it might compute a superset of the significant dependencies.

Contribution. This paper is an extended version of [7]. The novelties w.r.t. [7] are that here we consider Positive RSs, and hence we can also analyse the negative dependencies for the studied entities. We also define a new Dynamic Negative Slicing algorithm for computing the negative dependencies, and show that

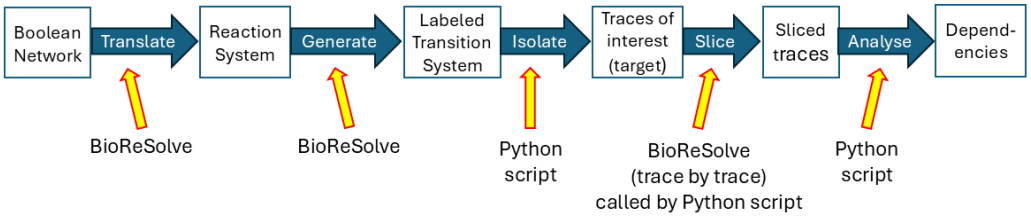


Fig. 1: Software architecture of the analysis pipeline.

it is more efficient than computing first a Positive version of a RS and then applying the standard slicing. On the other hand, we show that for the Positive version some static techniques of minimization can be applied to the computed transformation, improving the precision of the slicing algorithm. Another contribution is a formal comparison of the two slicing techniques, showing the correlation between them. In particular, we compare and relate all possible instances of the two algorithms when applied to standard RS, or to the positive version of a RS, with or without a minimization of the rules of the RS. The results of the comparison are summarized in Fig. 3.

To demonstrate the effectiveness of our methodology, we have applied it to the case study considered in [7], where we have modelled T cell differentiation in the immune system. This model encompasses reactions enabling T cells to manifest various phenotypes in response to environmental stimuli. To this aim we extended the implementation of analysis tool BioReSolve [9] that now includes the two slicing techniques. We also included some resuming tables to compare the precision and efficiency of the computed results with the two different techniques.

Several tools find fruitful application in our study. The first is BioReSolve [9], a freely available tool, first introduced in [5], which implements RSs in SWI-Prolog. In order to develop the analysis of attractors, we have designed and implemented a new Python script leveraging the `networkx` package. Our script interacts with BioReSolve through the Python-to-Prolog binding facilitated by the `swiplserver` Python package.

BioReSolve. BioReSolve is a rapid prototyping tool for the simulation, analysis, and verification of reaction system models. It is intended for researchers in computational systems biology and formal methods. BioReSolve implements fully non-deterministic simulation of a reaction systems model, as well as the whole state space generation. Once BioResolve is loaded on SWI-Prolog together with the system model specification, the user can easily query the tool by invoking the directive `main_do(...)` with different options, e.g., for simulation, sanity checks, reachability analysis, trace generation, causal analysis, LTS generation, model transformation, etc. Each option is represented by a Prolog atom and linked to a specific kind of analysis. For example, the directive `main_do(digraph)` generates the entire state-space as a labelled transition system, where the appearance of nodes, arcs and labels is fully customizable by the user. The output is in dot format that can be visualized with Graphviz or Cytoscape tool. Many other novel options have been developed to run the experiments in this paper and will be referred in the main text. Among them, we mention the option `bn2rs` to translate Boolean networks specification into RS specifications, the options `minimize`, `sppos`, and `minsppos` that apply different transformations to the RS specification itself, and the options `ordslice` and `negslice` that support the slicing algorithms we present. BioReSolve has been designed in close connection with the process algebraic version of reaction systems, it enjoys useful modularity and compositionality features of the process algebraic approach that made it possible to introduce novel features, such as the support for guarded contexts or quantitative analysis.

Python scripting and analysis pipeline. A schematic representation of the software architecture of our analysis pipeline is depicted in Figure 1. Roughly, our methodology consists of the following steps: firstly, we utilise BioReSolve to run our RS model, obtained through automatic translation from the Boolean network to the corresponding RS. Subsequently, we add the contexts which are necessary to start a computation in BioReSolve by considering a non-deterministic context which provides all combinations of entities/stimuli as different initial alternatives. The computation proceeds along each path by repeating the same initial context of that path until reaching a loop in the computation graph. BioReSolve returns

a representation of the LTS as a graph in dot format, which can be loaded by our Python script. Our Python script conducts an analysis of the attractors, considering different *targets* corresponding to the different combinations of phenotypes that a T cell can exhibit. The slicing tool, which performs a form of causality analysis, is used to simplify the computation traces, retaining only the information pertinent to producing a *target* of our analysis. This enables us to uncover significant facts, such as identifying the genes necessary for the expression of a target transcription factor. As a difference w.r.t. the previous work [7], here we consider also the translation of our RS models into a Positive RS, and the minimized versions of both the RS and the Positive RS models.

Our methodology is general, and hence it could be applied to many other case studies, such as the ones in the public data base *Cell Collective* [10], which contains our current case study. Moreover, our methodology introduces an original analysis of attractors combined with slicing algorithms that provide a deeper understanding of the evolution of the system and the entities involved. This can have a significant impact in drug design, since understanding which genes are necessary or, more generally, contribute to the expression of some other genes or transcription factors involved in a disease can be helpful for the identification of potential drugs or vaccine targets.

Related work. This work extends the approach started in [7] for the computational modelling and analysis of gene regulatory networks [3, 11]. In particular, the attractors and slicing analyses, combined together, constitute a new methodology for the investigation of causality properties in such networks. Causality properties can be studied by running simulations [12, 13], to investigate the role of initial configurations of active genes and identify attractors. Moreover, formal causality approaches have been investigated for biological applications in [14, 15], and in particular for RSs in [16, 17]. Compared to these approaches, ours provides information about the role of both the environmental stimuli (or the "input" nodes of the networks) and the intermediate genes involved in the achievement of a given target, computed in an integrated way and with scalable performances.

Structure of the paper. In Section 2, we introduce the basic framework of RSs and the slicing algorithm, then we revise the SOS semantics of Reaction Systems processes. In Section 3, we introduce the Positive RSs and the novel notion of minimization of RSs. In Section 4, we report our previous algorithm of slicing analysis of RSs. In Section 5, we present its novel extension to detect negative dependencies, and compare it to the previous algorithms. In Section 6, we define the RS model of T cell differentiation and present the implementation of our methodology. The analysis of the model is described in Section 7. We draw some concluding remarks and sketch future work in Section 8.

2 Reaction Systems

The formalism of Reaction Systems (RSs) [1, 18] originated in the field of Natural Computing as a qualitative computational model for the study of biochemical reactions in living cells. Since their introduction, RSs have been successfully applied to the analysis of complex systems in many different fields [19–24].

The main concepts of RSs are briefly recalled here, as introduced in the classical set-theoretic version. We will use the term *entities* to identify generic molecular substances (e.g., atoms, ions, molecules), or even any other components, involved in the dynamics of a (biochemical) complex system.

Let S be a (finite) set of entities. A reaction in S is a triple $r = (R, I, P)$, where $R, I, P \subseteq S$ are finite sets and $R \cap I = \varnothing$. The sets R, I, P are the sets of *reactants*, *inhibitors*, and *products*, respectively. For a reaction to take place in the current state: all the reactants must be present and all the inhibitors must be absent. Products are the outcome of the execution of a reaction, to be released in the next state. We denote with $rac(S)$ the set of all reactions over S.

Remark 1. *Without loss of generality, the sets of reactants, inhibitors, and products are typically assumed to be non-empty, since dummy entities—always present or always absent—can be introduced if necessary. However, in this paper, for the sake of readability, we also allow reactions with empty sets of reactants and/or inhibitors.*

Given a subset of entities $W \subseteq S$ and a reaction $r = (R, I, P) \in rac(S)$, the enabling predicate of r in W, denoted by $en_r(W)$ and the result of r on W, denoted by $res_r(W)$, are defined as follows:

$$en_r(W) \triangleq R \subseteq W \ \wedge \ I \cap W = \varnothing$$

$$res_r(W) \triangleq \begin{cases} P & \text{if } en_r(W) \\ \varnothing & \text{otherwise} \end{cases}$$

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

$r_1 = (\{\mathsf{tgfbr}, \mathsf{stat3}, \mathsf{il6r}\}, \{\mathsf{tbet}, \mathsf{gata3}, \mathsf{foxp3}\}, \{\mathsf{rorgt}\})$
$r_2 = (\{\mathsf{tgfbr}, \mathsf{stat3}, \mathsf{il21r}\}, \{\mathsf{tbet}, \mathsf{gata3}, \mathsf{foxp3}\}, \{\mathsf{rorgt}\})$
$r_3 = (\{\mathsf{il23r}\}, \varnothing, \{\mathsf{stat3}\})$
$r_4 = (\{\mathsf{il21}\}, \varnothing, \{\mathsf{il21r}\})$
$r_5 = (\{\mathsf{il6}\}, \varnothing, \{\mathsf{il6r}\})$
$r_6 = (\{\mathsf{tcr}\}, \{\mathsf{foxp3}\}, \{\mathsf{nfat}\})$
$r_7 = (\{\mathsf{il27}, \mathsf{nfat}\}, \varnothing, \{\mathsf{stat1}\})$
$r_8 = (\{\mathsf{stat1}\}, \{\mathsf{rorgt}, \mathsf{foxp3}\}, \{\mathsf{tbet}\})$

Fig. 2: The reactions of our running example $\mathcal{A}_{\text{toy}}$.

the reactions A on W, denoted $res_A(W)$, is just the additive lifting of res_r, i.e., $res_A(W) \triangleq \cup_{r\in A} res_r(W)$. Note that the result of a computation step does not depend on the order of application of the reactions.

Since living cells are seen as open systems that react to environmental stimuli, the behavior of a RS is formalized in terms of interactions with sets of entities possibly provided by the context. Let $\mathcal{A} = (S, A)$ be a RS and let $n \geq 0$. An n-steps *interactive process* in $\mathcal{A}$ is a pair $\pi = (\gamma, \delta)$ of sequences s.t. $\gamma = \{C_i\}_{i\in[0,n]}$ is the *context sequence* and $\delta = \{D_i\}_{i\in[0,n]}$ is the *result sequence*, where $C_i, D_i \subseteq S$ for any $i \in [0, n]$, and $D_{i+1} = res_A(D_i \cup C_i)$ for any $i \in [0, n-1]$. We call $\tau = \{W_i\}_{i\in[0,n]}$ the *state sequence*, with $W_i \triangleq C_i \cup D_i$ for any $i \in [0, n]$.

Remark 2. *Although the entities provided by the context can serve as reactants and/or inhibitors in reactions, they often do not overlap with any of the reaction products. Following a disciplined approach, we shall therefore make the distinction explicit, by assuming that the set of entities is partitioned in two subsets $S = S_D \uplus S_C$ such that for any context sequence $\gamma = \{C_i\}_{i\in[0,n]}$ it holds $C_i \subseteq S_C$, and for any reaction (R, I, P) it holds $P \subseteq S_D$.*

Example 1 (The reaction system $\mathcal{A}_{\text{toy}}$)**.** *As a running toy example, we consider a fragment of our main case study on the T-Cell differentiation system: here we exploit smaller sets of entities and reactions. We let $\mathcal{A}_{\text{toy}} \triangleq (S, A)$ and $S \triangleq S_C \uplus S_D$, where $S_C \triangleq \{\mathsf{tcr}, \mathsf{il27}\}$, and S_D consists of all the remaining entities that appear in the set of eight reactions $A \triangleq \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$ listed in Fig. 2. Then, we consider a 3-steps interactive process depicted in Table 1, where we listed the context and result sequence in the even rows while the odd rows list the reactions applied to obtain the next state.*

τ	γ	δ
W_0	$C_0 = \{\mathsf{tcr}\}$	$D_0 = \{\mathsf{il21}, \mathsf{il23r}\}$
enables	r_3, r_4, r_6	
W_1	$C_1 = \{\mathsf{il27}\}$	$D_1 = \{\mathsf{il21r}, \mathsf{nfat}, \mathsf{stat3}\}$
enables	r_7	
W_2	$C_2 = \varnothing$	$D_2 = \{\mathsf{stat1}\}$
enables	r_8	
W_3	$C_3 = \varnothing$	$D_3 = \{\mathsf{tbet}\}$

Table 1: Execution of the fragment $\mathcal{A}_{\text{toy}}$ of the T-Cell differentiation system. We take the context sequence $\gamma = \{C_i\}_{i\in[0,3]}$, where $C_0 = \{\mathsf{tcr}\}$, $C_1 = \{\mathsf{il27}\}$, $C_2 = C_3 = \varnothing$, and the result sequence $\delta = \{D_i\}_{i\in[0,3]}$, where $D_0 = \{\mathsf{il21}, \mathsf{il23r}\}$.

2.1 SOS Rules for Reaction Systems

In this section we briefly recall the algebraic syntax and the formal semantics for RSs based on SOS inference rules, as introduced in [5]; however we limit ourselves to a mostly intuitive description, as for the purposes of our work it is not necessary to go through the details of the SOS rules and the induced semantics. For a complete description we refer to [5].

Definition 1 (RS processes)**.** *Let $S \triangleq S_C \uplus S_D$ be a finite set of entities. A* RS process P *is any term defined by the following grammar:*

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

where $R, I \subseteq S$, $C \subseteq S_C$, $P, D \subseteq S_D$, $R, I, P \neq \varnothing$, and X belongs to the set of process variables.

A RS process P encloses a *mixture* process M composed by the parallel composition of some reactions (R, I, P), some sets of entities D (possibly empty), and some *context* K. We write $\prod_{j\in J} \mathsf{M}_j$ for the parallel composition of all M_j with $j \in J$. A context process K is a possibly non-deterministic and recursive system: the nil context $\mathbf{0}$ stops the computation; the prefixed context $C.\mathsf{K}$ provides the entities C to be available to the reactions, and the continuation K will be the context at the next step; the non-deterministic choice $\mathsf{K}_1 + \mathsf{K}_2$ allows the context to behave as either K_1 or K_2; X is a process variable, and $\mathsf{rec}\ X.\ \mathsf{K}$ is the usual recursive operator of process algebras.

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 commutative monoids (unit, associativity and commutativity) for parallel composition $\cdot|\cdot$, with $\varnothing$ as the unit, and the laws of idempotent and commutative 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 $D_1, D_2 \subseteq S_D$. Under these assumptions, any RS process M can always be written in a standard form $(\prod_{j\in J}(R_j, I_j, P_j)) \mid D \mid (\prod_{k\in K} \mathsf{K}_k)$ for some suitable sets of reactions $\{(R_j, I_j, P_j) \mid j \in J\}$ and contexts $\{\mathsf{K}_k \mid k \in K\}$.

Definition 2 (RSs as RS processes)**.** *Let $\mathcal{A} = (S, A)$ be a RS, and $\pi = (\gamma, \delta)$ an n-step interactive process in $\mathcal{A}$, with $\gamma = \{C_i\}_{i\in[0,n]}$ and $\delta = \{D_i\}_{i\in[0,n]}$. For any step $i \in [0, n]$, the corresponding RS process $[\![\mathcal{A}, \pi]\!]_i$ is defined as follows:*

$$[\![\mathcal{A}, \pi]\!]_i \triangleq \left[\prod_{r\in A} r \mid D_i \mid \mathsf{K}_{\gamma^i}\right]$$

where $\mathsf{K}_{\gamma^i} \triangleq C_i.C_{i+1}.\cdots.C_n.\mathbf{0}$ is the serialization of the entities offered by γ^i (the shifting of γ at the i-th step). We write $[\![\mathcal{A}, \pi]\!]$ as a shorthand for $[\![\mathcal{A}, \pi]\!]_0$.

The operational semantics of RS processes is compositionally defined by SOS inference rules defining the behaviour of each syntax operator. This semantics induces a labelled transition system, whose nodes are processes and whose transition labels ℓ carry some information about the conditions under which the transition from the source state to the target states is possible, like the available entities $D \subseteq S_D$, the entities $C \subseteq S_C$ provided by the context, the reactants whose presence enabled some reactions, the reactants whose absence disabled other reactions and much more. Here, for simplicity, we abstract away from most details that will not be exploited by our slicing algorithms and just mention that whenever $\mathsf{P} \xrightarrow{\ell} \mathsf{P}'$, it is possible to extract from the label ℓ the set of entities available in P, written D_ℓ, the set of entities provided by the context, written C_ℓ and the set of enabled reactions, written En_ℓ. Moreover, it is known from [5, Theorem 19] that the SOS semantics of an RS process matches the set-theoretic dynamics of its underlying Reaction System. More precisely, for any Reaction System $\mathcal{A} = (S, A)$ and any interactive process $\pi = (\gamma, \delta)$ in $\mathcal{A}$, with $\gamma = \{C_i\}_{i\in[0,n]}$ and $\delta = \{D_i\}_{i\in[0,n]}$, any outgoing transition from $[\![\mathcal{A}, \pi]\!]_i$ leads to $[\![\mathcal{A}, \pi]\!]_{i+1}$ and its label ℓ is such that $D_\ell = D_i$, $C_\ell = C_i$ and $En_\ell = \{r \in A \mid en_r(D_\ell \cup C_\ell)\}$.

The following example illustrates how the LTS semantics works.

Example 2. *The process $\mathsf{P}_0 = [\![\mathcal{A}_{\text{toy}}, \pi]\!]$ from Example 1 is defined as $\mathsf{P}_0 \triangleq [\mathsf{Reacts} \mid \{\mathsf{il23r}, \mathsf{il21}\} \mid \mathsf{K}_\gamma]$ where $\mathsf{Reacts} = r_1 \mid r_2 \mid r_3 \mid r_4 \mid r_5 \mid r_6 \mid r_7 \mid r_8$ and $\mathsf{K}_\gamma = \{\mathsf{tcr}\}.\{\mathsf{il27}\}.\varnothing.\mathbf{0}$. Then, $\mathsf{P}_0 \xrightarrow{\ell_1} \mathsf{P}_1 \xrightarrow{\ell_2} \mathsf{P}_2 \xrightarrow{\ell_3} \mathsf{P}_3$ where $D_{\ell_1} = \{\mathsf{il23r}, \mathsf{il21}\}$, $C_{\ell_1} = \{\mathsf{tcr}\}$, $En_{\ell_1} = \{r_3, r_4, r_6\}$, $\mathsf{P}_1 \equiv [\mathsf{Reacts} \mid \{\mathsf{nfat}, \mathsf{stat3}, \mathsf{il21r}\} \mid \{\mathsf{il27}\}.\varnothing.\mathbf{0}]$, $D_{\ell_2} = \{\mathsf{nfat}, \mathsf{stat3}, \mathsf{il21r}\}$, $C_{\ell_2} = \{\mathsf{il27}\}$, $En_{\ell_2} = \{r_7\}$, $\mathsf{P}_2 \equiv [\mathsf{Reacts} \mid \{\mathsf{stat1}\} \mid \varnothing.\mathbf{0}]$, $D_{\ell_3} = \{\mathsf{stat1}\}$, $C_{\ell_3} = \varnothing$, $En_{\ell_3} = \{r_8\}$, and $\mathsf{P}_3 \equiv [\mathsf{Reacts} \mid \{\mathsf{tbet}\} \mid \mathbf{0}]$.*

2.2 History Enriched Processes

For defining the slicing algorithms, we find it convenient to introduce a slightly different notation that records in the current state the whole past state sequence that led there. Correspondingly, we enrich the syntax of state processes by a novel prefixing production HP that juxtaposes the history H before the process P.

Definition 3 (History enriched RS processes)**.** *A* History enriched RS process P *is any term defined by the following grammar:*

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

where the history H has the form $\frac{D}{C} \xrightarrow{N}$ with $D \subseteq S_D$, $C \subseteq S_C$ and $N \subseteq rac(S)$, such that $en_r(D \cup C)$ for any $r \in N$.

Roughly, the history H records the set of result entities D, context entities C and reactions N that were exploited in the previous step to reach the current process P. It is immediate from the syntax that histories can be stacked, so that a generic process has the form $\mathsf{H}_0\mathsf{H}_1\cdots\mathsf{H}_m\mathsf{P}$, where H_m records the history of the last step and H_0 that of the first step. The SOS semantics is then extended in such a way that:

- if $\mathsf{M} \xrightarrow{\ell} \mathsf{M}'$, then $[\mathsf{M}] \xrightarrow{\ell} (\frac{D_\ell}{C_\ell} \xrightarrow{En_\ell} [\mathsf{M}'])$;
- if $\mathsf{P} \xrightarrow{\ell} \mathsf{P}'$, then $\mathsf{HP} \xrightarrow{\ell} \mathsf{H}'\mathsf{P}'$.

It follows that for any Reaction System $\mathcal{A} = (S, A)$ and any interactive process $\pi = (\gamma, \delta)$ in $\mathcal{A}$, with $\gamma = \{C_i\}_{i\in[0,n]}$ and $\delta = \{D_i\}_{i\in[0,n]}$, any outgoing trace of length $k \in [0, n]$ from $[\![\mathcal{A}, \pi]\!]_i$ leads to $\frac{D_0}{C_0} \xrightarrow{N_1} \cdots \frac{D_{k-1}}{C_{k-1}} \xrightarrow{N_k} [\![\mathcal{A}, \pi]\!]_{i+k}$ such that N_i is the set of reactions in A that are enabled by $D_{i-1} \cup C_{i-1}$, i.e., $N_i = \{r \in A \mid en_r(D_{i-1} \cup C_{i-1})\}$ for all $i \in [1, k]$.

