Process Calculi and Rewriting Techniques for Analyzing Reaction Systems*

Demis Ballis[1], Linda Brodo[2], Moreno Falaschi[3], and Carlos Olarte[4]

[1] DMIF, University of Udine, Italy
[2] Dipartimento di Scienze economiche e aziendali, Università di Sassari, Italy
[3] Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche Università di Siena, Italy
[4] LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord, France

Abstract. Reaction Systems (RSs) are a successful computational framework inspired by biological systems, where entities are produced by reactions and can be used to enable or inhibit other reactions. In RSs interaction with the environment is modeled by a linear sequence of sets of entities called context. In this paper, we present a new interaction language, called **ccReact** for implementing and specifying RSs. **ccReact** extends the classic RS computational model by allowing the specification of possibly recursive, nondeterministic conditional context sequences, enhancing the interactive capabilities of the models. We provide a rewriting logic semantics for **ccReact**, making it executable in the Maude system. This enables the simulation and the verification of (reachability) properties, and model checking temporal (LTL and CTL) formulas for our extended RSs. We analyze a breast cancer case study and show that our model checking analyses can in some cases improve the administration of monoclonal antibodies therapeutic treatments.

Keywords: reaction systems, model checking, SOS, rewriting logic, Maude

1 Introduction

Reaction Systems (RSs) [18] are a computational framework inspired by systems of living cells. Since their introduction, RSs have shown to be a general computational model with several applications ranging from the modeling of biological phenomena [8,23,7,12], and molecular chemistry [33] to theoretical foundations of computing [25,26]. A RS is built from a finite set of entities and a finite set of reactions acting on entities. A reaction is a triple (R, I, P) where R is the set of *reactants* (entities whose presence is needed to enable the reaction), I is the set of *inhibitors* (entities whose absence is needed to enable the reaction) and P is the set of *products* (entities that are produced if the reaction takes place).

* Research supported by the Next Generation EU project PNRR ECS00000017 "THE - Tuscany Health Ecosystem" (Spoke 3 - CUP B63C22000680007), NextGenerationEU projects MEDICA (PRIN 2022, CUP_B53D23013170006), and DELICE (PRIN PNRR 2022, P20223T2MF)

The behavior of a RS is typically specified as a rewrite system that models *processes interacting* with an external *context*. The external context specifies the entities provided by the environment at each step of the process. The current system state is determined by the union of the entities coming from the environment with those produced in the previous state. A state transition is then determined by applying all and only the enabled reactions in the current state.

The specification of RSs may be an error-prone activity, since system states typically contain a large number of entities which might be deeply interconnected due to the form of the reactions in play. In this scenario, the verification of RSs is a fundamental albeit nontrivial task which requires a proper automated support.

Rewriting Logic (RL) [31] is a very general *logical* and *semantic framework* that is particularly suitable for modeling and analyzing complex, nondeterministic systems. Rewriting Logic is efficiently implemented in the high-performance language Maude [24], which seamlessly integrates functional, logic, concurrent, and object-oriented computations. The specification unit in RL is a *rewrite theory* $\mathcal{R}$, that combines a set of rewrite rules R modeling the concurrent transitions of a system, and an equational theory $\mathcal{E}$, representing system states as terms of an algebraic datatype.

Contributions. This paper presents a new interaction language, called **ccReact**, for specifying RSs. The language is equipped with an *executable* rewriting logic semantics, suitable for both simulation and verification. **ccReact** extends the classic RS computational model by allowing the specification of context sequences that model external stimuli coming from the environment. It exploits process algebraic operators (such as action prefix, sum and recursion) and offers the possibility to express conditional statements. The advantage of having a language for specifying contexts is twofold. First, we can recursively define contexts that possibly exhibit complex nondeterministic behaviors in a precise way, as sometimes have already appeared in the literature [28,20,21]; second, we are able to represent a collection of experiments within a single semantic object, so that the consequences of some variation in the context sequence can then be more easily compared and analyzed. Our approach thus overcomes the usual limitation of *in silico* experiments, where a single context sequence is synthesized and one can only observe the outcomes for that specific input without the possibility of analyzing context changes in real time.

The semantics of **ccReact** is formalized within the RL framework, thus making available for it the rich set of Maude analysis tools. Our rewrite theory can be directly used to: (1) simulate biological experiments; (ii) perform reachability analysis; and (iii) model-check CTL and LTL temporal properties. The usefulness of our approach is evaluated on a biological case study in [27] that analyses the behavior of the protein signaling networks for the HER2-positive breast cancer subtype in the presence of different combinations of monoclonal antibody drugs. The authors in [27] aim at achieving the best drug treatment for three different breast cancer representative cell lines: BT474, SKBR3 and HCC1954. Our analyses not only corroborate previous results in the literature concerning this system, but also allow us to validate new hypotheses.

Organization. Section 2 recalls the basics of RSs and introduces the case study used as a running example. Section 3 first introduces the syntax for **ccReact** and then its operational semantics, and we prove that it adequately captures the behavior of RS interactive processes (Theorems 1 and 2). After a brief overview on RL, Section 4 provides a RL characterization of the operational semantics of **ccReact**. In Section 5, we show how our analysis methods can be used to validate already known and new hypotheses about the case study. Section 6 reviews related work and Section 7 concludes the paper. The complete Maude specification and case studies are available in the companion tool [11].