In the following, we assume that reactions are indexed by consecutive positive numbers, as it was the case in Example 1 and abuse the notation by letting the index j denote the j-th reaction r_j. Analogously, we use sets of indexes N_i within histories, and write $r_j \in N_i$ whenever $j \in N_i$.

Example 3. *Consider the process execution* $\mathsf{P}_0 \xrightarrow{\ell_1} \mathsf{P}_1 \xrightarrow{\ell_2} \mathsf{P}_2 \xrightarrow{\ell_3} \mathsf{P}_3$ *detailed in Example 2. In the case of history enriched processes, it becomes written*

$$\mathsf{P}_0 \xrightarrow{\ell_1} \mathsf{H}_0\mathsf{P}_1 \xrightarrow{\ell_2} \mathsf{H}_0\mathsf{H}_1\mathsf{P}_2 \xrightarrow{\ell_3} \mathsf{H}_0\mathsf{H}_1\mathsf{H}_2\mathsf{P}_3$$

where

- $\mathsf{H}_0 = \frac{\{\mathsf{il23r},\mathsf{il21}\}}{\{\mathsf{tcr}\}} \xrightarrow{\{3,4,6\}}$
- $\mathsf{H}_1 = \frac{\{\mathsf{il21r},\mathsf{stat3},\mathsf{nfat}\}}{\{\mathsf{il27}\}} \xrightarrow{\{7\}}$
- $\mathsf{H}_2 = \frac{\{\mathsf{stat1}\}}{\varnothing} \xrightarrow{\{8\}}$

Afterwards, whenever the set of enabled reactions is not relevant, we will just omit the transition labels, writing, e.g., $\mathsf{H}_0\mathsf{H}_1\mathsf{H}_2\mathsf{P}_3$ as

$$\frac{\{\mathsf{il23r},\mathsf{il21}\}}{\{\mathsf{tcr}\}} \rightarrow \frac{\{\mathsf{il21r},\mathsf{stat3},\mathsf{nfat}\}}{\{\mathsf{il27}\}} \rightarrow \frac{\{\mathsf{stat1}\}}{\varnothing} \rightarrow \mathsf{P}_3.$$

3 Positive Reaction Systems

In this paper we focus on a particular kind of Reaction Systems, namely the ones that are formed by reactions without inhibitors, previously studied in [8, 25] for closed RSs (i.e., for RSs not interacting with the context). Such reactions are called *positive* and can simply be written as pairs (R, P): they correspond to ordinary reactions $(R, \varnothing, P)$. We can always encode any standard RS $\mathcal{A} = (S, A)$ into an equivalent one without inhibitors. In this section we illustrate the main ideas of this encoding while referring the interested reader to [8, 25] for more details. The main idea is to track the absence of entities: for each entity $\mathsf{a} \in S$ we need to introduce a new (negative) entity that explicitly tracks the absence of an entity in a state.[1] Therefore, for any $\mathsf{a} \in S$ we introduce the new entity $\bar{\mathsf{a}}$ whose presence in a state indicates that a is absent. Note that, following this idea, in any meaningful state $W = D \cup C$ there is always either one between a and $\bar{\mathsf{a}}$, but never both. As a consequence, for any entity $\mathsf{a} \in S_C$, we must assume that the context will provide either a or $\bar{\mathsf{a}}$.

In the following, we will refer to the elements of the set S as *positive* entities and to the elements of the set $\bar{S} \triangleq \{\bar{\mathsf{a}} \mid \mathsf{a} \in S\}$ as *negative* entities. We leave implicit that S and $\bar{S}$ are disjoint sets and we denote by $\mathbf{S} \triangleq S \uplus \bar{S}$ the set of positive and negative entities. In the following we use boldface symbols to denote the elements and the subsets of $\mathbf{S}$. Whenever needed, we use the subscripts D and C to discriminate between entities related to reaction products (like in S_D, $\bar{S}_D$ and $\mathbf{S}_D$) and those provided by the context (like in (like in S_C, $\bar{S}_C$ and $\mathbf{S}_C$).

Remark 3. *Please note that throughout this paper we will use over-lining solely for symbol decoration and never for set complementation.*

We need now to formally state that in a state we can never have both a and $\bar{\mathsf{a}}$ since an entity cannot be present and absent at the same time.

Definition 4 (State consistency)**.** *A set* $\mathbf{W} \subseteq \mathbf{S}$ *is* non-contradictory *if for all entities* $\mathsf{a} \in S$ *it holds* $\{\mathsf{a}, \bar{\mathsf{a}}\} \not\subseteq \mathbf{W}$*. A non-contradictory state* $\mathbf{W} \subseteq \mathbf{S}$ *is* consistent *if, for any entity* $\mathsf{a} \in S$*, either* $\mathsf{a} \in \mathbf{W}$ *or* $\bar{\mathsf{a}} \in \mathbf{W}$ *holds.*

We are now ready to define Positive RSs.

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

1. *Each reaction* r *in* A *is positive, i.e.,* $r = (\mathbf{R}, \varnothing, \mathbf{P})$ *for some non-contradictory sets* $\mathbf{R}$ *and* $\mathbf{P}$*;*
2. *Consistency preservation: for any consistent state* $\mathbf{W}$*, the result set* $res_{\mathcal{A}^+}(\mathbf{W})$ *must be consistent.*

We assume that the initial state $\mathbf{D}_0 \subseteq \mathbf{S}_D$ is non-contradictory set such that $\forall a \in \mathbf{S}_D$, either $\mathsf{a} \in \mathbf{D}_0$ or $\bar{\mathsf{a}} \in \mathbf{D}_0$ and that the sets $\mathbf{C}_0, ..\mathbf{C}_n \subseteq \mathbf{S}_C$ provided by the context are non-contradictory sets such that $\forall a \in \mathbf{S}_C$, either $\mathsf{a} \in \mathbf{C}_i$ or $\bar{\mathsf{a}} \in \mathbf{C}_i$, for $i \in \{0, .., n\}$. Under these hypothesis, the second condition of Definition 5 guarantees that all result states traversed by the computation will be consistent as well.

3.1 From RSs to Positive RSs

Next we show that for each standard RS $\mathcal{A} = (S, A)$ it is possible to construct a Positive RS $\mathcal{A}^+$ that can exactly mimic the behavior of $\mathcal{A}$. The Positive RS $\mathcal{A}^+$ takes the form $(\mathbf{S}, A^+)$ whose reactions in A^+ can be split in two categories: A^+_{pos} that simply embeds the original reactions A and A^+_{neg} whose reactions serve for negative entities bookkeeping. More in details, for each reaction $(R, I, P) \in A$, there will be one positive reaction $(R \cup \bar{I}, P) \in A^+_{pos}$. Moreover, now that a negative entity $\bar{\mathsf{a}}$ tracking the absence of a has been introduced, we need to add reactions in A^+_{neg} guaranting that the negative entity $\bar{\mathsf{a}}$ is produced whenever no reaction in A that produces a is enabled. For this purpose, assume we collect all reactions in A that are capable of producing a: to ensure that none of them is enabled, we must make sure that, for each one, at

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

least one reactant is absent or at least one inhibitor is present. We formalize this intuition by introducing the *prohibiting* set for each entity a.

Definition 6 (Prohibiting set)**.** *Let $\mathcal{A} = (S, A)$ be a standard RS and $\mathsf{a} \in S_D$ one of its entities. A non-contradictory set $\mathbf{T} \subseteq \mathbf{S}$ is a* prohibiting set *for a if it satisfies the following condition:*

$$\mathbf{T} \in \bigcup\!\!\!\bigcup_{\{(R,I,P\cup\{a\})\in A\}} \{\{\mathbf{b}\} \mid \mathbf{b} \in \overline{R} \cup I\}$$

where, the component-wise union $\bigcup\!\!\!\bigcup$ of sets of sets is defined by letting

$$\{X_i\}_{i\in I} \Cup \{Y_j\}_{j\in J} \triangleq \{X_i \cup Y_j \mid i \in I, j \in J\}.$$

We denote by $Proh_{\mathcal{A}}(\mathsf{a})$ the set of all and only prohibiting sets for a.

Intuitively, a prohibiting set for a is constructed by selecting, for each reaction that produces a, either one missing reactant or one available inhibitor. Notably, the same entity may be selected multiple times to disable different reactions. Similarly, a single reaction may be inhibited by multiple elements, especially when these elements are shared across several reactions. This is illustrated by the following example.

Example 4. *Consider the two reactions of $\mathcal{A}_{\text{toy}}$ reported in Example 1 that produce the entity* rorgt*:*

$r_1 = (\{\mathsf{tgfbr}, \mathsf{stat3}, \mathsf{il6r}\}, \{\mathsf{tbet}, \mathsf{gata3}, \mathsf{foxp3}\}, \{\mathsf{rorgt}\})$
$r_2 = (\{\mathsf{tgfbr}, \mathsf{stat3}, \mathsf{il21r}\}, \{\mathsf{tbet}, \mathsf{gata3}, \mathsf{foxp3}\}, \{\mathsf{rorgt}\})$

Then the prohibiting set for rorgt *is the following*
$Proh_{\mathcal{A}}(\mathsf{rorgt}) = \{\{\overline{\mathsf{tgfbr}}\}, \{\overline{\mathsf{tgfbr}}, \overline{\mathsf{stat3}}\}, \{\overline{\mathsf{tgfbr}}, \overline{\mathsf{il21r}}\},$
$\{\overline{\mathsf{tgfbr}}, \mathsf{tbet}\}, \{\overline{\mathsf{tgfbr}}, \mathsf{gata3}\},$
$\{\overline{\mathsf{tgfbr}}, \mathsf{foxp3}\}, \{\overline{\mathsf{stat3}}\}, \{\overline{\mathsf{stat3}}, \overline{\mathsf{il21r}}\}$
$\{\overline{\mathsf{stat3}}, \mathsf{tbet}\}, \{\overline{\mathsf{stat3}}, \mathsf{gata3}\},$
$\{\overline{\mathsf{stat3}}, \mathsf{foxp3}\}, \{\overline{\mathsf{il6r}}, \overline{\mathsf{tgfbr}}\},$
$\{\overline{\mathsf{il6r}}, \overline{\mathsf{stat3}}\}, \{\overline{\mathsf{il6r}}, \overline{\mathsf{il21r}}\}\{\overline{\mathsf{il6r}}, \mathsf{tbet}\},$
$\{\overline{\mathsf{il6r}}, \mathsf{gata3}\}\{\overline{\mathsf{il6r}}, \mathsf{foxp3}\}, \{\mathsf{tbet}\}$
$\{\mathsf{tbet}, \overline{\mathsf{il21r}}\}, \{\mathsf{tbet}, \mathsf{gata3}\},$
$\{\mathsf{tbet}, \mathsf{foxp3}\}, \{\mathsf{gata3}\}, \{\mathsf{gata3}, \overline{\mathsf{il21r}}\}$
$\{\mathsf{gata3}, \mathsf{foxp3}\}, \{\mathsf{foxp3}\}$
$\{\mathsf{foxp3}, \overline{\mathsf{il21r}}\}\}$

Clearly, any prohibiting set for a forms a valid set of reactants for the production of $\bar{\mathsf{a}}$.

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

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

It can be proved that the resulting $\mathcal{A}^+$ is a Positive RS, i.e., that it satisfies the two conditions from Definition 5.

Example 5 (The positive RS $\mathcal{A}^+_{toy}$)**.** *Consider again the reaction system $\mathcal{A}_{\text{toy}}$ as we defined in Example 1. The complete Positive RS $\mathcal{A}^+_{toy}$ that we obtain is reported in Fig. A1 of the Appendix. Here, we just focus on the two reactions of $\mathcal{A}_{\text{toy}}$ reported in Example 4 that produce the entity* rorgt*. Correspondingly, for the production of the* rorgt *we will have two positive reactions in $\mathcal{A}^+_{toy}$ that are*

$(\{\mathsf{tgfbr}, \mathsf{stat3}, \mathsf{il6r}, \overline{\mathsf{tbet}}, \overline{\mathsf{gata3}}, \overline{\mathsf{foxp3}}\}, \{\mathsf{rorgt}\})$
$(\{\mathsf{tgfbr}, \mathsf{stat3}, \mathsf{il21r}, \overline{\mathsf{tbet}}, \overline{\mathsf{gata3}}, \overline{\mathsf{foxp3}}\}, \{\mathsf{rorgt}\})$

For the production of the negative entity $\overline{\mathsf{rorgt}}$, at this stage, we will have to add 26 reactions, one for each element of the prohibiting set we described in Example 4. It is worth noting that many of them are clearly redundant. We will address this topic in next section. Here, as an example, among the 26 reactions that produce $\overline{\mathsf{rorgt}}$ in $\mathcal{A}^+_{toy}$, we list all those reactions whose reactants involve the absence of the entity tgfbr*:*

$(\{\overline{\mathsf{tgfbr}}\}, \{\overline{\mathsf{rorgt}}\}),$ $(\{\overline{\mathsf{tgfbr}}, \overline{\mathsf{stat3}}\}, \{\overline{\mathsf{rorgt}}\}),$
$(\{\overline{\mathsf{tgfbr}}, \overline{\mathsf{il21r}}\}, \{\overline{\mathsf{rorgt}}\}),$ $(\{\overline{\mathsf{tgfbr}}, \mathsf{tbet}\}, \{\overline{\mathsf{rorgt}}\}),$
$(\{\overline{\mathsf{tgfbr}}, \mathsf{gata3}\}, \{\overline{\mathsf{rorgt}}\}),$ $(\{\overline{\mathsf{tgfbr}}, \mathsf{foxp3}\}, \{\overline{\mathsf{rorgt}}\}),$
$(\{\overline{\mathsf{il6r}}, \overline{\mathsf{tgfbr}}\}, \{\overline{\mathsf{rorgt}}\})$

It is immediate to see that whenever the first reaction is enabled the others are also enabled but redundant.

While the standard RS and its positive counterpart exploit different set of entities, respectively S and $\mathbf{S} = S \cup \bar{S}$, it is possible to mark a bijective correspondence between the states obtained at each step for the two different formalisms: the states of the standard RS are interpreted by adding negative entities corresponding to all missing entities. Exploiting this intuitive correspondence, it is proved in [8, Propositions 1–2] that standard RS and its corresponding positive version compute exactly the same states at each step.

3.2 Minimization

The procedure that we described in the previous section can produce a positive system with many redundant reactions. Indeed, this is the case for some of the 26 reactions introduced for the production of

$\overline{\mathsf{rorgt}}$ and discussed in Examples 4 and 5. Note that this is independent of whether the original reaction system exhibit some redundancy or not. In particular, reactions r_1 and r_2 of $\mathcal{A}_{\text{toy}}$ (see Example 4) are not redundant, however, if we minimize the 26 negative reactions that are responsible for the production of $\overline{\mathsf{rorgt}}$ in $\mathcal{A}^+_{\text{toy}}$, defined by applying the process described in the previous section, we get just six non redundant reactions.

As an easy example of simplification, it can be observed that any reaction $r_1 = (R_1, I_1, P)$ is redundant if we can find another reaction $r_2 = (R_2, I_2, P)$ such that $R_2 \subseteq R_1$ and $I_2 \subseteq I_1$. Other simplifications are also possible, for example if both reactions $r_1 = (R \cup \{\mathsf{a}\}, I, P)$ and $r_2 = (R, I \cup \{\mathsf{a}\}, P)$ are present, then we can replace them with the reaction $r = (R, I, P)$, because the absence/presence of $\{\mathsf{a}\}$ is not relevant. The same reasoning applies in the case of Positive RSs where the last simplification can be rephrased as follows. If we can find $r_1 = (R \cup \{\mathsf{a}\}, P)$ and $r_2 = (R \cup \{\bar{\mathsf{a}}\}, P)$ then we can eliminate both and replace them with $r = (R, P)$.

In general, we can apply a minimization process to any reaction system and derive a simplified version of the original system with fewer reactions. Minimization can be applied to both standard and Positive RSs. This process not only may reduce the number of reactions but also decrease the complexity of each individual reaction. In the following we denote with $\mu(\mathcal{A})$ the RS obtained by logically minimizing the reactions in $\mathcal{A}$. It is worth noting that, if $\mathcal{A}$ is a Positive RS, then $\mu(\mathcal{A})$ is also a Positive RS and that, in general, $\mu(\mathcal{A}^+)$ can be different from $\mu(\mathcal{A})^+$ because in the former both positive and negative reactions are minimized, while in the latter the negative reactions introduced by the transformation to Positive RS are not minimized.

Example 6 (The minimized positive RS $\mu(\mathcal{A}^+_{toy})$)**.** *Consider Example 5 where we listed all the reactions that produce* $\overline{\mathsf{rorgt}}$ *but that also require the absence of the entity* tgfbr *between the reactants:*

$$
\begin{array}{ll}
(\{\overline{\mathsf{tgfbr}}\}, \{\overline{\mathsf{rorgt}}\}), & (\{\overline{\mathsf{tgfbr}}, \overline{\mathsf{stat3}}\}, \{\overline{\mathsf{rorgt}}\}), \\
(\{\overline{\mathsf{tgfbr}}, \overline{\mathsf{il21r}}\}, \{\overline{\mathsf{rorgt}}\}), & (\{\overline{\mathsf{tgfbr}}, \mathsf{tbet}\}, \{\overline{\mathsf{rorgt}}\}), \\
(\{\overline{\mathsf{tgfbr}}, \mathsf{gata3}\}, \{\overline{\mathsf{rorgt}}\}), & (\{\overline{\mathsf{tgfbr}}, \mathsf{foxp3}\}, \{\overline{\mathsf{rorgt}}\}), \\
(\{\overline{\mathsf{il6r}}, \overline{\mathsf{tgfbr}}\}, \{\overline{\mathsf{rorgt}}\}) &
\end{array}
$$

It is easy to see that the last six reactions are subsumed by the first. Indeed, in all states where any of the other reactions are enabled, the first one is enabled as well. After minimization, only the first reaction is retained. Overall, there are only 6 minimized reactions for the production of $\overline{\mathsf{rorgt}}$*, namely:*

$$
\begin{array}{ll}
(\{\mathsf{foxp3}\}, \{\overline{\mathsf{rorgt}}\}), & (\{\mathsf{gata3}\}, \{\overline{\mathsf{rorgt}}\}), \\
(\{\overline{\mathsf{il21r}}, \overline{\mathsf{il6r}}\}, \{\overline{\mathsf{rorgt}}\}), & (\{\overline{\mathsf{stat3}}\}, \{\overline{\mathsf{rorgt}}\}), \\
(\{\overline{\mathsf{tgfbr}}\}, \{\overline{\mathsf{rorgt}}\}), & (\{\mathsf{tbet}\}, \{\overline{\mathsf{rorgt}}\})
\end{array}
$$

The complete (minimized) Positive RS $\mu(\mathcal{A}^+_{\text{toy}})$ *that we obtain is reported in Fig. A2 of the Appendix.*

4 Slicing RS Computations

In the context of programming, dynamic slicing is a technique that helps a user to debug a program by simplifying a partial execution trace, by pruning parts which are irrelevant and highlighting parts of the program which are responsible for the production of an error. In the case of RSs, the scenario that activates the slicing is given by a configuration $\frac{D_0}{C_0} \cdots \frac{D_{k-1}}{C_{k-1}} \mathsf{P}$ such that $\mathsf{P} \equiv [\mathsf{M}|D_\sigma]$, i.e., it includes a subset D_σ of entities that determines an erroneous situation. The goal of the slicing is then to highlight how the subset D_σ was originated. This includes the reactants and reactions that were responsible for producing D_σ.

Our slicing technique for RSs was initially defined in [6]. Here we can simplify it a little, by exploiting the fact that, at any computation step $W_i = C_i \cup D_i$, the intersection between the context $C_i \subseteq S_C$ and the result $D_i \subseteq S_D$ is empty. The code reported in Algorithm 1 is thus a simplified version of the one in [6]. We resume the basic ideas of the slicing algorithm here. The technique consists of three steps.

Enriched Semantics (Step **S1***).* The slicing process requires some extra information from the execution of the processes. More precisely, (1) at each operational step we need to highlight the reactions that have been applied; and (2) we need to determine the part of the context which adds to the previous state the entities which are necessary to produce the marked entities in the following state. For solving (1) and (2), in Section 2.2 we have introduced an enriched semantics that records the necessary information. Given the history enriched process $\mathsf{H}_0 \cdots \mathsf{H}_{m-1}[\mathsf{M}|D_m]$ we let the slicing take as input the trace $\mathsf{H}_0 \cdots \mathsf{H}_{m-1} \frac{D_m}{\varnothing}$.

Marking the state (Step **S2***).* Given the final configuration $\frac{D_m}{C_m}$ of a partial computation, the slicing is also dependent on a subset $D_\sigma \subseteq D_m$ for which we want to provide an explanation. The set D_σ can be provided by the user, or selected automatically by a monitor associated with a modal formula, as explained in [6].

Input: - a reaction system $\mathcal{A}$
- a trace $T = \frac{D_0}{C_0} \xrightarrow{N_1} \cdots \xrightarrow{N_m} \frac{D_m}{C_m}$
- a marking $D_\sigma \subseteq D_m$

Output: a sliced trace $\frac{D'_0}{C'_0} \xrightarrow{N'_1} \cdots \xrightarrow{N'_m} \frac{D_\sigma}{C_m}$

begin
  let $D'_m = D_\sigma$
  for $i = m$ *to* 1 **do**
    let $D'_{i-1} = \varnothing \wedge C'_{i-1} = \varnothing \wedge N'_i = \varnothing$
    for $r_j = (R_j, I_j, P_j) \in N_i$ *such that* $(D'_i \cap P_j \neq \varnothing)$ **do**
      let $N'_i = N'_i \cup \{j\}$
      let $C'_{i-1} = C'_{i-1} \cup (R_j \cap S_C)$
      let $D'_{i-1} = D'_{i-1} \cup (R_j \cap S_D)$
    end
  end
end

Algorithm 1: Trace Slicer $s(\mathcal{A}, T, D_\sigma)$.

Trace Slice (Step **S3***).* Starting from the pair $\frac{D_\sigma}{C_m}$ denoting the user's (or monitor) marking and proceeding backwards, we iteratively apply a slicing step that deletes from the partial computation all information not related to D_σ.

Slicing Algorithm

Let us now explain how the slicing Algorithm 1 works.

Given the trace $\frac{D_0}{C_0} \xrightarrow{N_1} \cdots \xrightarrow{N_m} \frac{D_m}{C_m}$ and a subset of marked entities $D'_m = D_\sigma \subseteq D_m$ our algorithm returns a sliced trace $\frac{D'_0}{C'_0} \xrightarrow{N'_1} \cdots \xrightarrow{N'_m} \frac{D'_m}{C_m}$ such that $D'_i \subseteq D_i$, $C'_i \subseteq C_i$ for any $i \in [0, m]$ and $N'_i \subseteq N_i$ for any index $i \in [1, m]$. The sliced trace contains only the (usually much smaller) subsets of entities and reactions which are necessary for deriving the marked entities. The backward slicing process starts from $i = m$, which is decremented by 1 at each iteration. Now, let us consider the generic iteration i of the slicer. Marking the relevant information $\frac{D'_i}{C'_i}$, in previous state $\frac{D_{i-1}}{C_{i-1}}$ requires analyzing the reactions which have been applied at step $i-1$. So, if $j \in N_{i-1}$, we check if $r_j = (R_j, I_j, P_j)$, produces at least one entity which is marked in the next state. If this is the case, j is added to the set of marked rules. Then, it is necessary to check which part of the entities in R_j comes from the context C_{i-1} and which part derives from D_{i-1}. Since r_j was enabled in $W_{i-1} = D_{i-1} \cup C_{i-1}$, it must be the case that $R_j \subseteq W_{i-1}$. Therefore we just include $R_j \cap S_C$ in C'_{i-1} and $R_j \cap S_D$ in D'_{i-1}.

Example 7 ($s(\mathcal{A}_{\text{toy}}, T, D_\sigma)$)**.** *Consider the RS* $\mathcal{A}_{\text{toy}}$ *and the process* $\mathsf{H_0H_1H_2P_3}$ *with history from Example 3, namely:*

$$\frac{D_0}{C_0} \xrightarrow{\{3,4,6\}} \frac{D_1}{C_1} \xrightarrow{\{7\}} \frac{D_2}{C_2} \xrightarrow{\{8\}} [\mathsf{Reacts} \mid \{\mathsf{tbet}\} \mid \mathbf{0}]$$

where $\frac{D_0}{C_0} = \frac{\{\mathsf{il23r,il21}\}}{\{\mathsf{tcr}\}}$, $\frac{D_1}{C_1} = \frac{\{\mathsf{il21r,stat3,nfat}\}}{\{\mathsf{il27}\}}$, *and* $\frac{D_2}{C_2} = \frac{\{\mathsf{stat1}\}}{\varnothing}$. *Letting* $D_\sigma = \{\mathsf{tbet}\}$, $\frac{D_3}{C_3} = \frac{\{\mathsf{tbet}\}}{\varnothing}$ *and*

$$T = \frac{D_0}{C_0} \xrightarrow{\{3,4,6\}} \frac{D_1}{C_1} \xrightarrow{\{7\}} \frac{D_2}{C_2} \xrightarrow{\{8\}} \frac{D_3}{C_3}$$

we now compute the slice $s(\mathcal{A}_{\text{toy}}, T, D_\sigma)$ *that highlights the entities whose presence was exploited to produce* $D_\sigma = \{\mathsf{tbet}\}$. *The result of the slicing algorithm is the following:*

$$T' = \frac{\varnothing}{\{\mathsf{tcr}\}} \xrightarrow{\{6\}} \frac{\{\mathsf{nfat}\}}{\{\mathsf{il27}\}} \xrightarrow{\{7\}} \frac{\{\mathsf{stat1}\}}{\varnothing} \xrightarrow{\{8\}} \frac{\{\mathsf{tbet}\}}{\varnothing}.$$

4.1 Slicing Positive RSs

The slicing Algorithm 1 can only capture dependencies related to reactants, not those related to inhibitors: if $r = (R, I, P)$ is responsible for the production of an entity $\mathsf{a} \in P$ that belongs to D_i, then we can track its dependencies $R \cap S_D \subseteq D_{i-1}$, but we miss the information that the entity a appeared in D_i because all inhibitors in $I \cap S_D$ are not present in D_{i-1}. However, if we transform the RS $\mathcal{A}$ to its positive version $\mathcal{A}^+$ and then apply the slicing to the trace generated in $\mathcal{A}^+$, then such "negative" dependencies can also be tracked automatically.

Example 8 (The slice $s(\mathcal{A}^+_{\text{toy}}, T^+, D_\sigma)$)**.** *Let us consider the Positive RS* $\mathcal{A}^+_{\text{toy}}$ *outlined in Example 5. The original trace*

$$T = \frac{D_0}{C_0} \xrightarrow{\{3,4,6\}} \frac{D_1}{C_1} \xrightarrow{\{7\}} \frac{D_2}{C_2} \xrightarrow{\{8\}} \frac{D_3}{C_3}$$

reported in Example 7 for $\mathcal{A}_{\text{toy}}$ *can be replicated in the Positive RS* $\mathcal{A}^+_{\text{toy}}$ *by including all negative counterparts of missing entities to obtain the following trace (for simplicity, we omit transition labels as they are not relevant).*

$$T^+ = \frac{\mathbf{D}_0}{\mathbf{C}_0} \to \frac{\mathbf{D}_1}{\mathbf{C}_1} \to \frac{\mathbf{D}_2}{\mathbf{C}_2} \to \frac{\mathbf{D}_3}{\mathbf{C}_3}$$

where $\mathbf{D}_i = D_i \cup \overline{(S_D \setminus D_i)}$ *and* $\mathbf{C}_i = C_i \cup \overline{(S_C \setminus C_i)}$ *for any* $i \in [0, 3]$. *While each* $\mathbf{D}_i$ *contains either the positive of negative instance of each entity in* S_D, *the slicing algorithm* $s(\mathcal{A}^+_{\text{toy}}, T^+, D_\sigma)$ *applied to the positive trace extracts just the set of positive and negative entities whose presence helps to produce* $D_\sigma = \{\mathsf{tbet}\} \subseteq \mathbf{D}_3$*:*

$$\hat{T}^+ = \frac{\{\overline{\mathsf{foxp3}}, \overline{\mathsf{il6}}\}}{\{\mathsf{tcr}\}} \to \frac{\{\overline{\mathsf{il6r}}, \overline{\mathsf{tgfbr}}, \mathsf{nfat}\}}{\{\mathsf{il27}\}} \to \frac{\{\overline{\mathsf{foxp3}}, \overline{\mathsf{rorgt}}, \mathsf{stat1}\}}{\varnothing} \to \frac{\{\mathsf{tbet}\}}{\varnothing}.$$

It is now interesting to observe, e.g., that the absence of both foxp3 *and* il6 *in the initial state plays a role for the production of* tbet *after three steps.*

4.2 Slicing Minimized (Positive) RSs

While redundant reactions have no visible effect on the computation of RSs, we can now note that they can instead add some noise to the detection of causes using the slicing algorithm. In fact, a redundant reaction can introduce additional dependencies that are not strictly necessary for the release of their products. This side effect can be easily shown building on our running toy example.

As a matter of notation, given a RS $\mathcal{A}$, a trace T and a marking D_σ, we denote with $s(\mathcal{A}, T, D_\sigma)$ the (sliced) trace obtained by applying the Trace Slicer given in Algorithm 1, and with $ns(\mathcal{A}, T, D_\sigma)$ the outcome of the Dynamic Negative Trace Slicer given in Algorithm 2.

Example 9 (The slice $s(\mu(\mathcal{A}^+_{\text{toy}}), T^+, D_\sigma)$)**.** *Let us consider the Positive RS $\mu(\mathcal{A}^+_{\text{toy}})$, which is in minimized form, as outlined in Example 6. The removal of redundant reactions preserves the trace sketched in Example 8 (except for the possible different naming of reactions, here omitted from the transition labels):*

$$T^+_\mu = \frac{\mathbf{D_0}}{\mathbf{C_0}} \rightarrow \frac{\mathbf{D_1}}{\mathbf{C_1}} \rightarrow \frac{\mathbf{D_2}}{\mathbf{C_2}} \rightarrow \frac{\mathbf{D_3}}{\mathbf{C_3}}$$

However, the slicing algorithm applied to the minimized positive computation allows to remove some spurious dependencies that were introduced by redundant reactions. In fact, by running $s(\mu(\mathcal{A}^+_{\text{toy}}), T^+, D_\sigma)$ we get the following slice (where, again, transition labels are omitted for simplicity):

$$\hat{T}^+_\mu = \frac{\{\overline{\text{foxp3}}\}}{\{\text{tcr}\}} \rightarrow \frac{\{\overline{\text{tgfbr}}, \text{nfat}\}}{\{\text{il27}\}} \rightarrow \frac{\{\overline{\text{foxp3}}, \overline{\text{rorgt}}, \text{stat1}\}}{\varnothing} \rightarrow \frac{\{\text{tbet}\}}{\varnothing}$$

With respect to the slice in Example 8, it is worth observing that the absence of il6 *from the initial state is no longer needed for the production of* tbet *after three steps, while the absence of* foxp3 *is still meaningful.*

Examples 8 and 9 show a general property of our slicing algorithm that will formalized by Theorem 2: at any slicing step, the slice $\frac{D'_i}{C'_i}$ computed in a RS $\mathcal{A}$ and the slice $\frac{\hat{D}'_i}{\hat{C}'_i}$ computed in the minimized RS $\mu(\mathcal{A})$ are such that $\hat{D}'_i \subseteq D'_i$ and $\hat{C}'_i \subseteq C'_i$, which we write more concisely as $\frac{\hat{D}'_i}{\hat{C}'_i} \leq \frac{D'_i}{C'_i}$.

Input: - a reaction system $\mathcal{A}$
- a trace $T = \frac{D_0}{C_0} \xrightarrow{N_1} \cdots \xrightarrow{N_m} \frac{D_m}{C_m}$
- a marking $D_\sigma \subseteq D_m$

Output: a sliced trace possibily extended with negative entities $\frac{\mathbf{D'_0}}{\mathbf{C'_0}} \xrightarrow{N'_1} \cdots \xrightarrow{N'_m} \frac{D_\sigma}{C_m}$

begin
  let $\mathbf{D'_m} = D_\sigma$
  for $i = m$ *to* 1 **do**
    let $\mathbf{D'_{i-1}} = \varnothing \wedge \mathbf{C'_{i-1}} = \varnothing \wedge N'_i = \varnothing$
    for $r_j = (R_j, I_j, P_j) \in N_i$ *such that* $(\mathbf{D'_i} \cap P_j \neq \varnothing)$ **do**
      let $N'_i = N'_i \cup \{j\}$
      let $\mathbf{C'_{i-1}} = \mathbf{C'_{i-1}} \cup (R_j \cap S_C) \cup (\overline{I_j \cap S_C})$
      let $\mathbf{D'_{i-1}} = \mathbf{D'_{i-1}} \cup (R_j \cap S_D) \cup (\overline{I_j \cap S_D})$
    end
    for $r_j = (R_j, I_j, P_j) \notin N_i$, *such that* $(\mathbf{D'_i} \cap \overline{P_j} \neq \varnothing)$ **do**
      let $\mathbf{C'_{i-1}} = \mathbf{C'_{i-1}} \cup ((R_j \cap S_C) \setminus C_{i-1}) \cup (I_j \cap C_{i-1})$
      let $\mathbf{D'_{i-1}} = \mathbf{D'_{i-1}} \cup ((R_j \cap S_D) \setminus D_{i-1}) \cup (I_j \cap D_{i-1})$
    end
  end
end

Algorithm 2: Dynamic Negative Trace Slicer $ns(\mathcal{A}, T, D_\sigma)$.

5 Dynamic Negative Slicing

In order to account for negative dependencies (i.e., dependencies from entities that have to be absent) the slicing approach defined in the previous section requires the RS under analysis to be translated into positive form. Such a translation can be computationally expensive and could generate a number of new reactions that is exponential in the size of the original RS. As a consequence, in this section we propose an alternative slicing approach that captures negative dependencies by working on the original RS specification, without translating it into positive form. The idea is to extend the Trace Slicer algorithm in order to add (dynamically) an entity in negative form in the sliced trace every time the absence of such an entity was necessary for the application of a reaction. Those entities are then traced back by looking for all of the reactions whose non-application caused them to be absent. The new slicing approach, called Dynamic Negative Trace Slicer, is formalized in Algorithm 2.

Dynamic Negative Slicing Algorithm

The dynamic negative slicing Algorithm 2 follows the same scheme as the slicing Algorithm 1, by computing with backward iteration the slice at step $i-1$ from the one at step i.

The first inner loop deals with the reactions N_i applied at the step i: the difference w.r.t. Algorithm 1

is the inclusion of the negative versions of inhibitors $\overline{I_j}$ in the slice $\frac{\mathbf{D'_{i-1}}}{\mathbf{C'_{i-1}}}$, partitioned according to S_C and S_D, respectively. In fact, the reaction r_j that is responsible for the production of the entities in $\mathbf{D'_i} \cap P_j \neq \varnothing$ is enabled thanks to the presence of its reactants R_j and to the absence of its inhibitors I_j, which are recorded as negative entities $\overline{I_j}$.

The second inner loop deals with the causes of negative dependencies. To this aim, let us remind that a negative entity $\overline{\mathsf{a}} \in \mathbf{D'_i}$ means that the absence of a must be tracked. Thus, we take any reaction r_j such that could have produced a but was not enabled (i.e., such that $r_j = (R_j, I_j, P_j) \notin N_i$ and $\mathbf{D'_i} \cap \overline{P_j} \neq \varnothing$) and record the reasons why it was not enabled: it could be because some of its reactants were absent in $W_{i-1} = D_{i-1} \cup C_{i-1}$ or because some of its inhibitors were present in W_{i-1}. Correspondingly we add the negative causes $R_j \setminus W_{i-1}$ and the positive causes $I_j \cap W_{i-1}$ to the slice, partitioned according to S_C and S_D.

Example 10 (The slice $ns(\mathcal{A}_{\text{toy}}, T, D_\sigma)$)**.** *Let us supply as input to the Dynamic Negative Trace Slicer algorithm the trace T from Example 7:*

$$\frac{\{\mathsf{il23r},\mathsf{il21}\}}{\{\mathsf{tcr}\}} \xrightarrow{\{3,4,6\}} \frac{\{\mathsf{il21r},\mathsf{stat3},\mathsf{nfat}\}}{\{\mathsf{il27}\}} \xrightarrow{\{7\}} \frac{\{\mathsf{stat1}\}}{\varnothing} \xrightarrow{\{8\}} \frac{\{\mathsf{tbet}\}}{\varnothing}$$

The result of $ns(\mathcal{A}_{\text{toy}}, T, D_\sigma)$ spells out the positive and negative dependencies that are responsible for the production of the entity tbet*. Then, we obtain as output the slice similar to the one in Example 8, where the numbering of the reactions refers to that in Example 1, namely*

$$\frac{\{\overline{\mathsf{foxp3}},\overline{\mathsf{il6}}\}}{\{\mathsf{tcr}\}} \xrightarrow{\{6\}} \frac{\{\overline{\mathsf{il6r}},\overline{\mathsf{tgfbr}},\mathsf{nfat}\}}{\{\mathsf{il27}\}} \xrightarrow{\{7\}} \frac{\{\overline{\mathsf{foxp3}},\overline{\mathsf{rorgt}},\mathsf{stat1}\}}{\varnothing} \xrightarrow{\{8\}} \frac{\{\mathsf{tbet}\}}{\varnothing}$$

This is because, for example, the production of tbet *is caused by the enabling of the reaction r_8 whose inhibitors* foxp3 *and* rorgt *are then recorded in $\mathbf{D'_2}$ in the form of negative dependencies. At the next iteration it is discovered that no reaction is responsible for the missing production of* foxp3*, which does not appear as a product of any reaction although* foxp3 $\in S_D$*. Instead* rorgt *could have been produced by r_1 or r_2, but the first is disabled by the absence of both* tgfbr *and* il6r *and the second by the absence of* tgfbr*, which are thus recorded in $\mathbf{D'_1}$. Finally, no reaction is responsible for the missing production of* tgfbr*, because, alike* foxp3*, the entity* tgfbr $\in S_D$ *does not appear as a product of any reaction. Instead,* il6r *could have been produced only by r_5 which is disabled by the absence of* il6*, which must be recorded in $\mathbf{D'_0}$. The absence of* foxp3 *is also recorded in $\mathbf{D'_0}$ because it is required in order to enable r_6 and produce* nfat $\in \mathbf{D'_1}$.

5.1 Comparing Slicing Approaches

The general problem we would like to solve can be formulated in terms of explainability: given a computational witness of some phenomenon, like a trace that ends up with the production of some given entities, we would like to isolate a set of dependencies that explain the occurrence of the observed phenomenon.

In the previous sections, we have witnessed that different slicing techniques applied over equivalent RSs can return different results that can be useful to focus on the chain of reactions that produced the phenomenon. Here we need to understand which guarantees are provided and which sliced trace is more convenient for us. In the following we show that the slicing of minimized positive RSs offers the best guarantees.

Remark 4. *It is worth remarking that the behaviour of RSs is in general not monotone, in the sense that $W \subseteq W'$ does not necessarily imply $res_A(W) \subseteq res_A(W')$. This is due to the presence of inhibitors: clearly the reaction $r = (R, I, P)$ is enabled in the set R, but not in the set $R \cup I \supseteq R$ when $I \neq \varnothing$. Positive RSs exhibit a monotone behaviour just because their reactions have empty sets of inhibitors.*

In order to compare the different notions of slicing, let us introduce a few notations. Given a RS $\mathcal{A}$ and its positive version $\mathcal{A}^+$ obtained from the encoding defined in Definition 7, as mentioned in Section 3.2, we denote with $\mu(\mathcal{A})$ and $\mu(\mathcal{A}^+)$ the RS and the PRS obtained by logically minimizing the reactions in $\mathcal{A}$ and $\mathcal{A}^+$, respectively. Moreover, given a trace $T = \frac{D_0}{C_0} \xrightarrow{N_1} \frac{D_1}{C_1} \xrightarrow{N_2} \cdots \xrightarrow{N_m} \frac{D_m}{C_m}$ obtained from a reaction system $\mathcal{A}$, we denote with $T_\mu = \frac{D_0}{C_0} \xrightarrow{N'_1} \frac{D'_1}{C_1} \xrightarrow{N'_2} \cdots \xrightarrow{N'_m} \frac{D'_m}{C_m}$ the trace obtained from $\mu(\mathcal{A})$ by starting with the same initial entities D_0 and by assuming the same context sequence $C_0, \ldots, C_m$ to be provided. Note that since minimization results into an equivalent RS, it holds $D'_i = D_i$ for every $i \in [1, m]$, while N_i and N'_i are in general different.[2]

Similarly, given the same trace T, we denote with $T^+ = \frac{\mathbf{D_0}}{\mathbf{C_0}} \xrightarrow{N'_1} \frac{\mathbf{D_1}}{\mathbf{C_1}} \xrightarrow{N'_2} \cdots \xrightarrow{N'_m} \frac{\mathbf{D_m}}{\mathbf{C_m}}$ the trace obtained from $\mu(\mathcal{A}^+)$ by starting with the initial entities $\mathbf{D_0}$ obtained by adding to D_0 the negative versions of the internal entities that are not present in D_0, and by assuming a context sequence

[2] Since minimization removes redundant reactions, one could expect that $N'_i \subseteq N_i$, but minimization can also replace a set of reactions with a new simpler reaction, which can then appear in N'_i.