2 Reaction Systems

In this section we recall some basic notions about RSs that are relevant to this work. For a full discussion refer to [18]. We also introduce our case study.

We use the term *entities* to denote generic molecular substances (e.g., atoms, ions, molecules) that may be present in the states of a biochemical system. The main mechanisms that regulate the functioning of a living cell are *facilitation* and *inhibition*. These mechanisms are based on the presence and absence of entities and are reflected in the basic definitions of RSs.

Definition 1 (Reaction Systems). *Let S be a (finite) set of entities. A reaction in S is a triple $a = (R, I, P)$, where $R, I, P \subseteq S$ are finite sets and $R \cap I = \emptyset$. We use $rac(S)$ to denote the set of all reactions on the set of entities S. A Reaction System (RS) is a pair $\mathcal{A} = (S, A)$ s.t. S is a finite set of entities, and $A \subseteq rac(S)$ is a finite set of reactions.*

The sets R, I, P are the sets of *reactants*, *inhibitors*, and *products*, respectively. Due to biological considerations, the sets R and P are usually nonempty: there is no creation from nothing ($R \neq \emptyset$), and if a reaction takes place then something is produced ($P \neq \emptyset$). A reaction can take place whenever all of its reactants are present in a given state while none of its inhibitors is present. If this happens, the reaction is enabled and creates its products. More formally,

Definition 2 (Reaction Result). *Given a (finite) set of entities S, and a subset $W \subseteq S$, we define the following:*

1. *Let $a = (R, I, P) \in rac(S)$ be a reaction in S. The result of reaction a on W, denoted by $res_a(W)$, is defined by: $res_a(W) \triangleq \begin{cases} P & \text{if } en_a(W) \\ \emptyset & \text{otherwise} \end{cases}$ where the* enabling *predicate is defined by $en_a(W) \triangleq (R \subseteq W) \wedge (I \cap W = \emptyset)$.*
2. *Let $A \subseteq rac(S)$ be a finite set of reactions. The result of A on W, denoted by $res_A(W)$, is defined by: $res_A(W) \triangleq \bigcup_{a \in A} res_a(W)$.*

RSs are based on three assumptions: **no permanency**, i.e., any entity vanishes unless it is sustained by a reaction; **no counting**, the basic model of RSs is very abstract and qualitative, i.e. the quantity of entities that are present in

a cell is not considered; **threshold nature of resources**, it is assumed that either an entity is available for all reactions, or it is not available at all.

Since living cells are seen as open systems that interact with the external environment, the behavior of a RS is formalized through the notion of *interactive process*, that is, a process that may react to external stimuli.

Definition 3 (Interactive Process). *Let $\mathcal{A} = (S, A)$ be a RS and let $n \geq 0$. An n-step* interactive process *in $\mathcal{A}$ is a pair $\pi = (\gamma, \delta)$ 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 $i \in [0, n]$, $D_0 = \emptyset$, and $D_{i+1} = res_A(D_i \cup C_i)$ for any $i \in [0, n-1]$. We call $\tau = W_0, \ldots, W_n$ with $W_i \triangleq C_i \cup D_i$, for any $i \in [0, n]$ the* state sequence*.*

The context sequence γ represents the environment interacting with the RS. It specifies the *input* that comes from the environment. The result sequence δ is entirely determined by γ and the reactions in A. As expected, distinct context sequences may lead to distinct outcomes for the same reaction set A, but a context sequence γ determines a *unique* result sequence δ.

Case Study. We consider the case study in [27] concerning the protein signaling networks in breast cancer and focus on a particular subtype of breast cancer: the HER2-positive subtype. This cancer subtype is characterized by an over expression of the human epidermal growth factor receptor 2 (HER2, also named ErbB-2). This is a receptor tyrosine kinase (RTK), and it is part of the epidermal growth factor (EGF) receptor family also composed by ErbB-1, ErbB-3, and ErbB-4. These four receptors form different homo- or hetero-dimers via growth factor activation (for example EGF for ErbB-1 or heregulin, HRG, for ErbB-3), that activate a signaling pathway that can contribute to the breast cancer tumor genesis and progression. The therapeutic treatment is typically based on monoclonal antibody drugs. Unfortunately, the therapy is often weakened by drug resistance that can be caused by the pathway deregulatory activity or by the ErbB-family receptors that allow to bypass the drug activity.

This case study focuses on a treatment where different drug combinations are tested to improve drug efficiency. The experimental analyses in [27] consider: three different kinds of breast cancer, identified by the cell lines BT474, HCC1954, and SKBR3; three types of drugs: erlotinib (e), pertuzumab (p), trastuzumab (t), provided alone or in combination; and two types of input stimulus: EGF and HRG. Input stimuli enable the activation of the EGF receptors and start the signaling pathways to be analyzed. Experiments in [27] aims at finding specific *attractors* in steady states, that is, entities that are key in the tumor genesis and progression which are present in a stationary system configuration. The search is carried out w.r.t. a persistent stimulus that includes both EGF and HRG and different drug combinations. Both short- and long-term experiments have been conducted using two distinct classes of attractors: attractors AKT, ERK1/2, p70S6K (short-term analysis), and attractors RPS6, RB (long-term analysis). Table 1 summarizes the results of [27] for the six considered systems (one per kind of cancer cell line, short and long term versions). Each

row of the table presents the efficacy of a drug combination under the same external stimulus for the three kinds of cancer (x stands for no drug treatment). The treatment is effective when attractors are not present in a steady state (cell value is equal to 0). In [27], a qualitative model, based on boolean networks, is used to describe the systems under consideration. To generate our reactions for the six systems in [27], we follow the translation method in [13] which allow Boolean networks to be directly converted into reaction sets. The resulting RSs we obtain for the six systems in Table 1, from left to right, include 39, 39, 34, 47, 50 and 55 reaction rules respectively.