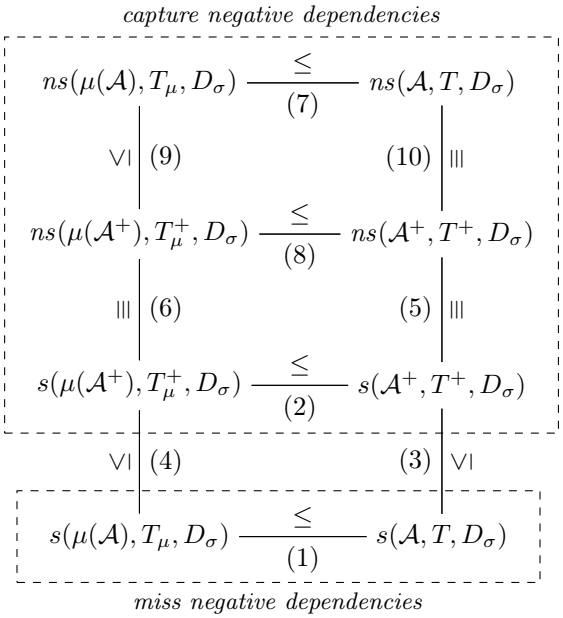


Fig. 3: Given a RS $\mathcal{A}$ one of its traces T and a set of marked entities D_σ the figure compares the output of the two slicing algorithms when executed on the ordinary RS $\mathcal{A}$ and its positive version $\mathcal{A}^+$ with or without minimization μ.

$\mathbf{C_0}, \ldots, \mathbf{C_m}$ obtained by adding negative versions of the contextual entities not present in each element of $C_0, \ldots, C_m$. The behavioral correspondence between $\mathcal{A}$ and $\mathcal{A}^+$ guaranteed by the encoding defined in Definition 7 ensures that it holds $\mathbf{D}_i = D_i \cup \overline{(S_D \setminus D_i)}$ and $\mathbf{C}_i = C_i \cup \overline{(S_C \setminus C_i)}$ for any $i \in [0, m]$.

Definition 8 (Trace Partial Order, $\leq$)**.** *Given two RSs $\mathcal{A}_1 = (S_1, A_1)$ and $\mathcal{A}_2 = (S_2, A_2)$ with $S_1 \cap S_2 \neq \varnothing$, and two traces T_1 and T_2 obtained from $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, and such that*
$T_1 = \frac{D^1_0}{C^1_0} \xrightarrow{N^1_1} \cdots \xrightarrow{N^1_m} \frac{D^1_m}{C^1_m}$ *and*
$T_2 = \frac{D^2_0}{C^2_0} \xrightarrow{N^2_1} \cdots \xrightarrow{N^2_m} \frac{D^2_m}{C^2_m}$,
we have that $T_1 \leq T_2$ if and only if for every $i \in [0, m]$ it holds $D^1_i \subseteq D^2_i$ and $C^1_i \subseteq C^2_i$. We write $T_1 \equiv T_2$ when $T_1 \leq T_2$ and $T_2 \leq T_1$.

From the previous discussion, it follows immediately that for any trace T it holds $T_\mu \equiv T \leq T^+ \equiv T^+_\mu$.

Theorem 1 (Slicing relationships)**.** *Given a RS $\mathcal{A}$, a trace T and a marking D_σ, all the relationships summarized by the diagram in Fig. 3 hold.*

Proof. Let us assume

- $T = \frac{D_0}{C_0} \xrightarrow{N_1} \frac{D_1}{C_1} \xrightarrow{N_2} \cdots \xrightarrow{N_m} \frac{D_m}{C_m}$,
- $T_\mu = \frac{D_0}{C_0} \xrightarrow{M_1} \frac{D_1}{C_1} \xrightarrow{M_2} \cdots \xrightarrow{M_m} \frac{D_m}{C_m}$,
- $T^+ = \frac{\mathbf{D_0}}{\mathbf{C_0}} \xrightarrow{N^+_1} \frac{\mathbf{D_1}}{\mathbf{C_1}} \xrightarrow{N^+_2} \cdots \xrightarrow{N^+_m} \frac{\mathbf{D_m}}{\mathbf{C_m}}$, and
- $T^+_\mu = \frac{\mathbf{D_0}}{\mathbf{C_0}} \xrightarrow{M^+_1} \frac{\mathbf{D_1}}{\mathbf{C_1}} \xrightarrow{M^+_2} \cdots \xrightarrow{M^+_m} \frac{\mathbf{D_m}}{\mathbf{C_m}}$

with $T_\mu \equiv T \leq T^+ \equiv T^+_\mu$, as $\mathbf{D}_i = D_i \cup \overline{(S_D \setminus D_i)}$ and $\mathbf{C}_i = C_i \cup \overline{(S_C \setminus C_i)}$. In order to prove the relation between the output of any two slicings in Fig. 3 we focus on a generic iteration i, because we can then extend the relation inductively to the whole trace.

We consider the various cases separately following the same pattern, which can be summarized as follows: given $T \leq \hat{T}$ we want to prove that $T' \leq \hat{T}'$, where T' is the slice returned by the slicing s_1 applied to T and $\hat{T}'$ is the slice returned by the slicing s_2 applied to $\hat{T}$. To this aim we show that for every transition $\frac{D_{i-1}}{C_{i-1}} \xrightarrow{N_i} \frac{D_i}{C_i}$ of T and $\frac{\hat{D}_{i-1}}{\hat{C}_{i-1}} \xrightarrow{\hat{N}_i} \frac{\hat{D}_i}{\hat{C}_i}$ of $\hat{T}$, if $\frac{D'_i}{C'_i} \leq \frac{\hat{D}'_i}{\hat{C}'_i}$ we then have $\frac{D'_{i-1}}{C'_{i-1}} \leq \frac{\hat{D}'_{i-1}}{\hat{C}'_{i-1}}$, where the primed versions denote the output of the slicing algorithms. As for $i = m$ we always have $\frac{D'_m}{C'_m} = \frac{D_\sigma}{\varnothing} = \frac{\hat{D}'_m}{\hat{C}'_m}$ the property can then be trivially extended to the whole sliced trace T' returned from s_1 and $\hat{T}'$ returned from s_2. As a matter of notation, we leave implicit that $W_i \triangleq D_i \cup C_i$ and $\hat{W}_i \triangleq \hat{D}_i \cup \hat{C}_i$ for any $i \in [0, m]$. The proof details are spelled out for D'_i components, but the case of context components C'_i is similar.

1) $T' = s(\mu(\mathcal{A}), T_\mu, D_\sigma) \leq \hat{T}' = s(\mathcal{A}, T, D_\sigma)$. Since $T_\mu \equiv T$, we have $\frac{\hat{D}_i}{\hat{C}_i} = \frac{D_i}{C_i}$ for every $i \in [0, m]$. Let us take $\mathsf{a} \in D'_{i-1}$ and prove that $\mathsf{a} \in \hat{D}'_{i-1}$. Since $\mathsf{a} \in D'_{i-1}$, it must be the reactant of some reaction $r_j = (R_j, I_j, P_j)$ such that $en_{r_j}(W_{i-1})$ and $D'_i \cap P_j \neq \varnothing$. Since $D'_i \subseteq \hat{D}'_i$ then $\hat{D}'_i \cap P_j \neq \varnothing$. Since r_j is in $\mu(\mathcal{A})$ it means that there exists some reaction $r = (R, I, P_j)$ in $\mathcal{A}$ such that $R_j \subseteq R$ and $I_j \subseteq I$, with r enabled at $\hat{W}_{i-1}$. Therefore, $\mathsf{a} \in R$ is added to $\hat{D}'_{i-1}$.

2) $T' = s(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma) \leq \hat{T}' = s(\mathcal{A}^+, T^+, D_\sigma)$. This case is completely analogous to the previous one, except for the fact that we consider Positive RSs.

3) $T' = s(\mathcal{A}, T, D_\sigma) \leq \hat{T}' = s(\mathcal{A}^+, T^+, D_\sigma)$. We have $\hat{D}_i = D_i \cup \overline{(S_D \setminus D_i)}$ and $\hat{C}_i = C_i \cup \overline{(S_C \setminus C_i)}$ for every $i \in [0, m]$. Let us take $\mathsf{a} \in D'_{i-1}$ and prove that $\mathsf{a} \in \hat{D}'_{i-1}$. Since $\mathsf{a} \in D'_{i-1}$, it must be the reactant of some reaction $r_j = (R_j, I_j, P_j)$ such that $en_{r_j}(W_{i-1})$ and $D'_i \cap P_j \neq \varnothing$. Since $D'_i \subseteq \hat{D}'_i$ then $\hat{D}'_i \cap P_j \neq \varnothing$.

Since r_j is in $\mathcal{A}$ it means that there exists some positive reaction $r = (R_j \cup \overline{I_j}, P_j)$ in $\mathcal{A}^+$ that is enabled at $\hat{W}_{i-1}$. Therefore, $\mathsf{a} \in R_j \subseteq R_j \cup \overline{I_j}$ is added to $\hat{D}'_{i-1}$.

4) $T' = s(\mu(\mathcal{A}), T_\mu, D_\sigma) \leq \hat{T}' = s(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma)$. We have $\hat{D}_i = D_i \cup \overline{(S_D \setminus D_i)}$ and $\hat{C}_i = C_i \cup \overline{(S_C \setminus C_i)}$ for every $i \in [0, m]$. Let us take $\mathsf{a} \in D'_{i-1}$ and prove that $\mathsf{a} \in \hat{D}'_{i-1}$. Since $\mathsf{a} \in D'_{i-1}$, it must be the reactant of some reaction $r_j = (R_j, I_j, P_j)$ such that $en_{r_j}(W_{i-1})$ and $D'_i \cap P_j \neq \varnothing$. Since $D'_i \subseteq \hat{D}'_i$ then $\hat{D}'_i \cap P_j \neq \varnothing$. Since r_j is in $\mu(\mathcal{A})$ it means that there exists some positive reaction $r = (R \cup \overline{I}, P)$ in $\mathcal{A}^+$ that is enabled at $\hat{W}_{i-1}$. Therefore, $\mathsf{a} \in R \subseteq R \cup \overline{I}$ is added to $\hat{D}'_{i-1}$.

5) $T' = s(\mathcal{A}^+, T^+, D_\sigma) \equiv \hat{T}' = ns(\mathcal{A}^+, T^+, D_\sigma)$. Since the equivalence relation $\equiv$ is considered instead of the order relation $\leq$, we simply assume that $\frac{D'_i}{C'_i} = \frac{\hat{D}'_i}{\hat{C}'_i}$ in order to prove that $\frac{D'_{i-1}}{C'_{i-1}} = \frac{\hat{D}'_{i-1}}{\hat{C}'_{i-1}}$. Since $\mathcal{A}^+$ is a Positive RS, all sets I_j appearing in the Dynamic Negative Trace Slicer are empty. Therefore the first inner loop behaves exactly as the loop in the ordinary Trace Slicer and cannot add negative entities to D'_i and $\hat{D}'_i$. Moreover, the second inner loop becomes vacuous because the condition $\mathbf{D'_i} \cap \overline{P_j} \neq \varnothing$ is never met, from which we conclude that $D'_{i-1} = \hat{D}'_{i-1}$.

6) $T' = s(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma) \equiv \hat{T}' = ns(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma)$. This case is completely analogous to the previous one and thus omitted.

7) $T' = ns(\mu(\mathcal{A}), T_\mu, D_\sigma) \leq \hat{T}' = ns(\mathcal{A}, T, D_\sigma)$. Since $T_\mu \equiv T$, we have $\frac{\hat{D}_i}{\hat{C}_i} = \frac{D_i}{C_i}$ for every $i \in [0, m]$. Let us take $\mathbf{a} \in D'_{i-1}$ and prove that $\mathbf{a} \in \hat{D}'_{i-1}$. We consider separately the case of positive entities and negative ones.

- If $\mathbf{a}$ is a positive entity $\mathsf{a} \in D'_{i-1}$, there are two possibilities: it can be the reactant of some enabled reaction $r_j = (R_j, I_j, P_j)$ such that $D'_i \cap P_j \neq \varnothing$ or the inhibitor of some inhibited reaction $r_j = (R_j, I_j, P_j)$ such that $D'_i \cap \overline{P_j} \neq \varnothing$. In any case, since r_j is in $\mu(\mathcal{A})$ it means that there exists some corresponding reaction $r = (R, I, P_j)$ in $\mathcal{A}$ such that $R_j \subseteq R$ and $I_j \subseteq I$. Therefore a is added to $\hat{D}'_{i-1}$.
- If $\mathbf{a}$ is a negative entity $\overline{\mathsf{a}} \in D'_{i-1}$, there are two possibilities: it can be the inhibitor of some enabled reaction $r_j = (R_j, I_j, P_j)$ such that $D'_i \cap P_j \neq \varnothing$ or the missing reactant of some inhibited reaction $r_j = (R_j, I_j, P_j)$ such that $D'_i \cap \overline{P_j} \neq \varnothing$. In any case, since r_j is in $\mu(\mathcal{A})$ it means that there exists some corresponding reaction $r = (R, I, P_j)$ in $\mathcal{A}$ such that $R_j \subseteq R$ and $I_j \subseteq I$. Therefore $\overline{\mathsf{a}}$ is added to $\hat{D}'_{i-1}$.

8) $T' = ns(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma) \leq \hat{T}' = ns(\mathcal{A}^+, T^+, D_\sigma)$. This case is completely analogous to the previous one, except for the fact that we consider Positive RSs.

9) $T' = ns(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma) \leq \hat{T}' = ns(\mu(\mathcal{A}), T_\mu, D_\sigma)$. We have $D_i = \hat{D}_i \cup \overline{(S_D \setminus \hat{D}_i)}$ and $C_i = \hat{C}_i \cup \overline{(S_C \setminus \hat{C}_i)}$ for every $i \in [0, m]$. Moreover, since the RS $\mu(\mathcal{A}^+)$ is positive, we know that $T' = ns(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma) = s(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma)$, i.e., that the slice returned by the Dynamic Negative Trace Slicer is the same as the one returned by the Trace Slicer. Let us take $\mathbf{a} \in D'_{i-1}$ and prove that $\mathbf{a} \in \hat{D}'_{i-1}$. Since $\mathbf{a} \in D'_{i-1}$, it must be the reactant of some reaction $r_j = (\mathbf{R_j}, \mathbf{P_j})$ in $\mu(\mathcal{A}^+)$ such that $en_{r_j}(W_i)$ and $D'_i \cap \mathbf{P_j} \neq \varnothing$ We consider separately the case of r_j being a positive reaction or a negative one.

- If r_j is a (minimal) positive reaction it means that there must exist a corresponding (minimal) reaction $r'_j = (R_j, I_j, P_j)$ in $\mu(\mathcal{A})$ such that $\mathbf{R_j} = R_j \cup \overline{I_j}$. Therefore r'_j is enabled at $\hat{W}_i$ and $\mathbf{a}$ is added to $\hat{D}'_{i-1}$.
- If r_j is a (minimal) negative reaction it means that that $\mathbf{R_j}$ is a prohibiting set for some missing entity in $\hat{D}_i$, i.e., $\mathbf{R_j} = I \cup \overline{R}$ for some suitable sets R and I. Therefore, a is added to $\hat{D}'_{i-1}$.

10) $T' = ns(\mathcal{A}, T, D_\sigma) \equiv \hat{T}' = ns(\mathcal{A}^+, T^+, D_\sigma)$. This is the most interesting result. We have $\hat{D}_i = D_i \cup \overline{(S_D \setminus D_i)}$ and $\hat{C}_i = C_i \cup \overline{(S_C \setminus C_i)}$ for every $i \in [0, m]$. Let us prove that $\mathbf{a} \in D'_{i-1}$ iff $\mathbf{a} \in \hat{D}'_{i-1}$. We consider separately the case of positive entities and negative ones.

- Let us take $\mathsf{a} \in D'_{i-1}$. Since $\mathsf{a} \in D'_{i-1}$, there are two possibilities: it can be the reactant of some reaction $r_j = (R_j, I_j, P_j)$ such that $en_{r_j}(W_{i-1})$ and $D'_i \cap P_j \neq \varnothing$ or the inhibitor of some inhibited reaction $r_j = (R_j, I_j, P_j)$ such that $D'_i \cap \overline{P_j} \neq \varnothing$. In the first case, since r_j is in $\mathcal{A}$, there exists some corresponding positive reaction $r = (R_j \cup \overline{I_j}, P_j)$ in $\mathcal{A}^+$ that is enabled at $\hat{W}_{i-1}$. In the second case, there exists some prohibiting set $I \cup \overline{R}$ for any of the negative entities $\overline{\mathsf{b}} \in D'_i \cap \overline{P_j}$ such that $\mathsf{a} \in I$ serves to inhibit the reaction r_j, with $I \cup \overline{R} \subseteq \hat{W}_{i-1}$. Therefore, we can find a reaction $(I \cup \overline{R}, \{\overline{\mathsf{b}}\})$ in $\mathcal{A}^+$ that is enabled at $\hat{W}_{i-1}$. In both cases, $\mathsf{a} \in D'_{i-1}$ iff $\mathsf{a} \in \hat{D}'_{i-1}$.
- Let us take $\overline{\mathsf{a}} \in D'_{i-1}$. Since $\overline{\mathsf{a}} \in D'_{i-1}$, there are two possibilities: it can be the inhibitor of some reaction

$r_j = (R_j, I_j, P_j)$ such that $en_{r_j}(W_{i-1})$ and $D'_i \cap P_j \neq \varnothing$ or the missing reactant of some inhibited reaction $r_j = (R_j, I_j, P_j)$ such that $D'_i \cap \overline{P_j} \neq \varnothing$. In the first case, since r_j is in $\mathcal{A}$, there exists some corresponding positive reaction $r = (R_j \cup \overline{I_j}, P_j)$ in $\mathcal{A}^+$ that is enabled at $\hat{W}_{i-1}$. In the second case, there exists some prohibiting set $I \cup \overline{R}$ for any of the negative entities $\overline{\mathsf{b}} \in D'_i \cap \overline{P_j}$ such that $\overline{\mathsf{a}} \in \overline{R}$ serves to inhibit the reaction r_j, with $I \cup \overline{R} \subseteq \hat{W}_{i-1}$. Therefore, we can find a reaction $(I \cup \overline{R}, \{\overline{\mathsf{b}}\})$ in $\mathcal{A}^+$ that is enabled at $\hat{W}_{i-1}$. In both cases, $\overline{\mathsf{a}} \in D'_{i-1}$ iff $\overline{\mathsf{a}} \in \hat{D}'_{i-1}$. □

Lemma 1 (Separation results)**.** *For every $\leq$-ordered pair relationship in Fig. 3, there exist a RS $\mathcal{A}$, a trace T and marking D_σ such that the relationship is strict.*

Proof. All strict relationships can be illustrated by means of our running example $\mathcal{A}_{\text{toy}}$ in this paper.

1) $s(\mu(\mathcal{A}), T_\mu, D_\sigma) < s(\mathcal{A}, T, D_\sigma)$. See Examples 8 and 9, by taking $\mathcal{A} = \mathcal{A}^+_{\text{toy}}$.

2) $s(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma) < s(\mathcal{A}^+, T^+, D_\sigma)$. See Examples 8 and 9, by taking $\mathcal{A} = \mathcal{A}_{\text{toy}}$.

3) $s(\mathcal{A}, T, D_\sigma) < s(\mathcal{A}^+, T^+, D_\sigma)$. See Examples 7 and 8, by taking $\mathcal{A} = \mathcal{A}_{\text{toy}}$.

4) $s(\mu(\mathcal{A}), T_\mu, D_\sigma) < s(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma)$. See Examples 7 and 9, by taking $\mathcal{A} = \mathcal{A}_{\text{toy}}$, since $\mu(\mathcal{A}_{\text{toy}}) = \mathcal{A}_{\text{toy}}$.

7) $ns(\mu(\mathcal{A}), T_\mu, D_\sigma) < ns(\mathcal{A}, T, D_\sigma)$. The RS $\mathcal{A}_{\text{toy}}$ of our running example is already minimized. However, if we add the redundant reaction $r_9 = (\{\mathsf{il6}, \mathsf{il21}\}, \varnothing, \{\mathsf{il6r}\})$ to $\mathcal{A}_{\text{toy}}$ we obtain a non minimized RS $\mathcal{A}'_{\text{toy}}$ such that $\mu(\mathcal{A}'_{\text{toy}}) = \mathcal{A}_{\text{toy}}$. Now, we can compare the slice obtained in Example 10 with the slice obtained by running $ns(\mathcal{A}'_{\text{toy}}, T, D_\sigma)$:

$$\frac{\{\overline{\mathsf{foxp3}}, \overline{\mathsf{il6}}, \overline{\mathsf{il21}}\}}{\{\mathsf{tcr}\}} \xrightarrow{\{6,9\}} \frac{\{\overline{\mathsf{il6r}}, \overline{\mathsf{tgfbr}}, \mathsf{nfat}\}}{\{\mathsf{il27}\}} \xrightarrow{\{7\}} \frac{\{\overline{\mathsf{foxp3}}, \overline{\mathsf{rorgt}}, \mathsf{stat1}\}}{\varnothing} \xrightarrow{\{8\}} \frac{\{\mathsf{tbet}\}}{\varnothing}$$

where it appears the extra negative dependency $\overline{\mathsf{il21}}$ in $\mathbf{D}_0$.

8) $ns(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma) < ns(\mathcal{A}^+, T^+, D_\sigma)$. See Examples 8 and 9, by taking $\mathcal{A}^+ = \mathcal{A}^+_{\text{toy}}$ and because the Dynamic Negative Trace Slicer coincides with the Trace Slicer for Positive RSs.

9) $ns(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma) < ns(\mu(\mathcal{A}), T_\mu, D_\sigma)$. See Examples 9 and 10, by taking $\mathcal{A} = \mathcal{A}_{\text{toy}}$ and because $\mu(\mathcal{A}_{\text{toy}}) = \mathcal{A}_{\text{toy}}$. □

Once we have established the relationship between the possible slicing, it remains open the question of which guarantees they can offer. Due to the non monotone behaviour of RSs, the best we can aim to is to derive some sufficient conditions that explain a certain phenomenon. In the following, we write $\mathbf{D} \models \mathbf{D}_i$ if $\mathbf{D}_i \subseteq \mathbf{D}$ and $D \models \mathbf{D}_i$ if $(D \cup \overline{(S_D \setminus D)}) \models \mathbf{D}_i$. Similarly for $\mathbf{C} \models \mathbf{C}_i$ and $C \models \mathbf{C}_i$.

Theorem 2 (Slices as sufficient conditions)**.** *Given a RS $\mathcal{A}$ one of its traces T and a set of marked entities D_σ, let $T' = \frac{\mathbf{D'_0}}{\mathbf{C'_0}} \xrightarrow{N'_1} \cdots \xrightarrow{N'_m} \frac{D_\sigma}{\varnothing}$ be the output of $ns(\mathcal{A}, T, D_\sigma)$. Then, for any initial set of entities D_0 and context sequence $C_0, ..., C_m$ such that $D_0 \models \mathbf{D'_0}$ and $C_i \models \mathbf{C'_i}$ for $i \in [0, m]$ we have that the entities D_σ will be produced after m transitions.*

Proof. We prove that whenever $D \models \mathbf{D'_i}$ and $C \models \mathbf{C'_i}$, then $res_A(D \cup C) \models \mathbf{D'_{i+1}}$ for any $i \in [0, m-1]$. Then the thesis follows from a simple inductive argument.

Let us take a generic $i \in [0, m-1]$ and assume that $\mathbf{D'_i} \subseteq (D \cup \overline{(S_D \setminus D)})$ and $\mathbf{C'_i} \subseteq (C \cup \overline{(S_C \setminus C)})$. We want to show that $\mathbf{D'_{i+1}} \subseteq (res_A(D \cup C) \cup \overline{(S_D \setminus res_A(D \cup C))})$. Let us take $\mathbf{a} \in \mathbf{D'_{i+1}}$. We consider separately the case of positive entities and negative ones.

- Let us take $\mathsf{a} \in \mathbf{D'_{i+1}}$ and prove that $\mathsf{a} \in res_A(D \cup C)$. Since $\mathsf{a} \in \mathbf{D'_{i+1}}$ it must have been produced by some reaction $r_j = (R_j, I_j, P_j)$ whose causes must have been recorded in $\mathbf{D'_i}$ and $\mathbf{C'_i}$. In particular, $(R_j \cap S_D) \cup \overline{(I_j \cap S_D)} \subseteq \mathbf{D'_i}$ and $(R_j \cap S_C) \cup \overline{(I_j \cap S_C)} \subseteq \mathbf{C'_i}$. Therefore $R_j \cap S_D \subseteq D$, $I_j \cap S_D \subseteq S_D \setminus D$, $R_j \cap S_C \subseteq C$, and $I_j \cap S_C \subseteq S_C \setminus C$, from which it follows that $R_j \subseteq D \cup C$ and $I_j \subseteq (S \setminus (D \cup C))$. Therefore $en_{r_j}(D \cup C)$ and $\mathsf{a} \in res_A(D \cup C)$.
- Let us take $\overline{\mathsf{a}} \in \mathbf{D'_{i+1}}$ and prove that $\overline{\mathsf{a}} \notin res_A(D \cup C)$. Since $\overline{\mathsf{a}} \in \mathbf{D'_{i+1}}$ there must be some prohibiting set $I \cup \overline{R}$ that prevents the production of a and that has been recorded in $\mathbf{D'_i}$ and $\mathbf{C'_i}$. In particular, $(I \cap S_D) \cup \overline{(R \cap S_D)} \subseteq \mathbf{D'_i}$ and $(I \cap S_C) \cup \overline{(R \cap S_C)} \subseteq \mathbf{C'_i}$. Therefore $I \cap S_D \subseteq D$, $R \cap S_D \subseteq S_D \setminus D$, $I \cap S_C \subseteq C$, and $R \cap S_C \subseteq S_C \setminus C$, from which it follows that $I \subseteq D \cup C$ and $R \subseteq (S \setminus (D \cup C))$. Therefore any reaction r_j having a as a product is inhibited by the prohibiting set $I \cup \overline{R}$ (i.e., it is not the case that $en_{r_j}(D \cup C)$) and thus $\mathsf{a} \notin res_A(D \cup C)$. □

The above result shows that the initial set $\mathbf{D'_0}$ and the context sequence $\mathbf{C'_0}, ..., \mathbf{C'_m}$ express sufficient conditions for the production of the marked entities D_σ after m transitions.

It can be shown that this is not the case for the ordinary slicing, where only positive entities are

recorded, unless Positive RSs are considered. A simple example demonstrating that positive entities do not represent a sufficient condition, is a RS consisting of a single reaction with at least one inhibitor and a single product. Recoding as causes for the production of the product only the positive entities (i.e., the reactants) does not provide a sufficient condition. The presence of the reactants is indeed not enough to guarantee the production of the product. It is also necessary to ensure that inhibitors are not present. Instead, collecting both positive entities (for reactants) and negative ones (for inhibitors) provide information that expresses a sufficient condition. In the T cell case study we will see that the collection of positive entities produce a very limited amount of information compared to what is obtained by taking also negative entities into account (see Fig. 8).

The next result allows us to conclude that the slicing on the minimized positive RS model $\mu(\mathcal{A}^+)$ is in general the most convenient, among those available. However, it is worth noting that, in general, the slicing does not produce necessary conditions, but only sufficient ones. This is because the dependencies identified by the slicing mix and/or conditions (the same product can be produced by many reactions in a single step).

Corollary 1 (Best slicing)**.** *Given a trace T in a RS $\mathcal{A}$, the less restrictive sufficient conditions for the production of D_σ are obtained by running $ns(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma)$, or equivalently $s(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma)$.*

Proof. By Theorem 2 we know that the Dynamic Negative Trace Slicer always returns sufficient conditions. By Theorem 1 we know that $ns(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma)$ returns the less restrictive answer and that it coincides with the one obtained by running $s(\mu(\mathcal{A}^+), T^+_\mu, D_\sigma)$. □

6 RS Model of T Cell Differentiation

In this section we define the RS model of T cell differentiation on which we demonstrate our attractor and slicing analyses. The model is based on the Boolean network model proposed in [13], and its translation into RSs was already given in [7]. Here, we consider also its translation into Positive RSs, and the minimized versions of both the RS and the Positive RS models.

6.1 The Boolean network model

T cell differentiation is a widely studied biological phenomenon for which several Boolean network models have been proposed [13, 26, 27]. We considered the model investigated in [13] and available on CellCollective [28]. The analysis performed in [13] is based on the simulation method provided by such a platform.

We choose this model since it describes a realistic regulation system that is involved in many diseases [29–31] and it has been investigated under both the viewpoints of causality (effect of environmental conditions) and of reachability (reachable phenotypes). The Boolean network model is graphically represented as shown in Fig. 4 and is fully specified by the Boolean update formulas shown in Fig. 5. It includes 9 *input* nodes (orange), that correspond to stimuli that the T cell can receive by the external environment. Then, it contains 23 nodes representing relevant T cell genes, and 6 nodes representing secreted cytokines.

T cells can differentiate by expressing four different phenotypes, denoted Th1, Th2, Th17 and iTreg. Such phenotypes correspond to the expression of four different transcription factors included in the model: Tbet (for Th1), GATA3 (for Th2), RORgt (for Th17) and Foxp3 (for iTreg). There exists experimental evidence that a T cell can express more than one phenotype (hence, more than one of the associated transcription factors) [32]. Moreover, through the computational analysis performed in [13], the authors hypothesize that T cells may express also other combinations of phenotypes, that might include three or even four of them. Finally, the case in which none of the four relevant transcription factors is expressed (i.e., no phenotype) is denoted as Th0.

6.2 The RS and Positive RS Models

Translating a Boolean network model into an equivalent RS can be done by turning every Boolean formula of the network into disjunctive normal form. Then, every term of the disjunction (which is a conjunction of atoms, possibly negated) is translated into a reaction in which (i) reactants are the positive atoms, (ii) inhibitors are the negated atoms and (iii) the (only) product is the Boolean variable that is updated through the considered formula. For example, the Boolean formula for IL12R (also present in Fig. 5, in blue), namely:

```
IL12R = (IL12 and NFAT) or (STAT4 and not GATA3)
        or Tbet or (TCR and not GATA3)
```

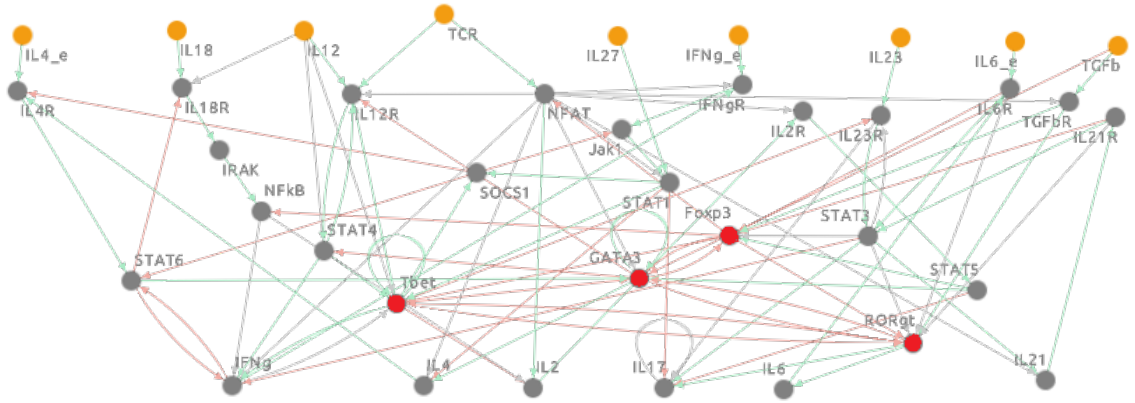


Fig. 4: Graphical representation of the Boolean network model of T cell differentiation from [13]. The model is available in electronic format at [10].

is translated into the following four reactions:

$$(\{\texttt{IL12}, \texttt{NFAT}\}, \varnothing, \{\texttt{IL12R}\})$$
$$(\{\texttt{STAT4}\}, \{\texttt{GATA3}\}, \{\texttt{IL12R}\})$$
$$(\{\texttt{Tbet}\}, \varnothing, \{\texttt{IL12R}\})$$
$$(\{\texttt{TCR}\}, \{\texttt{GATA3}\}, \{\texttt{IL12R}\})$$

To run our analyses of the T cell differentiation model, we will exploit BioReSolve [9], a tool implementing the transformation of Boolean network specifications into RSs models, the SOS semantics of RSs, a minimization algorithm, the translation into Positive RSs and the slicing analysis methods described in the previous sections. It has been developed in SWI-Prolog [33] by some of the authors of this paper, according to the specifications given in [34] and [6]. The RS specification can be generated from the Boolean network model by invoking the `main_do(bn2rs)` directive of BioReSolve. In BioReSolve syntax, the four reactions above are coded as follows (also in Fig. 6, in blue):

```
react([il12,nfat],[],[il12r]),
react([stat4],[gata3],[il12r]),
react([tbet],[],[il12r]),
react([tcr],[gata3],[il12r])
```

Even more complex forms of Boolean networks (such as Threshold Boolean Networks [35]) can be translated into RSs, as we demonstrated in [36].

Reactions of the RS model of T cell differentiation obtained from the translation of the Boolean network are given in Fig. 6. The obtained model $\mathcal{A}_{\text{Tcell}}$ consists of nine contextual entities (corresponding to the nine orange nodes in Fig. 4), 29 ordinary entities (corresponding to the remaining nodes in Fig. 4) and 51 reactions, among which we extracted the eight reactions of our running example $\mathcal{A}_{\text{toy}}$. Moreover, by applying the minimization feature of BioReSolve (by invoking the `main_do(minimize)` directive), we obtain the RS model $\mu(\mathcal{A}_{\text{Tcell}})$ consisting of 49 reactions (shown in Fig. A3 in the Appendix). Indeed, minimization causes the removal of reaction `react([nfat,nfkb,stat4],[stat3,stat6],[ifng])` (made redundant by the presence of reaction `react([nfkb],[],[ifng])`) and the removal of reaction `react([il23,stat3],[tbet],[il23r])` (made redundant by the presence of reaction `react([stat3],[],[il23r])`). We remark that these redundancies are not due to an inaccurate translation of the original Boolean network into RSs, but were already present in the Boolean network. For example, by looking at the Boolean formula for `IL23R` in Fig. 5, it is clear that the conjunction `(IL23 and STAT3 and not Tbet)` is redundant in the formula.

The RS model $\mathcal{A}_{\text{Tcell}}$ has then been used also to generate Positive RS specifications. BioReSolve implements the encoding into Positive RS given in Definition 7 by invoking the `main_do(sppos)` directive. All positive reactions for the model $\mathcal{A}^{+}_{\text{Tcell}}$, obtained by applying such an encoding, are shown in Fig. A4 in the Appendix. Negative entities considered in Positive RSs are represented in BioReSolve as entities with a name starting with the `neg_` prefix. For example, the negative entity $\overline{\texttt{tbet}}$ is represented by the term `neg_tbet`. Moreover, the minimized version $\mu(\mathcal{A}^{+}_{\text{Tcell}})$ of those positive reactions (which can be obtained either by invoking `main_do(minimize)` on the Positive RS specification $\mathcal{A}^{+}_{\text{Tcell}}$, or by invoking directly `main_do(minsppos)` on the original RS specification $\mathcal{A}_{\text{Tcell}}$) are shown in Fig. A5 in the Appendix. The non minimized Positive RS specification $\mathcal{A}^{+}_{\text{Tcell}}$ turns out to include 275 reactions, while its minimization $\mu(\mathcal{A}^{+}_{\text{Tcell}})$ only 128.

```
 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 not Tbet and
        ((STAT3 and IL21R) or (STAT3 and IL6R))
        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. 5: Update rules of the Boolean network model of T cell differentiation from [13]. Available in electronic format at [10]. As an example, the processing of update rule for `IL12R` (in blue) is presented in Section 6.2.

.

6.3 Implementation of the Analyzer

To apply our method in the analysis of gene regulatory networks, we interface with BioReSolve through a Python script. This script makes it possible (i) to exploit Python packages such as `networkx` to process the LTS generated by BioReSolve for identifying attractors, and (ii) to automate the execution of the slicing analysis method for multiple gene targets and LTS states. This interaction is made possibile by the usage of the `swiplserver` Python package, allowing to run SWI-Prolog code via Python function invocations.

More precisely, the Python script performs the following steps:

1. it assumes that the LTS generated by BioReSolve is already available in dot format, and uses it to create a `networkx DiGraph` object;
2. it initializes the list of targets of the analysis as a list of objects of a class `Target` with two fields, `present` and `absent`, denoting the genes that have to be expressed and non expressed, respectively, in the states of attractors that have to be considered as target;
3. it defines a few functions that will be used during the attractor analysis:
   - function `check_node(node)`: checks whether the LTS state `node` belongs to an attractor of the LTS (i.e., it is in a cycle);
   - function `compute_attractor(node)`: returns the list of genes that are present in states of the attractor reachable from state `node`[3];
   - function `target_computations(target)`: selects the LTS traces that lead to attractors in target `target`, and for each of them it provides a description of the corresponding context sequences;
4. it executes the main loop that, for every target to be investigated, invokes the `target_computation` function and then for every state of every attractor of such a target, asks BioReSolve to compute the sliced computation leading to that state. Entities mentioned in a sliced computation are then collected in a set representing the entities that actually played a role for the presence of the target genes in the attractor. Finally, the set of entities obtained for each target are used to compute two results: the union of all of them (representing those entities that *may* play a role to achieve the target) and the intersection of all of them (representing those entities that *must* play a role).

The Python script we developed, along with the BioReSolve specification of the RS model to be analysed in this paper, are freely available at [37].

7 Attractors and Slicing Analyses

In this section we describe the analyses we performed on the RS model of T cell differentiation.

[3] We will show that the LTS for the analysis described in this paper are such that every state, apart from the initial one, leads to exactly one attractor.

```
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([tbet],[],[il12r]),
react([tcr],[gata3],[il12r]),
react([stat4],[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. 6: The 51 reactions of the RS model $\mathcal{A}_{\text{Tcell}}$ in BioReSolve syntax. As an example, as explained in Section 6.2, the reactions encoding the update rule for `IL12R` are colored in blue.

This case study will allow us to illustrate which kind of information we can extract from the model by using attractors and slicing analyses, and also to compare the results we can obtain from the different notions of slicing we proposed in the previous sections. We start from the generation of the LTSs obtained by the execution of BioReSolve. Then, we perform an analysis aimed at identifying attractors and characterizing environmental configurations (i.e., RS contexts) leading to them. Finally, we perform the slicing analysis in order to identify relevant and crucial internal regulators for the expression of the different phenotypes.

7.1 Generation of the LTSs

The LTS of the RS model is generated by invoking the `main_do(digraph)` directive of BioReSolve after providing model and context specifications in the expected syntax. In order to test all of the possibile configurations of the environment, the RS context in the BioReSolve specification is defined as a process that, at the first step, chooses in a non-deterministic way the set of environmental stimuli that have to be present. Then, the chosen stimuli will persist in the context.

7.1.1 LTSs of the RS models

We start by generating the LTSs for the models of T cell differentiation expressed as standard RSs, namely the original RS model $\mathcal{A}_{\text{Tcell}}$, whose reactions are listed in Fig. 6, and its minimized variant $\mu(\mathcal{A}_{\text{Tcell}})$ consisting of all reactions listed in Fig. A3. In these cases, according to the syntax of RS processes given in Definition 1, the context providing one of all possibile combinations of environmental stimuli is expressed by the following parallel composition of processes:

$$\begin{aligned}&\mathsf{rec\ x11.}\{\mathtt{TGFb}\}\mathsf{.x11} + \mathsf{rec\ x0.}\varnothing\mathsf{.x0} \mid \ldots\\ &\qquad\qquad \ldots \mid \mathsf{rec\ x91.}\{\mathtt{TCR}\}\mathsf{.x91} + \mathsf{rec\ x0.}\varnothing\mathsf{.x0}\end{aligned}$$

which, in the BioReSolve model specification, is expressed as follows:

```
myenvironment('[
    x1 = ({tgfb}.x11 + {}.x0),
    x2 = ({il23}.x21 + {}.x0),
    x3 = ({il12}.x31 + {}.x0),
    x4 = ({il18}.x41 + {}.x0),
    x5 = ({il4e}.x51 + {}.x0),
    x6 = ({il27}.x61 + {}.x0),
    x7 = ({il6e}.x71 + {}.x0),
    x8 = ({ifnge}.x81 + {}.x0),
    x9 = ({tcr}.x91 + {}.x0),
    x11 = {tgfb}.x11, x21 = {il23}.x21,
    x31 = {il12}.x31, x41 = {il18}.x41,
    x51 = {il4e}.x51, x61 = {il27}.x61,
    x71 = {il6e}.x71, x81 = {ifnge}.x81,
    x91 = {tcr}.x91, x0 = {}.x0 ]').
mycontext("[x1,x2,x3,x4,x5,x6,x7,x8,x9]").
```

Since we have 9 input entities representing the possible stimuli, this definition of the context process leads to an LTS with an initial branching into 2^9 different states, each being the start of a deterministic computation (i.e., a linear trace).

7.1.2 LTSs of the Positive RS models

The LTSs of the Positive RS models, namely $\mathcal{A}^{+}_{\text{Tcell}}$ consisting of reactions listed in Fig. A4 and its minimized variant $\mu(\mathcal{A}^{+}_{\text{Tcell}})$ consisting of reactions listed in Fig. A5, can be generated as those of ordinary RS models, but with a different specification of the context processes. In this case, for every environmental stimulus the context has always to provide either its positive or its negative representation. As a consequence, the context process specification becomes:

$$\begin{aligned}&\mathsf{rec\ x11.\{TGFb\}.x11} + \mathsf{rec\ x12.\{\overline{TGFb}\}.x12} \mid \ldots\\ &\qquad \ldots \mid \mathsf{rec\ x91.\{TCR\}.x91} + \mathsf{rec\ x92.\{\overline{TCR}\}.x92}\end{aligned}$$

which, in the BioReSolve model specification, is expressed as follows:

```
myenvironment('[
    x1 = ({tgfb}.x11 + {neg_tgfb}.x12),
    x2 = ({il23}.x21 + {neg_il23}.x22),
    x3 = ({il12}.x31 + {neg_il12}.x32),
    x4 = ({il18}.x41 + {neg_il18}.x42),
    x5 = ({il4e}.x51 + {neg_il4e}.x52),
    x6 = ({il27}.x61 + {neg_il27}.x62),
    x7 = ({il6e}.x71 + {neg_il6e}.x72),
    x8 = ({ifnge}.x81 + {neg_ifnge}.x82),
    x9 = ({tcr}.x91 + {neg_tcr}.x92),
    x11 = {tgfb}.x11, x21 = {il23}.x21,
    x31 = {il12}.x31, x41 = {il18}.x41,
    x51 = {il4e}.x51, x61 = {il27}.x61,
    x71 = {il6e}.x71, x81 = {ifnge}.x81,
    x91 = {tcr}.x91,
    x12 = {neg_tgfb}.x12, x22 = {neg_il23}.x22,
    x32 = {neg_il12}.x32, x42 = {neg_il18}.x42,
    x52 = {neg_il4e}.x52, x62 = {neg_il27}.x62,
    x72 = {neg_il6e}.x72, x82 = {neg_ifnge}.x82,
    x92 = {neg_tcr}.x92, x0 = {}.x0 ]').
mycontext("[x1,x2,x3,x4,x5,x6,x7,x8,x9]").
```

We remark that all the RS models and Positive RS models produce homomorphic LTSs (all of them include 2.529 states and 3.040 transitions). Indeed, all of these specifications are logically equivalent.