	Short-term									Long-term					
	BT474			HCC1954			SKBR3			BT474		HCC1954		SKBR3	
	AKT	ERK1/2	p70S6K	AKT	ERK1/2	p70S6K	AKT	ERK1/2	p70S6K	RPS6	RB	RPS6	RB	RPS6	RB
x	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0
e	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0
p	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
t	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0
e,p	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0
e,t	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0
p,t	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
e,p,t	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0

Table 1: Short-term and long-term experiments from [27].

Below we use some of the involved entities and reactions to illustrate the above definitions.

Example 1. Let $\mathcal{A} = (S, A)$ be the RS where $S = \{\mathsf{egf}, \mathsf{e}, \mathsf{p}, \mathsf{erbb1}, \mathsf{erk12}, \mathsf{p70s6k}\}$, and $A = \{a_1, a_2, a_3\}$ with $a_1 = (\{\mathsf{egf}\}, \{\mathsf{e}, \mathsf{p}\}, \{\mathsf{erbb1}\})$, $a_2 = (\{\mathsf{egf}\}, \emptyset, \{\mathsf{erk12}\})$, $a_3 = (\{\mathsf{erk12}\}, \emptyset, \{\mathsf{p70s6k}\})$. Consider the context sequence $\gamma = C_0, \cdots, C_3$, with $C_0 = \{\mathsf{egf}, \mathsf{e}\}$, $C_1 = \emptyset$, $C_2 = \{\mathsf{egf}\}$, and $C_3 = \emptyset$. Then, $\pi = (\gamma, \delta)$ in an interactive process where $\delta = D_0, \cdots, D_3$, with $D_0 = \emptyset$, $D_1 = \{\mathsf{erk12}\}$, $D_2 = \{\mathsf{p70s6k}\}$, and $D_3 = \{\mathsf{erbb1}, \mathsf{erk12}\}$. The resulting state sequence is $\tau = W_0, \cdots, W_3$, where $W_0 = \{\mathsf{egf}, \mathsf{e}\}$, $W_1 = \{\mathsf{erk12}\}$, $W_2 = \{\mathsf{egf}, \mathsf{p70s6k}\}$, and $W_3 = \{\mathsf{erbb1}, \mathsf{erk12}\}$.

3 ccReact: A Language for Executing and Analyzing RSs

Drawing inspiration from process algebras, this section introduces the syntax and the operational semantics for **ccReact**, a language of processes that allows for the definition of external contexts interacting with a RS. The purpose of **ccReact** processes is to provide stimuli to the RS in the form of a context C, i.e., a subset of the set of entities *controlled* by the environment. Our language includes constructors to model the situation in *in silico* experiments where different alternative inputs need to be tested and some of them may depend on the current state of the system. This opens up the possibility of designing new experiments as shown later in Section 5. Next we define the syntax of **ccReact**:

Definition 4 (ccReact Processes). *The language of processes is defined as:*

$$
\begin{array}{lll}
c & ::= e \mid \overline{e} \mid c \wedge c \mid c \vee c & \textit{(Conditions)} \\
p & ::= C \mid c \rightarrow C & \textit{(Prefixes)} \\
K, M, N & ::= \mathbf{0} \mid X \mid \sum_{i \in I} p_i.K_i \mid K \parallel M \mid \mathbf{rec}\ X.K & \textit{(Processes)}
\end{array}
$$

where e is an entity, X is a process variable, I is a finite nonempty set of indices, and $C \subseteq S$ is a possibly empty set of entities. The set of processes is denoted by **Proc** *and the set of prefixes by* **Pref**. *Processes are quotiented by a structural*

congruence relation $\equiv$ *satisfying: (1)* $K \equiv M$ *if they differ only by a renaming of local variables in recursive definitions (alpha-conversion); (2)* $K \parallel M \equiv M \parallel K$*; (3)* $K \parallel (M \parallel N) \equiv (K \parallel M) \parallel N$*; and (4)* $K \parallel \mathbf{0} \equiv \mathbf{0} \parallel K \equiv K$*.*

Let $D \subseteq S$ be a set of entities. The *prefix* C is always *enabled* in D and it *produces* the set of entities C. The prefix $c \ \rightarrow \ C$ is *enabled* if condition c holds in D. This happens when the function $check(c, D)$ evaluates to true where: $check(e, D) \triangleq e \in D$, $check(\overline{e}, D) \triangleq e \notin D$, and $check(c \bullet c', D) \triangleq check(c) \bullet check(c')$ with $\bullet \in \{\wedge, \vee\}$. For instance, the prefix $e_1 \wedge e_2 \wedge \overline{e_3} \wedge \overline{e_4} \ \rightarrow \ d$ is enabled in D if $\{e_1, e_2\} \subseteq D$ and $\{e_3, e_4\} \cap D = \emptyset$ (i.e., e_1, e_2 are present and e_3, e_4 absent). In that case, the entity d is produced.