7.2 Attractors analysis

BioReSolve returns the LTS as a graph in dot format, which is then loaded by our Python script. Then, attractors related with a target of interest are identified by looking for cycles in the LTS. Due to the LTS structure, this is particularly simple, and it is done in the `target_computations` function defined in our Python script by invoking repeatedly the `find_cycle` function provided by `networkx` package.

The analysis is performed by considering different *targets* corresponding to the different combinations of phenotypes that a T cell can express. As we described in Section 6.1, T cells can exhibit four phenotypes, Th1, Th2, Th17 and iTreg, represented by the expression of the four transcription factors Tbet, GATA3, RORgt and Foxp3, respectively. Since we are interested also in combinations of phenotypes, we consider all of the 2^4 combinations of such transcription factors: each of them will be a target of our analysis (hence, 16 targets overall).

For each target, we identify the attractors that include states in which the target transcription factors are expressed (not all necessarily in the same state), and in which the other transcription factors are not expressed. For example, for the target containing the combination of transcription factors {Tbet,GATA3}, we select the attractors (i.e., the cycles) that include 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 filtering procedure just described gives as a result the subset of the attractors that correspond to the achievement of the considered target. Each of these attractors is reachable by making a different choice of environmental stimuli in the first step. The collection of all of those choices allows us to determine, for each target, which combinations of stimuli leading to it.

In Table 2 we report the number of different contexts (choices of environmental stimuli) that lead to each target. The table shows only targets that can be reached, and they result to be those in which only one transcription factor is expressed, and those in which Tbet is expressed together with another transcription factor. For each target, the contexts leading to it are summarized in the table by a Boolean formula, while in Fig. 7 they are depicted graphically.

Analysis results show that 248 contexts (out of 512) lead to the expression of at least one of the four transcription factors under study. Tbet, corresponding to phenotype Th1, is expressed in the majority of the cases (216 out of 248), half of the times in combination with another transcription factor (in 108 cases out of 216). Also Foxp3 and RORgt are often expressed (in 64 and 56 cases out of 248, respectively, mostly in combination with Tbet), while GATA3 is less frequent (12 cases only).

Fig. 7 allows us to make the following main observations about the role of the environment in the T cell differentiation process:

Target	Contexts	Formula
Tbet	108	(not tgfb) and (tcr) and ((il12) or ((not il4e) and (not ifnge)) or (il27))
GATA3	8	(not tgfb) and (not il12) and (il4e) and (not il27) and (not ifnge) and (tcr)
Foxp3	8	(tgfb) and (not il12) and (not il27) and (not il6e) and (not ifnge) and (tcr) or (il12))
RORgt	16	(tgfb) and (not il12) and (not il27) and (il6e) and (tcr)
Tbet,GATA3	4	(not tgfb) and (not il12) and (il4e) and (not il27) and (not il6e) and (ifnge) and (tcr)
Tbet,Foxp3	56	(tgfb) and (tcr) and (not il6e) and ((il12) or (il27) or (ifnge))
Tbet,RORgt	48	(tgfb) and (il6e) and (tcr) and ((il12) or (il27))
TOTAL	**248**	

Table 2: Summary of contexts leading to each reachable target.

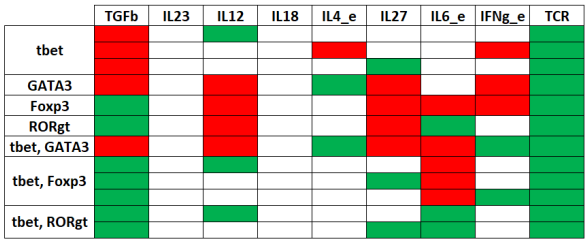


Fig. 7: Graphical representation of the Boolean formula given in Table 2 and characterizing contexts leading to each reachable target. Green cells denote that the corresponding environmental stimulus has to be present, red cells denote absences, and white cells denote that the stimulus is irrelevant.

- TCR has to be present in order to express any of the phenotypes
- IL23 and IL18 do not contribute to determine any of the phenotypes
- The presence of TGFb mostly discriminates between the expression of either Foxp3/RORgt or Tbet/GATA3.

Moreover, the detailed characterization of the contexts leading to the expression of each phenotype could allow several more specific observations to be done.

The results of the analysis we conducted partially agree with those of a similar analysis done in [13]. However, there are also some significant differences between the results of the two studies. First of all, in [13] the authors show that it is also possibile to reach configurations in which three or four (i.e., all) phenotypes are expressed. Moreover, they show that a configuration in which only RORgt is expressed cannot be reached. The Boolean network we started from is exactly the same as the one considered in [13]. The main difference between the two analysis approaches lies in the semantics of the environment: in our case the environment is modelled by a context process that remains the same after the initial non-deterministic choice, while in [13] the analysis is performed by applying a simulation method that allows activity levels and noise to be taken into account for the input species. Hence, the method applied in [13] can lead to a larger range of hypotheses about the modelled system behaviours (such as the possibility for T cell to express more than two phenotypes), while ours is more conservative and consistent with the standard simulation approaches.

As a final remark, we point out that the attractor analysis, that we conducted on the LTS of the non-minimized RS model, does not need to be repeated on the other LTSs we constructed since on the basis of the logical equivalences of the specifications, the results would be exactly the same.

7.3 Slicing analyses

In this section, we execute the different variants of slicing analysis we proposed in Sections 4 and 5 to the RS and Positive RS models we constructed for the T cell differentiation case study.

For each target, slicing analysis is conducted by invoking either the `main_do(ordslice)` directive of BioReSolve (to execute the ordinary slicing algorithm, as in Sections 4, 4.1 and 4.2) or the `main_do(negslice)` directive (to execute the dynamic negative slicing proposed in Section 5) for each state of each attractor of the target. This is obtained by implementing suitable nested loops in the Python script that, at each iteration, execute BioReSolve through the Python-to-Prolog binding provided by the `swiplserver` package.

Slicing analysis in BioReSolve requires the specification of a proper context sequence and of a few predicates necessary to characterize the trace on

which slicing has to be applied. This corresponds to providing a trace T and a marking D_σ to the slicing algorithms as described in Section 5.1. For our analysis, given a state of an attractor of interest, the context sequence provides constantly the entities that led to such an attractor, as identified during attractors analysis in Section 7.2. The additional predicates, instead, are necessary in order to characterize the state in which the trace has to be interrupted (since we reached the state of interest) and the entities in such a state that have to be marked. The state where to interrupt the trace can be expressed through the predicate `mytarget([...])`, while the entities to be marked through the predicates `mypos([...])` and `myneg([...])`. In the analysis of our case study, for each target we are going to mark the entities representing the corresponding combination of Tbet, GATA3, Foxp3 and/or RORgt. This can be done by listing those entities in the `mypos([...])` predicate. In principle, the tool would allow us also to mark entities that are absent in the state of interest and for which we would like the slicing algorithm to identify causes, but this will not be done in this case study. So, we will always provide an empty list to the corresponding predicate `myneg([...])`.

Here, we show an example of specification of both the context and the additional predicates for the slicing analysis to be applied on the RS model of our case study. The considered target is {`Tbet`, `RORgt`}, which is reachable when the context only provides `IL12`, `IL4_e` and `INFg_e` (denoted as `x31` `x51` and `x81`, respectively, as in Section 7.1.1):

```
mycontext("[x31,x51,x81]").
mytarget([il12r,il21,il21r,il23r,il6r,nfat,
          rorgt,socs1,stat1,stat3,tbet,tgfbr]).
mypos([tbet,rorgt]).
myneg([]).
```

In the case of a Positive RS specification, the context has to provide also negative entities (when absent). This is done by adding context processes `x12`, `x22`, `x42`, `x62`, `x72` and `x92`, as defined in Section 7.1.2. Hence, for the same example, we have:

```
mycontext("[x31,x51,x81,
            x12,x22,x42,x62,x72,x92]").
mytarget([il12r,il21,il21r,il23r,il6r,nfat,
          rorgt,socs1,stat1,stat3,tbet,tgfbr
          neg_foxp3,neg_gata3,neg_ifng,neg_ifngr,
          neg_il17,neg_il18r,neg_il2,neg_il2r,
          neg_il4,neg_il4r,neg_il6,neg_irak,
          neg_jak1,neg_nfkb,neg_stat4,neg_stat5,
          neg_stat6]).
mypos([tbet,rorgt]).
myneg([]).
```

Note the presence of negative entities in the target state: they must be made explicit because the tool searches for the exact state specified by the `mytarget([...])` predicate.

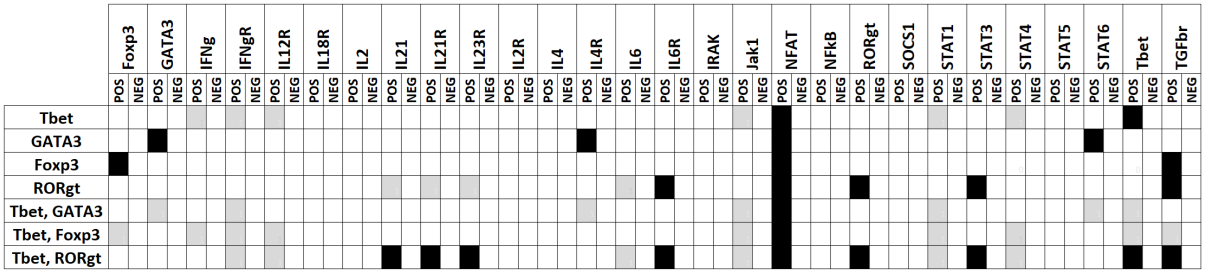
We start by applying the original slicing approach proposed in Section 4 to the RS model $\mathcal{A}_{\text{Tcell}}$ obtained from the translation of the Boolean network as described in Section 6.2 (both non-minimized and minimized). Then, we consider the Positive RS models $\mathcal{A}^+_{\text{Tcell}}$ described in the same section, and we discuss the differences arising from applying the slicing analysis to the minimized variant $\mu(\mathcal{A}^+_{\text{Tcell}})$ rather than to the non-minimized one $\mathcal{A}^+_{\text{Tcell}}$. Finally, we present the application of the dynamic negative slicing method proposed in Section 5 to the non-minimized RS model $\mathcal{A}_{\text{Tcell}}$. All results we will obtain from the different slicing analyses will relate each other in agreement with the relationships we proved in Theorem 1. We conclude this section with a number of observations on the slicing results, to illustrate on this case study which kinds of information and new knowledge it is possibile to extract from these experiments.

7.3.1 Slicing analysis on RS models

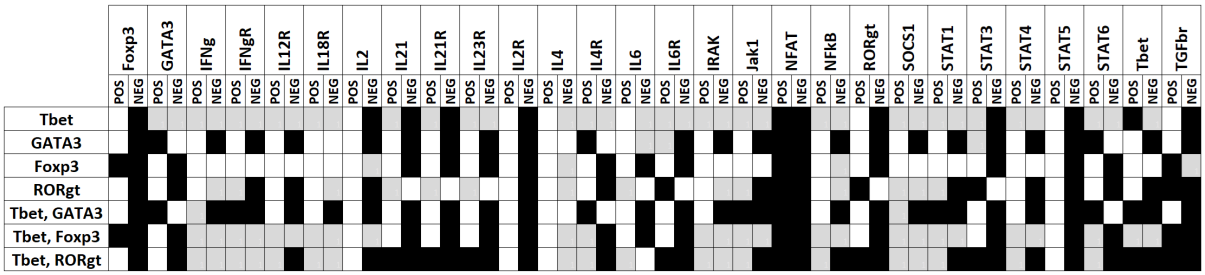
Results of slicing analysis (the ordinary one) on the (non Positive) RS models $\mathcal{A}_{\text{Tcell}}$ of the T cell differentiation network are summarized in Fig. 8a, where it is possibile to see, for each target, which internal genes are *strictly necessary* (black) and which are somehow *relevant* (grey) for the expression of the transcription factor of the target. We remark that this is a form of causality analysis: for example, one gene that results to be necessary may cause the execution of a chain of reactions leading after some steps to the expression of one of the target transcription factors.

Results reported in Fig. 8a are the same for both the cases of the non-minimized model $\mathcal{A}_{\text{Tcell}}$ and its minimization $\mu(\mathcal{A}_{\text{Tcell}})$. The two small redundancies identified in the non-minimized model (see Section 6.2) did not introduce any additional (and fictitious) dependency. As expected, only positive dependencies are identified.

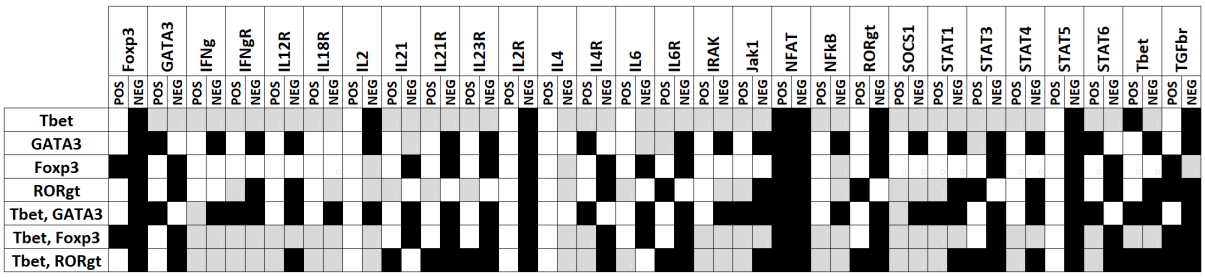
From these results we could deduce that, for instance, the activation of NFAT is strictly necessary-for the achievement of all of the target. Moreover, the activation of IL6 results to be necessary for achieving target GATA3, and so on.



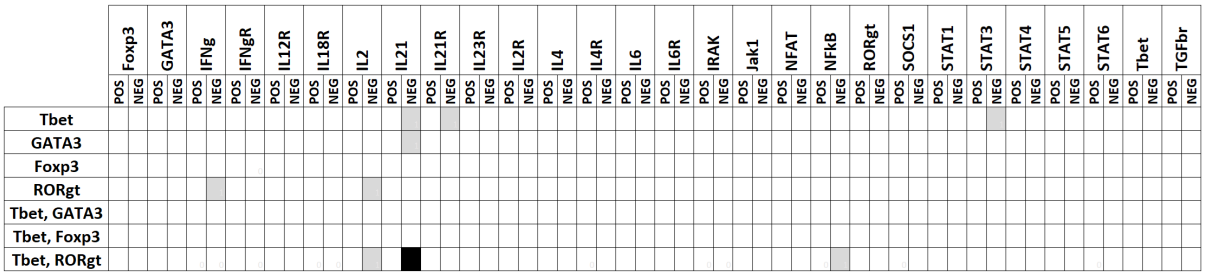
(a) Original RS model $\mathcal{A}_{\text{Tcell}}$ and minimized RS model $\mu(\mathcal{A}_{\text{Tcell}})$



(b) Non-minimized Positive RS model $\mathcal{A}^{+}_{\text{Tcell}}$ and original RS model $\mathcal{A}_{\text{Tcell}}$ with dynamic negative slicing



(c) Minimized Positive RS model $\mu(\mathcal{A}^{+}_{\text{Tcell}})$



(d) $\mathcal{A}^{+}_{\text{Tcell}}$ vs $\mu(\mathcal{A}^{+}_{\text{Tcell}})$: Difference between non-minimized and minimized Positive RS models

Fig. 8: Results of slicing analysis. In tables (a), (b) and (c) black cells identify genes (in columns) that are strictly necessary for the achievement of each target (in rows). Gray cells identify relevant genes. White cells identify genes that are not relevant. Table (d) shows the differences between (b) and (c).

7.3.2 Slicing analysis on Positive RS models

By applying slicing analysis (again, the ordinary one) to the Positive RS models, a lot of additional dependencies are identified. They are shown in Fig. 8b for the non-minimized Positive RS model $\mathcal{A}^{+}_{\text{Tcell}}$, and in Fig. 8c for the minimized one $\mu(\mathcal{A}^{+}_{\text{Tcell}})$. A number of negative dependencies are now identified. For example, IL2R turns out to play an inhibitor role for all of the T cell differentiation targets. However,

a lot of additional positive dependencies are identified compared to those already present in Fig. 8a. These represent entities that have to be present in order to avoid entities involved in negative dependencies to be produced. As a consequence, these are positive dependencies that could not be identified by analyzing positive dependencies only.

Another interesting observation on these new slicing results is that there exist many entities for which the analysis identified both positive and negative dependencies. This happens for instance for NFAT, which results to be both a promoter and an inhibitor for the achievement of all of the targets. This result could seem contradictory, but it corresponds to the situation in which the same entity is involved in at least two different reactions, once as a reactant and once as an inhibitor, and both reactions have to take place in order for the target to be achieved. Of course, this has to happen at two different times, namely in two different steps of the trace leading to the target.

The analysis of the non-minimized and of the minimized Positive RS models provide very similar results, that are however not exactly the same. The difference between the two is emphasized in Fig. 8d in which dependencies identified on the non-minimized model and missing in the minimized model are depicted. The existence of such a difference agrees with the relationship we proved in Theorem 1 and the consequence is that the analysis of the non-minimized model produced a slight over-approximation of the dependencies compared to those identified on the minimized model.

7.3.3 Dynamic negative slicing analysis on non-minimized RS model

By applying the dynamic negative slicing analysis to the non-minimized RS model of T cell differentiation, we obtain exactly the same results we obtained by applying the ordinary slicing algorithm on the non minimized Positive RS model (see Fig. 8b). Again, this agrees with the relationships stated in Theorem 1.

Running the analysis on the RS model allows us to skip the translation into Positive RS, which could require to generate an exponential number of new reactions. On the other hand, the analysis itself is more time consuming since, at each step, the algorithm has potentially to consider all of the reactions in order to investigate causes for non application. The choice between the two methods depends also on the length of the trace to be investigated, on how many traces the analysis have to be analyzed, and includes also a memory space vs computational time trade off.

7.3.4 Observations on slicing results

In order to make some final considerations about the analysis results, let us focus on those obtained from the application of ordinary slicing to the minimal Positive RS model, that are the most accurate results. The high density of grey and black cells in Fig. 8c means that all of the entities included in the model are (positively and/or negatively) relevant for the achievement of some target. This is not surprising, since we started our analysis from a Boolean network model specifically designed to model the process for the activation of those targets, and missing dependencies from one entity would mean that the corresponding gene would be included in the Boolean network by mistake, or by assuming a role for it that was not actually relevant.

An important information that can be extracted from the slicing results is about the role each gene has for different targets. Identifying genes that are highly specific for the achievement of a target of interest is particularly valuable, since those genes could be considered as potential drug targets if the target is related to a disease. For example, IL4R activation turns out to be crucial to activate target GATA3, irrelevant to achieve RORgt and partially relevant to achieve the other targets. If GATA3 would be associated to some disease, it could be interesting to investigate the development of a drug inhibiting IL4R activation.

The choice of which dependencies to choose for the development of a drug or, more generally, to further investigate the network and the biological process it describes, could be supported by combining our slicing analysis with some analysis for node importance estimation, which do not provide a causal information, but could allow to prioritize genes on the basis of how a perturbation on their behavior impact on the overall network dynamics [38, 39].

8 Conclusions and future work

In this paper we have presented a framework for analysing RSs using two slicing algorithms. We considered a previously defined algorithm for detecting dependencies between entities in a computation and we defined a new Dynamic Negative Slicing algorithm for detecting negative dependencies, arising from the inhibitors in the reactions. We considered a transformation on RSs for computing statically Positive

RSs, which represent negative dependencies by special entities, and which add to a RS several new reactions, which can be statically minimized. We discussed formally the relation between the two slicing algorithms by showing a diagram which compares the two algorithms when applied to standard RS, or to the positive version of a RS, with or without a minimization of the rules of the RS. The implementation in BioReSolve [9] now includes the two slicing techniques and the experiments in [7] have been now performed by using the two slicing algorithms. The experiments confirm that the Dynamic Negative Slicing on a RS is more efficient than the other slicing algorithm applied to the positive version of the RS, as the transformation to the positive version is not necessary. We also show that the minimization on the reactions allows to obtain a more precise analysis of the dependencies.

Our methodology allows us to analyze the attractors of a RS model and identify the biological entities that are responsible for their achievement. Our framework considers an operational semantics of RSs which allows us to construct Labeled Transition Systems (LTSs) to describe the behaviours of the model. Then, we apply our slicing algorithms to identify the minimal entities which describe a behaviour that we analyse.

Our methodology can be applied to Boolean network models of gene regulation, by translating them into RSs. We remark that such a translation emphasizes the different causes for the activation of each gene (representing them as different reactions). Moreover, it enables the application of analysis tools already developed for RSs.

We have applied our slicing analyses for the negative and positive dependencies to a model of T cell differentiation in the immune system. The analysis of the model allowed us to determine which combinations of stimuli lead to each phenotype, and also which proteins inside T cells are involved in each case. This analysis contributes to the understanding of the behaviour of the model, to correct it in case some mistakes are identified and to use it for identifying drug targets. We discussed the relation with other approaches in the literature.

Our methodology is general and can be applied to the (large number of) case studies in the public database on the CellCollective platform [28], as well as to other Boolean network models.

As a future work we plan to extend our methodology in several ways, for instance by deriving information on the number of paths in which important molecules for a given target are used, by considering subsets of molecules and by counting the paths in which a (set of) molecule(s) is present. Our method might also be combined with other analyses (e.g. [5, 40–42]) which are based on quantitative extensions of RSs, in order to define models which can express directly properties which require to count the number of entities in the system, or to express the time and speed of a reaction.

Declarations

Ethical Approval
This is not applicable.

Competing interests
The authors have no relevant financial or non-financial interests to disclose.

Authors' contributions
The authors of this paper have contributed equally.

Funding
This research has been partially supported by the Italian MUR PRIN 2022 project "MEDICA" (2022RNTYWZ, CUP_B53D23013170006), by the Next Generation EU programme project PNRR ECS00000017 - "THE - Tuscany Health Ecosystem" - Spoke 3 - CUP B63C22000680007, and Spoke 6 - CUP I53C22000780001, by the Italian MUR PRIN PNRR 2022 project "DELICE" *Decentralized Ledgers in Circular Economy* (P20223T2MF), by the Italian MUR PRIN 2022 PNRR project *Resource Awareness in Programming: Algebra, Rewriting, and Analysis* (P2022HXNSC), by the Next Generation EU programme project PNRR "SEcurity and RIghts In the CyberSpace - SERICS" (PE00000014 - CUP H73C2200089001), and by the INdAM–GNCS projects CUP E53C22001930001 and CUP E53C24001950001.

Availability of data and materials
Data Availability Statement: No Data associated in the manuscript.

Code availability
The source code of all software tools and scripts described in this paper, as well as the BioReSolve specifications of the T Cell differentiation models presented as case study are freely available at https://github.com/Unipisa/TCell-AttractorsSlicingAnalyses/.

References

[1] Ehrenfeucht, A., Rozenberg, G.: Reaction systems. Fundamenta Informaticae **76**, 1–18 (2006)

[2] Ehrenfeucht, A., Rozenberg, G.: Reaction systems: a formal framework for processes based on biochemical interactions. Electronic Communications of the EASST **26**, 1–10 (2010) https://doi.org/10.1007/978-3-642-02424-5_3

[3] Barbuti, R., Gori, R., Milazzo, P., Nasti, L.: A survey of gene regulatory networks modelling methods: from differential equations, to Boolean and qualitative bioinspired models. Journal of Membrane Computing **2**, 207–226 (2020) https://doi.org/10.1007/s41965-020-00046-y

[4] Barbuti, R., Gori, R., Milazzo, P.: Encoding Boolean networks into reaction systems for investigating causal dependencies in gene regulation. Theor. Comput. Sci. **881**, 3–24 (2021) https://doi.org/10.1016/j.tcs.2020.07.031

[5] Brodo, L., Bruni, R., Falaschi, M.: A logical and graphical framework for reaction systems. Theoretical Computer Science **875**, 1–27 (2021) https://doi.org/10.1016/j.tcs.2021.03.024

[6] Brodo, L., Bruni, R., Falaschi, M.: Dynamic slicing of reaction systems based on assertions and monitors. In: Hanus, M., Inclezan, D. (eds.) Proc. of Practical Aspects of Declarative Languages - 25th Int. Symp., PADL 2023. Lecture Notes in Computer Science, vol. 13880, pp. 107–124. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-24841-2_8

[7] Brodo, L., Bruni, R., Falaschi, M., Gori, R., Milazzo, P.: Attractor and slicing analysis of a T cell differentiation model based on Reaction Systems. In: Broccia, G., Cerone, A. (eds.) Proc. of 11th International Symposium From Data to Models and Back (DATAMOD 2023). Lecture Notes in Computer Science, vol. 14618, pp. 69–89. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-87217-4_4

[8] Brodo, L., Bruni, R., Falaschi, M., Gori, R., Milazzo, P., Montagna, V., Puleri, P.: Causal analysis of positive reaction systems. International Journal on Software Tools for Technology Transfer **26**, 509–526 (2024) https://doi.org/10.1016/j.tcs.2012.04.003

[9] BioReSolve web page, a Prolog interpreter for Reaction Systems analysis. Accessed: 1 April 2025. http://pages.di.unipi.it/bruni/LTSRS/

[10] Puniya, B.L.: CD4+ T cell Differentiation model webpage on the CellCollective platform. Accessed: 1 April 2025. https://research.cellcollective.org/?dashboard=true#module/6678:1/cd4-t-cell-differentiation/1

[11] Karlebach, G., Shamir, R.: Modelling and analysis of gene regulatory networks. Nature reviews Molecular cell biology **9**(10), 770–780 (2008) https://doi.org/10.1038/nrm2503

[12] Li, F., Long, T., Lu, Y., Ouyang, Q., Tang, C.: The yeast cell-cycle network is robustly designed. Proceedings of the National Academy of Sciences **101**(14), 4781–4786 (2004) https://doi.org/10.1073/pnas.0305937101

[13] Puniya, B.L., Todd, R.G., Mohammed, A., Brown, D.M., Barberis, M., Helikar, T.: 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 (2018) https://doi.org/10.3389/fphys.2018.00878

[14] Clark, A., Galpin, V., Gilmore, S., Guerriero, M.L., Hillston, J.: Formal methods for checking the consistency of biological models. In: Advances in Systems Biology, pp. 461–475 (2012). https://doi.org/10.1007/978-1-4419-7210-1_27 . Springer

[15] Bodei, C., Gori, R., Levi, F.: Causal static analysis for brane calculi. Theoretical Computer Science **587**, 73–103 (2015)

[16] Barbuti, R., Gori, R., Levi, F., Milazzo, P.: Investigating dynamic causalities in reaction systems. Theor. Comput. Sci. **623**, 114–145 (2016) https://doi.org/10.1016/j.tcs.2015.11.041

[17] Barbuti, R., Gori, R., Levi, F., Milazzo, P.: Generalized contexts for reaction systems: definition and study of dynamic causalities. Acta Informatica **55**, 227–267 (2018) https://doi.org/10.1007/s00236-017-0296-3

[18] Brijder, R., Ehrenfeucht, A., Main, M., Rozenberg, G.: A tour of reaction systems. Int. J. Found. Comput. Sci. **22**(07), 1499–1517 (2011)

https://doi.org/10.1142/S0129054111008842

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

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

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

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

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

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

[25] Bruni, R., Gori, R., Milazzo, P., Siboulet, H.: Melding boolean networks and reaction systems under synchronous, asynchronous and most permissive semantics. Nat. Comput. **23**(2), 235–267 (2024) https://doi.org/10.1007/S11047-024-09990-5

[26] Saez-Rodriguez, J., Simeoni, L., Lindquist, J.A., Hemenway, R., Bommhardt, U., Arndt, B., Haus, U.-U., Weismantel, R., Gilles, E.D., Klamt, S., *et al.*: A logical model provides insights into T cell receptor signaling. PLoS computational biology **3**(8), 163 (2007) https://doi.org/10.1371/journal.pcbi.0030163

[27] Thakar, J., Albert, R.: Boolean models of within-host immune interactions. Current opinion in microbiology **13**(3), 377–381 (2010) https://doi.org/10.1016/j.mib.2010.04.003

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

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

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

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

[32] Luckheeram, R.V., Zhou, R., Verma, A.D., Xia, B.: CD4+ T cells: differentiation and functions. Clinical and developmental immunology **2012** (2012) https://doi.org/10.1155/2012/925135

[33] SWI-Prolog home page. Accessed: 18 March 2024. https://www.swi-prolog.org/

[34] Brodo, L., Bruni, R., Falaschi, M.: A process algebraic approach to reaction systems. Theoretical Computer Science **881**, 62–82 (2021) https://doi.org/10.1016/j.tcs.2020.09.001

[35] Bornholdt, S.: Boolean network models of cellular regulation: prospects and limitations. Journal of the Royal Society Interface **5**(suppl_1), 85–94 (2008) https://doi.org/10.1098/rsif.2008.0132.focus

[36] Barbuti, R., Gori, R., Milazzo, P.: Encoding Boolean networks into reaction systems for investigating causal dependencies in gene regulation. Theoretical Computer Science **881**, 3–24 (2021) https://doi.org/10.1016/j.tcs.2020.07.031

[37] GitHub repository with the Python script developed for this paper. Accessed: 1 April 2025. https://github.com/Unipisa/TCell-AttractorsSlicingAnalyses

[38] Pham, G., Milazzo, P.: A comprehensive review of the use of shapley value to assess node importance in the analysis of biological networks. Computer Methods and Programs in Biomedicine Update **7**(100185) (2025) https://doi.org/10.1016/j.cmpbup.2025.100185

[39] Pham, G., Milazzo, P.: Gene importance assessment based on shapley value for boolean networks: Validation and scalability analysis. In: Proc. of 12th Int. Symposium "From Data to Models and Back (DataMod 2024)". Lecture Notes in Computer Science, vol. 15556, pp. 19–33. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-87908-1_2

[40] Brodo, L., Bruni, R., Falaschi, M., Gori, R., Levi, F., Milazzo, P.: Exploiting modularity of SOS semantics to define quantitative extensions of reaction systems. In: Aranha, C., Martín-Vide, C., Vega-Rodríguez, M.A. (eds.) Proc. of TPNC 2021. Lecture Notes in Computer Science, vol. 13082, pp. 15–32. Springer, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-030-90425-8_2

[41] Brodo, L., Bruni, R., Falaschi, M., Gori, R., Levi, F., Milazzo, P.: Quantitative extensions of reaction systems based on SOS semantics. Neural Comput. Appl. **35**(9), 6335–6359 (2023) https://doi.org/10.1007/s00521-022-07935-6

[42] Bodei, C., Brodo, L., Gori, R., Levi, F., Bernini, A., Hermith, D.: A static analysis for brane calculi providing global occurrence counting information. Theor. Comput. Sci. **696**, 11–51 (2017) https://doi.org/10.1016/J.TCS.2017.07.008

Appendix A Full lists of reactions of models considered in the paper

Here we provide the full lists of reactions of all relevant RS and Positive RS models described in the paper. In particular, we list reactions of all variants of the running toy example $\mathcal{A}_{\text{toy}}$ and of all variants of the T cell differentiation model $\mathcal{A}_{\text{Tcell}}$ introduced in Section 6.

A.1 Toy example model variants

The 50 reactions of the Positive RS $\mathcal{A}^{+}_{toy}$ considered in Example 5 are shown in Fig. A1. The 30 reactions of the minimized Positive RS $\mu(\mathcal{A}^{+}_{toy})$ considered in Example 6 are listed in Fig. A2.

A.2 T Cell model variants

The 49 reactions of the minimized RS model $\mu(\mathcal{A}_{Tcell})$ of T Cell differentiation are shown in Fig. A3. In Fig. A4 we show the 275 reactions of the non-minimized Positive RS model $\mathcal{A}^{+}_{Tcell}$ and in Fig. A5 their minimized version $\mu(\mathcal{A}^{+}_{Tcell})$ with 124 reactions.

```
react([il21],[],[il21r]),
react([il6],[],[il6r]),
react([],[],[neg_foxp3]),
react([],[],[neg_gata3]),
react([],[],[neg_il21]),
react([neg_il21],[],[neg_il21r]),
react([],[],[neg_il23r]),
react([],[],[neg_il6]),
react([neg_il6],[],[neg_il6r]),
react([foxp3],[],[neg_nfat]),
react([neg_tcr],[],[neg_nfat]),
react([foxp3],[],[neg_rorgt]),
react([foxp3,gata3],[],[neg_rorgt]),
react([foxp3,neg_il21r],[],[neg_rorgt]),
react([foxp3,neg_il6r],[],[neg_rorgt]),
react([foxp3,neg_stat3],[],[neg_rorgt]),
react([foxp3,neg_tgfbr],[],[neg_rorgt]),
react([foxp3,tbet],[],[neg_rorgt]),
react([gata3],[],[neg_rorgt]),
react([gata3,neg_il21r],[],[neg_rorgt]),
react([gata3,neg_il6r],[],[neg_rorgt]),
react([gata3,neg_stat3],[],[neg_rorgt]),
react([gata3,neg_tgfbr],[],[neg_rorgt]),
react([gata3,tbet],[],[neg_rorgt]),
react([neg_il21r,neg_il6r],[],[neg_rorgt]),
react([neg_il21r,neg_stat3],[],[neg_rorgt]),
react([neg_il21r,neg_tgfbr],[],[neg_rorgt]),
react([neg_il21r,tbet],[],[neg_rorgt]),
react([neg_il6r,neg_stat3],[],[neg_rorgt]),
react([neg_il6r,neg_tgfbr],[],[neg_rorgt]),
react([neg_il6r,tbet],[],[neg_rorgt]),
react([neg_stat3],[],[neg_rorgt]),
react([neg_stat3,neg_tgfbr],[],[neg_rorgt]),
react([neg_stat3,tbet],[],[neg_rorgt]),
react([neg_tgfbr],[],[neg_rorgt]),
react([neg_tgfbr,tbet],[],[neg_rorgt]),
react([tbet],[],[neg_rorgt]),
react([neg_il27],[],[neg_stat1]),
react([neg_nfat],[],[neg_stat1]),
react([neg_il23r],[],[neg_stat3]),
react([foxp3],[],[neg_tbet]),
react([neg_stat1],[],[neg_tbet]),
react([rorgt],[],[neg_tbet]),
react([],[],[neg_tgfbr]),
react([neg_foxp3,tcr],[],[nfat]),
react([il21r,neg_foxp3,neg_gata3,neg_tbet,stat3,tgfbr],[],[rorgt]),
react([il6r,neg_foxp3,neg_gata3,neg_tbet,stat3,tgfbr],[],[rorgt]),
react([il27,nfat],[],[stat1]),
react([il23r],[],[stat3]),
react([neg_foxp3,neg_rorgt,stat1],[],[tbet])
```

Fig. A1: Reactions of the Positive RS version of the toy model $\mathcal{A}^{+}_{toy}$ from Example 5 in BioReSolve syntax: non-minimized variant.

```
react([il21],[],[il21r]),
react([il6],[],[il6r]),
react([],[],[neg_foxp3]),
react([],[],[neg_gata3]),
react([],[],[neg_il21]),
react([neg_il21],[],[neg_il21r]),
react([],[],[neg_il23r]),
react([],[],[neg_il6]),
react([neg_il6],[],[neg_il6r]),
react([foxp3],[],[neg_nfat]),
react([neg_tcr],[],[neg_nfat]),
react([foxp3],[],[neg_rorgt]),
react([gata3],[],[neg_rorgt]),
react([neg_il21r,neg_il6r],[],[neg_rorgt]),
react([neg_stat3],[],[neg_rorgt]),
react([neg_tgfbr],[],[neg_rorgt]),
react([tbet],[],[neg_rorgt]),
react([neg_il27],[],[neg_stat1]),
react([neg_nfat],[],[neg_stat1]),
react([neg_il23r],[],[neg_stat3]),
react([foxp3],[],[neg_tbet]),
react([neg_stat1],[],[neg_tbet]),
react([rorgt],[],[neg_tbet]),
react([],[],[neg_tgfbr]),
react([neg_foxp3,tcr],[],[nfat]),
react([il21r,neg_foxp3,neg_gata3,neg_tbet,stat3,tgfbr],[],[rorgt]),
react([il6r,neg_foxp3,neg_gata3,neg_tbet,stat3,tgfbr],[],[rorgt]),
react([il27,nfat],[],[stat1]),
react([il23r],[],[stat3]),
react([neg_foxp3,neg_rorgt,stat1],[],[tbet])
```

Fig. A2: Reactions of the minimized Positive RS version $\mu(\mathcal{A}^{+}_{toy})$ of the toy model from Example 6 in BioReSolve syntax.

```
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([nfkb],[],[ifng]),
react([tbet],[stat3],[ifng]),
react([ifng,nfat],[],[ifngr]),
react([ifnge,nfat],[],[ifngr]),
react([il12,nfat],[],[il12r]),
react([tbet],[],[il12r]),
react([tcr],[gata3],[il12r]),
react([stat4],[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([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. A3: Reactions of the RS model $\mu(\mathcal{A}_{\text{Tcell}})$ in BioReSolve syntax: minimized variant. As an example, the reactions encoding the update rule for `IL12R` are colored in blue.