The process $\mathbf{0}$ represents the end of a context sequence. The process X represents the *call* to a recursively defined process. The process $\sum_{i \in I} p_i.K_i$ nondeterministically chooses one of the enabled prefixes p_i and precludes the others from execution. To simplify the notation: when the set of indices I is a singleton, we omit the "$\sum_{i \in I}$" part and we write $p.K$; if $|I| = 2$, we write $p_1.K_1 + p_2.K_2$.

The parallel composition of two processes is represented by $K \parallel M$, where the outputs of K and M are merged together.

The process $\mathbf{rec}\ X.K$ allows for recursive definitions. We assume that all occurrences of X in $\mathbf{rec}\ X.K$ appear in the scope of a prefix (e.g., $\mathbf{rec}\ X.p.X$ is guarded while $\mathbf{rec}\ X.X$ is not). Moreover, all the occurrences of X in a process K must be bound by the binder "$\mathbf{rec}\ X.$" (i.e., there are no free variables in a well-formed process). These conditions are usually required in process algebras to guarantee, e.g., that the resulting transition system is finitely branching.

To avoid parentheses, the convention is that $\wedge$ binds tighter than $\vee$, and, in decreasing order of precedence, we have recursion, prefix, sums and parallel composition (e.g., $\mathbf{rec}\ X.K \parallel p_1.K_1 + p_2.K_2$ must be understood as $(\mathbf{rec}\ X.K) \parallel (p_1.K_1 + p_2.K_2)$). Moreover, we write $!p$ instead of $\mathbf{rec}\ X.p.X$, i.e., the unbounded repetition of the prefix p.

Operational Semantics. Now we define the input/output behavior of **ccReact** processes when interacting with a reaction system. For that, we consider three reduction relations:

1. $p \overset{D}{\rightsquigarrow} C$, determining that the prefix p produces the output C when the current state is D;
2. $K \xrightarrow{(D,C)} K'$, determining that the context specified by the process K reduces to K' on input D. On doing that, it produces C; and
3. $(D, K) \xRightarrow{(C)} (D', K')$, capturing the interaction of K with the RS where the process K outputs C on D. After that, the new *configuration* is (D', K'), where the state of the RS is D' and the context is provided by K'.

The next definition formalizes the relations in (1) and (2) above.

Definition 5 (Semantics of processes). $\rightsquigarrow$: $\mathbf{Pref} \times S \times S$ *and* $\longrightarrow$: $\mathbf{Proc} \times S \times S \times \mathbf{Proc}$ *are the least relations satisfying, respectively, the rules* prefix *and* prefix-c*, and the rules* sum*,* $\mathsf{sum}_\emptyset$*,* parallel *and* rec*, in Figure 1.*

$$\frac{check(c,D) = true}{(c \ \rightarrow \ C) \overset{D}{\rightsquigarrow} C}\ \textsf{prefix-c} \qquad \frac{}{C \overset{D}{\rightsquigarrow} C}\ \textsf{prefix} \qquad \frac{p_j \overset{D}{\rightsquigarrow} C_j \quad \text{for some } j \in I}{\sum_I p_i.K_i \xrightarrow{(D,C_j)} K_j}\ \textsf{sum}$$

$$\frac{p_j \overset{D}{\not\rightsquigarrow} \quad \text{for all } j \in I}{\sum_I p_i.K_i \xrightarrow{(D,\emptyset)} \mathbf{0}}\ \textsf{sum}_\emptyset \qquad \frac{K \xrightarrow{(D,C_K)} K' \text{ and } M \xrightarrow{(D,C_M)} M'}{K \parallel M \xrightarrow{(D,C_K \cup C_M)} K' \parallel M'}\ \textsf{parallel}$$

$$\frac{K[\mathbf{rec}\ X.K/X] \xrightarrow{(D,C)} K'}{\mathbf{rec}\ X.K \xrightarrow{(D,C)} K'}\ \textsf{rec} \qquad \frac{K \xrightarrow{(D,C)} K' \text{ and } D' = res_{\mathcal{A}}(D \cup C)}{(D,K) \xRightarrow{(C)} (D',K')}\ \textsf{output}$$

Fig. 1: Input/output behavior of **ccReact** processes

The rule prefix produces the output C on input D when the prefix p is either C, or p is of the form $c \ \rightarrow \ C$ and c holds in D. The rule sum selects for execution one of the enabled prefixes and rules out the others. If none of the prefixes is enabled, then the empty set of entities is produced (rule $\textsf{sum}_\emptyset$). The rule parallel combines the outputs of K and M in the parallel composition $K \parallel M$. The rule rec unfolds the definition whenever the resulting process produces some output. The process $\mathbf{0}$ does not exhibit any transition (no rule for it), since it simply terminates a given process and halts the associated production of contexts.

The interaction of a process K with a reaction system $\mathcal{A}$ is a possibly infinite sequence of contexts produced by K together with the output (i.e. the result sets) generated by $\mathcal{A}$ (under the contexts produced by K).

Definition 6 (Interaction). *Let $\mathcal{A} = (S, A)$ be a RS. $\Longrightarrow \ : S \times \mathbf{Proc} \times S \times S \times \mathbf{Proc}$ is the least relation satisfying the rule output in Figure 1. A run ρ of $K = K_0$ on $\mathcal{A}$ is a sequence of the form $(D_0, K_0) \xRightarrow{(C_0)} (D_1, K_1) \xRightarrow{(C_1)} \cdots$ where $D_0 = \emptyset$. Moreover: the context sequence associated to ρ, notation $\gamma(\rho)$, is the sequence $\{C_i\}_{i \geq 0}$, the result sequence $\delta(\rho)$ is the sequence $\{D_i\}_{i \geq 0}$ and the state sequence associated to ρ is the sequence $\{C_i \cup D_i\}_{i \geq 0}$.*

Some **ccReact** processes may engage in an infinite sequence of interactions with a RS (take e.g., **rec** $X.C.X$). Given an infinite sequence s, we use $s \downarrow n$ to denote the first $n + 1$ elements of s. Given a context sequence $\gamma = \{C_i\}_{i \in [0,n]}$, we use $P(\gamma)$ to denote the **ccReact** process $C_0. \cdots .C_n.\mathbf{0}$. Note that, given an arbitrary RS $\mathcal{A}$, there exists a unique run of length n of $P(\gamma)$ on $\mathcal{A}$.

The following result shows that n-step interactive processes (Definition 3) coincide with the associated state sequences of **ccReact** processes (Definition 6). This follows directly from the definition of the rule output (that coincides with the definition of D_i in Definition 3).

Theorem 1. *Let $\mathcal{A} = (S, A)$ be a RS. **(1)** Let $\gamma = \{C_i\}_{i \in [0,n]}$ be a context sequence, and ρ the unique run of $P(\gamma)$ on $\mathcal{A}$. Then, $\pi = (\gamma(\rho), \delta(\rho))$ is an n-step interactive process in $\mathcal{A}$; and **(2)** let K be a process and ρ a run of K on $\mathcal{A}$. If ρ is finite with length n, then $\pi = (\gamma(\rho), \delta(\rho))$ is an n-step interactive process in $\mathcal{A}$. If ρ is infinite, then for all $n \geq 0$, $\pi = (\gamma(\rho) \downarrow n \ , \ \delta(\rho) \downarrow n)$ is an n-step interactive process in $\mathcal{A}$.*

4 Rewriting Logic Semantics for ccReact

This section gives a rewriting logic (RL) [30,31] semantics to **ccReact**. The proposed rewrite theory is executable in the Maude [22,24] system. This opens up the possibility of performing different analyses as shown later in Section 5. For presentation, we omit some details. The complete Maude specification can be found in the companion tool of this paper [11]. We start with a quick overview of RL and Maude (see [31,24] for further details).

Overview of Rewriting Logic and Maude. A *rewrite theory* [30] is a tuple $\mathcal{R} = (\Sigma, E, L, R)$ such that: (Σ, E) is an equational theory where Σ is a signature that declares sorts, subsorts, and function symbols; E is a set of (conditional) equations of the form $t = t'$ **if** ψ, where t and t' are terms of the same sort, and ψ is a conjunction of equations; L is a set of *labels*; and R is a set of labeled (conditional) rewrite rules of the form $l : q \longrightarrow r$ **if** ψ, where $l \in L$ is a label, q and r are terms of the same sort, and ψ is a conjunction of equations and *rewrite expressions* of the form $t \Rightarrow t'$. This latter expression is interpreted as a rewriting-based reachability goal in $\mathcal{R}$.

$T_{\Sigma,s}$ denotes the set of ground terms of sort s, and $T_\Sigma(X)_s$ denotes the set of terms of sort s over a set of sorted variables X. $T_\Sigma(X)$ and T_Σ denote all terms and ground terms, respectively. A substitution $\sigma : X \to T_\Sigma(X)$ maps each variable to a term of the same sort, and $t\sigma$ denotes the term obtained by simultaneously replacing each variable x in a term t with $\sigma(x)$.

A *one-step rewrite* $t \longrightarrow_\mathcal{R} t'$ holds if there is a rule $l : q \longrightarrow r$ **if** ψ, a subterm u of t, and a substitution σ such that $u = q\sigma$ (modulo equations), t' is the term obtained from t by replacing u with $r\sigma$, and $v\sigma = v'\sigma$ holds for each $v = v'$ in ψ. The reflexive-transitive closure of $\longrightarrow_\mathcal{R}$ is denoted as $\longrightarrow^*_\mathcal{R}$. Intuitively, terms in the equational theory (Σ, E) define system states, while the behavior of the system is given by local transitions between states described by rewrite rules.