```
react([neg_gata3,neg_il21r,neg_il6r,stat5],[],[foxp3]),
react([neg_gata3,neg_il21r,neg_il6r,tgfbr],[],[foxp3]),
react([neg_gata3,neg_il21r,neg_stat3,stat5],[],[foxp3]),
react([neg_gata3,neg_il21r,neg_stat3,tgfbr],[],[foxp3]),
react([gata3,neg_tbet],[],[gata3]),
react([neg_foxp3,neg_rorgt,neg_tbet,neg_tgfb,nfat,stat6],[],[gata3]),
react([neg_foxp3,neg_rorgt,neg_tbet,neg_tgfb,stat5],[],[gata3]),
react([neg_stat3,neg_stat6,nfat,nfkb,stat4],[],[ifng]),
react([neg_stat3,tbet],[],[ifng]),
react([nfkb],[],[ifng]),
react([ifng,nfat],[],[ifngr]),
react([ifnge,nfat],[],[ifngr]),
react([il12,nfat],[],[il12r]),
react([neg_gata3,stat4],[],[il12r]),
react([neg_gata3,tcr],[],[il12r]),
react([tbet],[],[il12r]),
react([il17,il23r,neg_stat1,neg_stat5,stat3],[],[il17]),
react([neg_stat1,rorgt],[],[il17]),
react([il12,il18,neg_stat6],[],[il18r]),
react([neg_tbet,nfat,nfkb],[],[il2]),
react([nfat,stat3],[],[il21]),
react([il21],[],[il21r]),
react([il23,neg_tbet,stat3],[],[il23r]),
react([stat3],[],[il23r]),
react([il2,nfat],[],[il2r]),
react([gata3,neg_stat1,nfat],[],[il4]),
react([il4,neg_socs1],[],[il4r]),
react([il4e],[],[il4r]),
react([rorgt],[],[il6]),
react([il6],[],[il6r]),
react([il6e],[],[il6r]),
react([il18r],[],[irak]),
react([ifngr,neg_socs1],[],[jak1]),
react([gata3],[],[neg_foxp3]),
react([gata3,il21r],[],[neg_foxp3]),
react([gata3,il21r,il6r],[],[neg_foxp3]),
react([gata3,il21r,il6r,neg_stat5],[],[neg_foxp3]),
react([gata3,il21r,il6r,neg_tgfbr],[],[neg_foxp3]),
react([gata3,il21r,il6r,stat3],[],[neg_foxp3]),
react([gata3,il21r,neg_stat5],[],[neg_foxp3]),
react([gata3,il21r,neg_stat5,neg_tgfbr],[],[neg_foxp3]),
react([gata3,il21r,neg_stat5,stat3],[],[neg_foxp3]),
react([gata3,il21r,neg_tgfbr],[],[neg_foxp3]),
react([gata3,il21r,neg_tgfbr,stat3],[],[neg_foxp3]),
react([gata3,il21r,stat3],[],[neg_foxp3]),
react([gata3,il6r],[],[neg_foxp3]),
react([gata3,il6r,neg_stat5],[],[neg_foxp3]),
react([gata3,il6r,neg_stat5,neg_tgfbr],[],[neg_foxp3]),
react([gata3,il6r,neg_stat5,stat3],[],[neg_foxp3]),
react([gata3,il6r,neg_tgfbr],[],[neg_foxp3]),
react([gata3,il6r,neg_tgfbr,stat3],[],[neg_foxp3]),
react([gata3,il6r,stat3],[],[neg_foxp3]),
react([gata3,neg_stat5],[],[neg_foxp3]),
react([gata3,neg_stat5,neg_tgfbr],[],[neg_foxp3]),
react([gata3,neg_stat5,neg_tgfbr,stat3],[],[neg_foxp3]),
react([gata3,neg_stat5,stat3],[],[neg_foxp3]),
react([gata3,neg_tgfbr],[],[neg_foxp3]),
react([gata3,neg_tgfbr,stat3],[],[neg_foxp3]),
react([gata3,stat3],[],[neg_foxp3]),
react([il21r],[],[neg_foxp3]),
react([il21r,il6r],[],[neg_foxp3]),
react([il21r,il6r,neg_stat5],[],[neg_foxp3]),
react([il21r,il6r,neg_stat5,neg_tgfbr],[],[neg_foxp3]),
react([il21r,il6r,neg_stat5,stat3],[],[neg_foxp3]),
react([il21r,il6r,neg_tgfbr],[],[neg_foxp3]),
react([il21r,il6r,neg_tgfbr,stat3],[],[neg_foxp3]),
react([il21r,il6r,stat3],[],[neg_foxp3]),
react([il21r,neg_stat5],[],[neg_foxp3]),
react([il21r,neg_stat5,neg_tgfbr],[],[neg_foxp3]),
react([il21r,neg_stat5,neg_tgfbr,stat3],[],[neg_foxp3]),
react([il21r,neg_stat5,stat3],[],[neg_foxp3]),
react([il21r,neg_tgfbr],[],[neg_foxp3]),
react([il21r,neg_tgfbr,stat3],[],[neg_foxp3]),
react([il21r,stat3],[],[neg_foxp3]),
react([il6r,neg_stat5,neg_tgfbr],[],[neg_foxp3]),
react([il6r,neg_stat5,neg_tgfbr,stat3],[],[neg_foxp3]),
react([il6r,neg_stat5,stat3],[],[neg_foxp3]),
react([il6r,neg_tgfbr,stat3],[],[neg_foxp3]),
react([il6r,stat3],[],[neg_foxp3]),
react([neg_stat5,neg_tgfbr],[],[neg_foxp3]),
react([neg_stat5,neg_tgfbr,stat3],[],[neg_foxp3]),
react([foxp3,neg_gata3],[],[neg_gata3]),
react([foxp3,neg_gata3,neg_nfat],[],[neg_gata3]),
react([foxp3,neg_gata3,neg_stat5],[],[neg_gata3]),
react([foxp3,neg_gata3,neg_stat6],[],[neg_gata3]),
react([foxp3,neg_gata3,rorgt],[],[neg_gata3]),
react([foxp3,neg_gata3,tbet],[],[neg_gata3]),
react([foxp3,neg_gata3,tgfb],[],[neg_gata3]),
react([foxp3,neg_nfat,tbet],[],[neg_gata3]),
react([foxp3,neg_stat5,tbet],[],[neg_gata3]),
react([foxp3,neg_stat6,tbet],[],[neg_gata3]),
react([foxp3,rorgt,tbet],[],[neg_gata3]),
react([foxp3,tbet],[],[neg_gata3]),
react([foxp3,tbet,tgfb],[],[neg_gata3]),
react([neg_gata3,neg_nfat,neg_stat5],[],[neg_gata3]),
react([neg_gata3,neg_nfat,rorgt],[],[neg_gata3]),
react([neg_gata3,neg_nfat,tbet],[],[neg_gata3]),
react([neg_gata3,neg_nfat,tgfb],[],[neg_gata3]),
react([neg_gata3,neg_stat5,rorgt],[],[neg_gata3]),
react([neg_gata3,neg_stat5,tbet],[],[neg_gata3]),
react([neg_gata3,neg_stat5,tgfb],[],[neg_gata3]),
react([neg_gata3,neg_stat6,rorgt],[],[neg_gata3]),
react([neg_gata3,neg_stat6,tbet],[],[neg_gata3]),
react([neg_gata3,neg_stat6,tgfb],[],[neg_gata3]),
react([neg_gata3,rorgt],[],[neg_gata3]),
react([neg_gata3,rorgt,tbet],[],[neg_gata3]),
react([neg_gata3,rorgt,tgfb],[],[neg_gata3]),
react([neg_gata3,tbet],[],[neg_gata3]),
react([neg_gata3,tbet,tgfb],[],[neg_gata3]),
react([neg_gata3,tgfb],[],[neg_gata3]),
react([neg_nfat,neg_stat5,tbet],[],[neg_gata3]),
react([neg_nfat,rorgt,tbet],[],[neg_gata3]),
react([neg_nfat,tbet],[],[neg_gata3]),
react([neg_nfat,tbet,tgfb],[],[neg_gata3]),
react([neg_stat5,neg_stat6,tbet],[],[neg_gata3]),
react([neg_stat5,rorgt,tbet],[],[neg_gata3]),
react([neg_stat5,tbet],[],[neg_gata3]),
react([neg_stat5,tbet,tgfb],[],[neg_gata3]),
react([neg_stat6,rorgt,tbet],[],[neg_gata3]),
react([neg_stat6,tbet],[],[neg_gata3]),
react([neg_stat6,tbet,tgfb],[],[neg_gata3]),
react([rorgt,tbet],[],[neg_gata3]),
react([rorgt,tbet,tgfb],[],[neg_gata3]),
react([tbet],[],[neg_gata3]),
react([tbet,tgfb],[],[neg_gata3]),
react([neg_nfat,neg_nfkb,neg_tbet],[],[neg_ifng]),
react([neg_nfat,neg_nfkb,stat3],[],[neg_ifng]),
react([neg_nfkb,neg_stat4,neg_tbet],[],[neg_ifng]),
react([neg_nfkb,neg_stat4,stat3],[],[neg_ifng]),
react([neg_nfkb,neg_tbet],[],[neg_ifng]),
react([neg_nfkb,neg_tbet,stat3],[],[neg_ifng]),
react([neg_nfkb,neg_tbet,stat6],[],[neg_ifng]),
react([neg_nfkb,stat3],[],[neg_ifng]),
react([neg_nfkb,stat3,stat6],[],[neg_ifng]),
react([neg_ifng,neg_ifnge],[],[neg_ifngr]),
react([neg_ifng,neg_nfat],[],[neg_ifngr]),
react([neg_ifnge,neg_nfat],[],[neg_ifngr]),
react([neg_nfat],[],[neg_ifngr]),
react([gata3,neg_il12,neg_stat4,neg_tbet],[],[neg_il12r]),
react([gata3,neg_il12,neg_tbet],[],[neg_il12r]),
react([gata3,neg_il12,neg_tbet,neg_tcr],[],[neg_il12r]),
react([gata3,neg_nfat,neg_stat4,neg_tbet],[],[neg_il12r]),
react([gata3,neg_nfat,neg_tbet],[],[neg_il12r]),
react([gata3,neg_nfat,neg_tbet,neg_tcr],[],[neg_il12r]),
react([neg_il12,neg_stat4,neg_tbet,neg_tcr],[],[neg_il12r]),
react([neg_nfat,neg_stat4,neg_tbet,neg_tcr],[],[neg_il12r]),
react([neg_il17,neg_rorgt],[],[neg_il17]),
react([neg_il17,stat1],[],[neg_il17]),
react([neg_il23r,neg_rorgt],[],[neg_il17]),
react([neg_il23r,stat1],[],[neg_il17]),
react([neg_rorgt,neg_stat3],[],[neg_il17]),
react([neg_rorgt,stat1],[],[neg_il17]),
react([neg_rorgt,stat5],[],[neg_il17]),
react([neg_stat3,stat1],[],[neg_il17]),
react([stat1],[],[neg_il17]),
react([stat1,stat5],[],[neg_il17]),
react([neg_il12],[],[neg_il18r]),
react([neg_il18],[],[neg_il18r]),
react([stat6],[],[neg_il18r]),
react([neg_nfat],[],[neg_il2]),
react([neg_nfkb],[],[neg_il2]),
react([tbet],[],[neg_il2]),
react([neg_nfat],[],[neg_il21]),
react([neg_stat3],[],[neg_il21]),
react([neg_il21],[],[neg_il21r]),
react([neg_il23,neg_stat3],[],[neg_il23r]),
react([neg_stat3],[],[neg_il23r]),
react([neg_stat3,tbet],[],[neg_il23r]),
react([neg_il2],[],[neg_il2r]),
react([neg_nfat],[],[neg_il2r]),
react([neg_gata3],[],[neg_il4]),
react([neg_nfat],[],[neg_il4]),
react([stat1],[],[neg_il4]),
react([neg_il4,neg_il4e],[],[neg_il4r]),
react([neg_il4e,socs1],[],[neg_il4r]),
react([neg_rorgt],[],[neg_il6]),
react([neg_il6,neg_il6e],[],[neg_il6r]),
react([neg_il18r],[],[neg_irak]),
react([neg_ifngr],[],[neg_jak1]),
react([socs1],[],[neg_jak1]),
react([foxp3],[],[neg_nfat]),
react([neg_tcr],[],[neg_nfat]),
react([foxp3],[],[neg_nfkb]),
react([neg_irak],[],[neg_nfkb]),
react([foxp3],[],[neg_rorgt]),
react([foxp3,gata3],[],[neg_rorgt]),
react([foxp3,neg_il21r],[],[neg_rorgt]),
react([foxp3,neg_il6r],[],[neg_rorgt]),
react([foxp3,neg_stat3],[],[neg_rorgt]),
react([foxp3,neg_tgfbr],[],[neg_rorgt]),
react([foxp3,tbet],[],[neg_rorgt]),
react([gata3],[],[neg_rorgt]),
react([gata3,neg_il21r],[],[neg_rorgt]),
react([gata3,neg_il6r],[],[neg_rorgt]),
react([gata3,neg_stat3],[],[neg_rorgt]),
react([gata3,neg_tgfbr],[],[neg_rorgt]),
react([gata3,tbet],[],[neg_rorgt]),
react([neg_il21r,neg_il6r],[],[neg_rorgt]),
react([neg_il21r,neg_stat3],[],[neg_rorgt]),
react([neg_il21r,neg_tgfbr],[],[neg_rorgt]),
react([neg_il21r,tbet],[],[neg_rorgt]),
react([neg_il6r,neg_stat3],[],[neg_rorgt]),
react([neg_il6r,neg_tgfbr],[],[neg_rorgt]),
react([neg_il6r,tbet],[],[neg_rorgt]),
react([neg_stat3],[],[neg_rorgt]),
react([neg_stat3,neg_tgfbr],[],[neg_rorgt]),
react([neg_stat3,tbet],[],[neg_rorgt]),
react([neg_tgfbr],[],[neg_rorgt]),
react([neg_tgfbr,tbet],[],[neg_rorgt]),
react([tbet],[],[neg_rorgt]),
react([neg_stat1,neg_tbet],[],[neg_socs1]),
react([neg_il27,neg_jak1],[],[neg_stat1]),
react([neg_jak1,neg_nfat],[],[neg_stat1]),
react([neg_il21r,neg_il23r,neg_il6r],[],[neg_stat3]),
react([gata3],[],[neg_stat4]),
react([neg_il12],[],[neg_stat4]),
react([neg_il12r],[],[neg_stat4]),
react([neg_il2r],[],[neg_stat5]),
react([ifng],[],[neg_stat6]),
react([neg_il4r],[],[neg_stat6]),
react([socs1],[],[neg_stat6]),
react([foxp3],[],[neg_tbet]),
react([foxp3,ifng],[],[neg_tbet]),
react([foxp3,ifng,neg_stat1],[],[neg_tbet]),
react([foxp3,ifng,neg_stat4],[],[neg_tbet]),
react([foxp3,ifng,rorgt],[],[neg_tbet]),
react([foxp3,il12],[],[neg_tbet]),
react([foxp3,il12,neg_stat1],[],[neg_tbet]),
react([foxp3,il12,neg_stat4],[],[neg_tbet]),
react([foxp3,il12,rorgt],[],[neg_tbet]),
react([foxp3,neg_stat1],[],[neg_tbet]),
react([foxp3,neg_stat1,neg_stat4],[],[neg_tbet]),
react([foxp3,neg_stat1,neg_tbet],[],[neg_tbet]),
react([foxp3,neg_stat1,rorgt],[],[neg_tbet]),
react([foxp3,neg_stat4],[],[neg_tbet]),
react([foxp3,neg_stat4,neg_tbet],[],[neg_tbet]),
react([foxp3,neg_stat4,rorgt],[],[neg_tbet]),
react([foxp3,neg_tbet],[],[neg_tbet]),
react([foxp3,neg_tbet,rorgt],[],[neg_tbet]),
react([foxp3,rorgt],[],[neg_tbet]),
react([ifng,neg_stat1,neg_stat4],[],[neg_tbet]),
react([ifng,neg_stat1,rorgt],[],[neg_tbet]),
react([ifng,neg_stat4,rorgt],[],[neg_tbet]),
react([ifng,rorgt],[],[neg_tbet]),
react([il12,neg_stat1,neg_stat4],[],[neg_tbet]),
react([il12,neg_stat1,rorgt],[],[neg_tbet]),
react([il12,neg_stat4,rorgt],[],[neg_tbet]),
react([il12,rorgt],[],[neg_tbet]),
react([neg_stat1,neg_stat4,neg_tbet],[],[neg_tbet]),
react([neg_stat1,neg_stat4,rorgt],[],[neg_tbet]),
react([neg_stat1,neg_tbet,rorgt],[],[neg_tbet]),
react([neg_stat1,rorgt],[],[neg_tbet]),
react([neg_stat4,neg_tbet,rorgt],[],[neg_tbet]),
react([neg_stat4,rorgt],[],[neg_tbet]),
react([neg_tbet,rorgt],[],[neg_tbet]),
react([rorgt],[],[neg_tbet]),
react([neg_nfat],[],[neg_tgfbr]),
react([neg_tgfbl],[],[neg_tgfbr]),
react([neg_foxp3,tcr],[],[nfat]),
react([irak,neg_foxp3],[],[nfkb]),
react([il21r,neg_foxp3,neg_gata3,neg_tbet,stat3,tgfbr],[],[rorgt]),
react([il6r,neg_foxp3,neg_gata3,neg_tbet,stat3,tgfbr],[],[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,neg_gata3],[],[stat4]),
react([il12r],[],[stat5]),
react([il4r,neg_ifng,neg_socs1],[],[stat6]),
react([neg_foxp3,neg_ifng,neg_il12,neg_rorgt,tbet],[],[tbet]),
react([neg_foxp3,neg_rorgt,stat1],[],[tbet]),
react([neg_foxp3,neg_rorgt,stat4],[],[tbet]),
react([nfat,tgfb],[],[tgfbr])
```

Fig. A4: Reactions of the RS model $\mathcal{A}^{+}_{\mathrm{Tcell}}$ in BioReSolve syntax: positive non-minimized variant. As an example, the reactions encoding the update rule for `IL12R` are colored in blue.

```
react([neg_gata3,neg_il21r,neg_il6r,stat5],[],[foxp3]),
react([neg_gata3,neg_il21r,neg_il6r,tgfbr],[],[foxp3]),
react([neg_gata3,neg_il21r,neg_stat3,stat5],[],[foxp3]),
react([neg_gata3,neg_il21r,neg_stat3,tgfbr],[],[foxp3]),
react([gata3,neg_tbet],[],[gata3]),
react([neg_foxp3,neg_rorgt,neg_tbet,neg_tgfb,nfat,stat6],[],[gata3]),
react([neg_foxp3,neg_rorgt,neg_tbet,neg_tgfb,stat5],[],[gata3]),
react([neg_stat3,tbet],[],[ifng]),
react([nfkb],[],[ifng]),
react([ifng,nfat],[],[ifngr]),
react([ifnge,nfat],[],[ifngr]),
react([il12,nfat],[],[il12r]),
react([neg_gata3,stat4],[],[il12r]),
react([neg_gata3,tcr],[],[il12r]),
react([tbet],[],[il12r]),
react([il17,il23r,neg_stat1,neg_stat5,stat3],[],[il17]),
react([neg_stat1,rorgt],[],[il17]),
react([il12,il18,neg_stat6],[],[il18r]),
react([neg_tbet,nfat,nfkb],[],[il2]),
react([nfat,stat3],[],[il21]),
react([il21],[],[il21r]),
react([stat3],[],[il23r]),
react([il2,nfat],[],[il2r]),
react([gata3,neg_stat1,nfat],[],[il4]),
react([il4,neg_socs1],[],[il4r]),
react([il4e],[],[il4r]),
react([rorgt],[],[il6]),
react([il6],[],[il6r]),
react([il6e],[],[il6r]),
react([il18r],[],[irak]),
react([ifngr,neg_socs1],[],[jak1]),
react([gata3],[],[neg_foxp3]),
react([il21r],[],[neg_foxp3]),
react([il6r,stat3],[],[neg_foxp3]),
react([neg_stat5,neg_tgfbr],[],[neg_foxp3]),
react([foxp3,neg_gata3],[],[neg_gata3]),
react([neg_gata3,neg_nfat,neg_stat5],[],[neg_gata3]),
react([neg_gata3,neg_stat5,neg_stat6],[],[neg_gata3]),
react([neg_gata3,rorgt],[],[neg_gata3]),
react([neg_gata3,tgfb],[],[neg_gata3]),
react([tbet],[],[neg_gata3]),
react([neg_nfkb,neg_tbet],[],[neg_ifng]),
react([neg_nfkb,stat3],[],[neg_ifng]),
react([neg_ifng,neg_ifnge],[],[neg_ifngr]),
react([neg_nfat],[],[neg_ifngr]),
react([gata3,neg_il12,neg_tbet],[],[neg_il12r]),
react([gata3,neg_nfat,neg_tbet],[],[neg_il12r]),
react([neg_il12,neg_stat4,neg_tbet,neg_tcr],[],[neg_il12r]),
react([neg_nfat,neg_stat4,neg_tbet,neg_tcr],[],[neg_il12r]),
react([neg_il17,neg_rorgt],[],[neg_il17]),
react([neg_il23r,neg_rorgt],[],[neg_il17]),
react([neg_rorgt,neg_stat3],[],[neg_il17]),
react([neg_rorgt,stat5],[],[neg_il17]),
react([stat1],[],[neg_il17]),
react([neg_il12],[],[neg_il18r]),
react([neg_il18],[],[neg_il18r]),
react([stat6],[],[neg_il18r]),
react([neg_nfat],[],[neg_il2]),
react([neg_nfkb],[],[neg_il2]),
react([tbet],[],[neg_il2]),
react([neg_nfat],[],[neg_il21]),
react([neg_stat3],[],[neg_il21]),
react([neg_il21],[],[neg_il21r]),
react([neg_stat3],[],[neg_il23r]),
react([neg_il2],[],[neg_il2r]),
react([neg_nfat],[],[neg_il2r]),
react([neg_gata3],[],[neg_il4]),
react([neg_nfat],[],[neg_il4]),
react([stat1],[],[neg_il4]),
react([neg_il4,neg_il4e],[],[neg_il4r]),
react([neg_il4e,socs1],[],[neg_il4r]),
react([neg_rorgt],[],[neg_il6]),
react([neg_il6,neg_il6e],[],[neg_il6r]),
react([neg_il18r],[],[neg_irak]),
react([neg_ifngr],[],[neg_jak1]),
react([socs1],[],[neg_jak1]),
react([foxp3],[],[neg_nfat]),
react([neg_tcr],[],[neg_nfat]),
react([foxp3],[],[neg_nfkb]),
react([neg_irak],[],[neg_nfkb]),
react([foxp3],[],[neg_rorgt]),
react([gata3],[],[neg_rorgt]),
react([neg_il21r,neg_il6r],[],[neg_rorgt]),
react([neg_stat3],[],[neg_rorgt]),
react([neg_tgfbr],[],[neg_rorgt]),
react([tbet],[],[neg_rorgt]),
react([neg_stat1,neg_tbet],[],[neg_socs1]),
react([neg_il27,neg_jak1],[],[neg_stat1]),
react([neg_jak1,neg_nfat],[],[neg_stat1]),
react([neg_il21r,neg_il23r,neg_il6r],[],[neg_stat3]),
react([gata3],[],[neg_stat4]),
react([neg_il12],[],[neg_stat4]),
react([neg_il12r],[],[neg_stat4]),
react([neg_il2r],[],[neg_stat5]),
react([ifng],[],[neg_stat6]),
react([neg_il4r],[],[neg_stat6]),
react([socs1],[],[neg_stat6]),
react([foxp3],[],[neg_tbet]),
react([ifng,neg_stat1,neg_stat4],[],[neg_tbet]),
react([il12,neg_stat1,neg_stat4],[],[neg_tbet]),
react([neg_stat1,neg_stat4,neg_tbet],[],[neg_tbet]),
react([rorgt],[],[neg_tbet]),
react([neg_nfat],[],[neg_tgfbr]),
react([neg_tgfb],[],[neg_tgfbr]),
react([neg_foxp3,tcr],[],[nfat]),
react([irak,neg_foxp3],[],[nfkb]),
react([il21r,neg_foxp3,neg_gata3,neg_tbet,stat3,tgfbr],[],[rorgt]),
react([il6r,neg_foxp3,neg_gata3,neg_tbet,stat3,tgfbr],[],[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,neg_gata3],[],[stat4]),
react([il2r],[],[stat5]),
react([il4r,neg_ifng,neg_socs1],[],[stat6]),
react([neg_foxp3,neg_ifng,neg_il12,neg_rorgt,tbet],[],[tbet]),
react([neg_foxp3,neg_rorgt,stat1],[],[tbet]),
react([neg_foxp3,neg_rorgt,stat4],[],[tbet]),
react([nfat,tgfb],[],[tgfbr])
```

Fig. A5: Reactions of the RS model $\mu(\mathcal{A}^{+}_{\text{Tcell}})$ in BioReSolve syntax: positive minimized variant. As an example, the reactions encoding the update rule for `IL12R` are colored in blue.