Maude [22] is a language and tool supporting the specification and analysis of rewrite theories. A Maude module (`mod` M `is` ... `endm`) specifies a rewrite theory $\mathcal{R}$. Sorts and subsort relations are declared by the keywords `sort` and `subsort`; function symbols, or *operators*, are introduced with the `op` keyword: `op` f `:` s_1 ... s_n `->` s, where $s_1, \ldots, s_n$ are the sorts of the arguments of operator f, and s is its (value) sort. Operators can have user-definable syntax, with underbars '`_`' marking each of the argument positions (e.g., `_+_`). Some operators can have equational attributes, such as `assoc`, `comm`, and `id:` t, stating that the operator is, respectively, associative, commutative, and/or has identity element t. Equations are specified with the syntax `eq` t `=` t' or `ceq` t `=` t' `if` ψ; and rewrite rules as `rl [`l`] :` u `=>` v or `crl [`l`] :` u `=>` v `if` ψ. The mathematical variables in such statements are declared with the notations `var` and `vars` or, on the fly, with the notation `X:S` (variable `X` of sort `S`).

Maude provides a large set of analysis methods, including simulation by rewriting (`rew` t, that rewrites the term t) and explicit-state reachability analysis (`search` t `=>*` t' `such that` ψ, searching for a state reachable from t that matches the pattern t' and satisfies the condition ψ).

Reaction Systems as Maude's terms. Entities in a RS are constants of `sort Entity`. For instance, `ops egf e p erbb1 : -> Entity` declares some of the entities in Example 1. The sort `SetEntity` represents "`,`"-separated sets of entities ("`,`" is an `assoc`iative, `comm`utative operator with `empty` as `id`entity, and satisfies the equation for idempotency `eq X , X = X`). For instance, "`e , p`" is a term of sort `SetEntity`. Sets of reactions can be then specified as "`,`"-separated sets of terms of `sort Reaction`, built with the following constructor:

```
op [_;_;_] : SetEntity SetEntity SetEntity -> Reaction .
```

For instance, the reaction a_1 in Example 1 can be represented with the term `[egf ; (e , p) ; erbb1]` of sort `Reaction`.

ccReact processes are defined by the following operators:

```
sorts Condition Process PreProcess SumProcess .
subsort Qid < Process .          --- For processes variables
subsort PreProcess < SumProcess < Process .
subsort Entity < Condition .     --- An entity is a condition
op _^ : Entity -> Condition .    --- An entity that must be absent
op tt : -> Condition .           --- True condition
ops _and_ _or_ : Condition Condition -> Condition .
op {_}._  : SetEntity Process                        -> PreProcess .
op `[ _-->{_} `] . _  : Condition SetEntity Process -> PreProcess .
op 0      :                        -> Process .
op _+_    : PreProcess SumProcess -> SumProcess .
op _||_   : Process Process        -> Process [assoc comm id: 0] .
op rec_._ : Qid Process            -> Process .
```

The sort `Qid` in Maude denotes quoted identifiers as in `'X`. They are used here to denote process variables. Terms of sort `PreProcess` are *prefixed* processes of the form $p.K$ where $p \in \mathbf{Pref}$ and $K \in \mathbf{Proc}$. Summations $\sum_{i \in I} p_i.K_i$ are represented as terms of the sort `SumProcess`. For instance, the process $\mathbf{rec}\ X.(\mathsf{ERBB1}.X + (\mathsf{e} \wedge \overline{\mathsf{p}} \rightarrow \mathsf{PLCG}).X)$ that either provides as input ERBB1 or PLCG (when e is present and p absent) is represented by the following term:

```
rec 'X . (erbb1 . 'X   +   [ e and p ^  --> { plcg } ] . 'X).
```

***ccReact** Semantics in Maude.* The behavior of a **ccReact** process interacting with a RS is specified by: (1) a *deterministic* part, using equations, for checking *conditions* and computing the effects on the RS when a set of entities C is provided as input context; and (2) rewrite rules defining the non-deterministic behavior of **ccReact** processes.

The following specification defines the operator `eval`, that recursively computes (`evalRec`) the output D (a set of entities) from a set of reactions $SetR$ when the environment provides as stimulus the set of entities C:

```
vars C D R I P IN OUT : SetEntity . vars SetR : SetReaction .
op eval    : SetEntity SetReaction           -> SetEntity .
op evalRec : SetEntity SetReaction SetEntity -> SetEntity .
eq eval(C, SetR) = eval(C, SetR, empty) .
eq evalRec(C, empty, D) = D .
```

```
eq evalRec(C, ([R; I; P], SetR), D) =
   evalRec(C, SetR, (D, output(C, [R; I; P]))) .
op output : SetEntity Reaction -> SetEntity .
eq output(C, [R ; I ; P]) = --- Returns P if [R;I;P] is firable on C
 if (R subset C) and-then intersection(C,I) == empty then P else empty fi
```

As expected, the definition of `evalRec` checks every reaction against the current input C. A reaction (R, I, P) adds to the output D the products P, when the reactants R needed are a `subset` of the input C and $C \cap I = \emptyset$ (definition of `output`). In this specification, `if_then_else_fi` is the usual ternary operator.

Similarly, the operator `op check : Condition SetEntity -> Bool` is defined so that to check when a condition is true in a given state, as well as the operator `op firable : SetEntity SumProcess -> Bool` checks whether at least one of the prefixes in a summation is enabled.

States and Rules. We consider the following sorts and operators:

```
sorts PState IOState .      subsort PState IOState < State .
op { process:_ ; input:_ }  : Process SetEntity -> PState .
op { next:_; input:_; output:_ } : Process SetEntity SetEntity -> IOState
```

Terms of sort `PState` are used to capture the evolution of a process during the current interaction with the RS (relations $\rightsquigarrow$ and $\longrightarrow$). Terms of sort `IOState` specify the output of the system in the current interaction (relation $\Longrightarrow$). Note that both `PState` and `IOState` are subsorts of `State` which is defined in Maude's model-checker prelude [9] to represent the states of the transition system induced by a rewrite theory. Adding theses subsort relations allows us to seamlessly endow **ccReact** with verification capabilities (see Section 5).

A process $p.K$ (sort `PreProcess`) produces an output by using the union of the entities in the current input and those generated by the prefix p:

```
rl [prefix] : { process: ( { C } . K)  ; input: IN } =>
              { next: K ; input: IN ; output: eval((C, IN), reactions)} .
rl [prefixC] : { process: ( [ COND --> { C } ] . K)  ; input: IN } =>
   if check(COND, IN)
    then { next: K  ; input: IN ; output: eval( (C , IN), reactions) }
    else { next: 0  ; input: IN ; output: eval(      IN, reactions) } fi
```

In these rules, `reactions` is the set of reactions of the RS under consideration. The rule `prefix` produces (function `eval`) the output using the prefix C and the current input (set `(C , IN)`). In `prefixC`, if the condition *COND* holds, the output is generated using the union of the entities in the current input and C. Otherwise, the output depends only on the current input ($\mathsf{sum}_\emptyset$ in Figure 1).

In a summation $M = p.K + KS$, where $KS = \sum_{i \in I} p_i.K_i$, all the enabled (if any) prefixes must be considered. Hence, if p is enabled (i.e., `firable` on `IN`), the process $p.K$ can be scheduled for execution (rule `choiceL` below). Moreover, if there exists at least one enabled prefix in the continuation KS, KS is also considered for execution (rule `choiceR`).

```
crl [choiceL] : {process: Kp + KS ; input: IN} =>
```

```
    {process: Kp ; input: IN} if firable(IN, Kp) .
crl [choiceR] : {process: Kp + KS ; input: IN} =>
    {process: KS ; input: IN} if firable(IN, KS) .
```

The behavior of the parallel composition $K_1 \parallel K_2$ is specified by the rewrite rule `parallel` which exploits rewrite expressions in its condition to compute the outputs of both K_1 and K_2 (namely, `OUT1` and `OUT2`). Then, it combines `OUT1` and `OUT2` together by using the set union operator "`_,_`":

```
vars K1 K2 : Process . vars OUT1 OUT2 : SetEntity .
crl [parallel] : {process: K1 || K2 ; input: IN} =>
                 {next: K1' || K2' ; input: IN ; output: (OUT1 , OUT2)}
 if {process: K1; input: IN} => {next: K1'; input: IN ; output: OUT1} /\
    {process: K2 ; input: IN} => {next: K2'; input: IN ; output: OUT2} .
```

For recursive processes, the rule below unfolds the definition of the process with the aid of the function `subst` that computes the substitution $P[\mathbf{rec}\ X.P/X]$:

```
rl [rec] : { process: (rec X . K)              ; input: IN } =>
           { process: subst(X, (rec X . K), K) ; input: IN } .
```

Finally, the following rule allows us to observe the execution of the system across time-instants, where the entities produced in the current time-instant (i.e. `OUT`) are used as inputs in the next one:

```
rl [next] : {next: K; input: IN; output: OUT} => {process: K; input: OUT}
```

Adequacy. The representation of entities, reactions and processes as terms of the appropriate sorts is quite natural and allows for easily defining maps $[\![\cdot]\!]$ between RS components and their Maude counterparts, e.g., from sets of entities to terms of sort `SetEntity`, reactions to terms of sort `Reaction`, etc. In the following, given any syntactic entity ξ (entity, reaction, etc), we shall use $[\![\xi]\!]$ to denote the corresponding term of the appropriate sort.

Theorem 2 (Adequacy). *For all processes K, K' and sets of entities C, D,* $(D, K) \xRightarrow{(C)} (D', K')$ *iff* $\{\texttt{process}: [\![\texttt{K}]\!]\ ;\ \texttt{input}: [\![\texttt{D}]\!]\} \longrightarrow_{\mathcal{R}}^{*}$

$\{\texttt{next}: [\![\texttt{K}']\!]\ ;\ \texttt{input}: [\![\texttt{D}]\!]\ ;\texttt{output}: [\![\texttt{C}]\!]\} \longrightarrow_{\mathcal{R}} \{\texttt{process}: [\![\texttt{K}']\!]\ ;\ \texttt{input}: [\![\texttt{D}']\!]\}$.

5 Case Studies and Analyses

In this section we show how different analysis methods including simulation by rewriting, reachability analysis, and LTL/CTL model checking are available within our framework. We showcase the use of **ccReact** for the analysis of a RS that models the case study presented in Section 2. For this system, we verify results already found in the literature [27], and verify new hypotheses. It is worth noting that **ccReact** shows excellent performance in all the considered experiments (e.g., model-checking the properties under examination takes few milliseconds). However, further investigation is needed to benchmark system scalability on more complex RSs.

Simulation and State Reachability. Our Maude specification provides an executable model for **ccReact** that can be directly used to simulate interactive processes of RSs. This can be achieved by using the built-in Maude command `rew [n]` that performs a rewrite sequence involving `n` rewrite steps. For instance, consider the RS in Example 1. The following specification declares the entities and input context to represent this system.

```
ops egf  e  p  erbb1  erk12  p70s6k : -> Entity .
eq reactions = [(egf) ; (e ,p) ; (erbb1)] ,
   [(egf) ; ( empty ) ; (erk12)], [(erk12) ; ( empty ) ; (p70s6k)] .
op proc : -> Process . eq proc = {(egf,e)}.{empty}.{egf}.{empty}.0 .
op init : -> IOState  .
eq init = { next: proc ; input: empty  ; output: empty } .
```

Note that the **ccReact** process `proc` specifies the context sequence $\gamma = C_0, \ldots, C_3$ of the interactive process $\pi = (\gamma, \delta)$ in Example 1. The initial state `init` includes `proc`, the `input` representing D_0 (i.e., the first element of the result sequence δ) and the `output` field which is initially set to empty. Now, the following `rew` command generates a two-step rewrite sequence[5] that rewrites `init` and exactly simulates the computation of the next result set D_1 of δ. Indeed, the `output` in the resulting state is `erk12` as in Example 1.

```
Maude> rew [2] init . --- 2 steps of rewriting
result IOState: {next: {empty}. {egf}. ...; input: empty; output: erk12}
```

More interesting, we can search for all the `IOState`s reachable from `init`, thus obtaining the full result sequence $\delta = D_0, \ldots, D_3$ of π [6]:

```
Maude> search init =>* S:IOState .
Solution 1 (state 0)
S:IOState --> {next: {(egf, e)}. ... ; input: empty ; output: empty}
Solution 2 (state 2)
S:IOState --> {next: {empty}. {egf}. ... ; input: empty ; output: erk12}
...
Solution 5 (state 8)
S:IOState --> {next: 0 ; input: erbb1, erk12 ; output: p70s6k}
No more solutions.
```

The Maude search capability can be used to answer more complex reachability queries. For instance, we can fully reproduce the results in [27] (see Table 1). We start by defining a predicate `isState` to check whether the steady state has been reached. In Maude, predicates (or atomic propositions) are terms of the sort `Prop` whose meaning is defined by the operator `op _|=_ : State Prop -> Bool`, specifying when a state satisfies the proposition. We then identify steady states by inspecting the `input` and `output` fields of any `IOState` as follows:

[5] Some outputs are simplified (`...`) and comments (`---`) are added to explain the output. The complete specification and outputs are available in [11].

[6] Maude outputs an "extra" configuration D_4 (Solution 5), that corresponds to $D_4 = res_A(D_3 \cup C_3)$. "(`state n`)" is the number of the state in the search tree.

```
op isSteady : -> Prop . --- Is it a steady state?
eq { next: K ; input: IN ; output: OUT } |= isSteady   =   IN == OUT .
eq STATE:PState |= isSteady = false .
```

Note that the predicate `isState` (and similarly for other predicates introduced below) does not hold in states of sort `PState`. Recall that such states are only instrumental to decompose **ccReact** process and no output is really produced in them.

We also define the following predicates:

```
subsort SetEntity < Prop . --- Sets of entities as propositions
eq STATE:PState |= C = false .
eq { next: K ; input: IN ; output: OUT } |= C = C subset OUT .
op io-state : -> Prop .  --- Is it an IOState?
eq STATE:PState  |= io-state  = false .
eq STATE:IOState |= io-state  = true .
```

Due to the subsort relation `SetEntity < Prop` and its associated equations, a set of entities C is also an atomic proposition that holds in a `IOState` S iff C is a `subset` of the `OUT`put in S.

The results in Table 1 can be obtained by checking the presence of the attractors (e.g., AKT), when considering a **ccReact** process of the form ! (S, D). This process provides a context sequence including the stimulus S ={EGF,HRG} together with the drugs, e.g., D = {e, t}. For instance, for the BT474 cancer, the combination of erlotinib and trastuzumab is effective, whereas BT474 is resistant to the treatment with the drugs erlotinib and pertuzumab:

```
Maude> search [1] { process: ! (hrg, egf, e, t) ; input: empty } =>*
       STATE:IOState such that STATE:IOState |= (akt)  /\
                               STATE:IOState |= isSteady .
No solution.
Maude> search [1] { process: ! (hrg, egf, e, p) ...  } =>*  ...
Solution 1 (state 8)
STATE:IOState --> {... output: akt, p70s6k, mtor, plcg, erk12, pkca}
```

Model-checking. The Unified Maude Model-Checking tool (`umaudemc`)[34] is a general interface to different model checkers for Maude models. Here we show how `umaudemc` can be used to formally verify LTL as well as CTL temporal formulas against our models for HER2-positive cancers.

LTL and CTL temporal formulas are build from atomic `Proposition`, Boolean connectives and the usual temporal operators. In the case of LTL, we have $\Box$ (always), $\Diamond$ (eventually), $\circ$ (next), **U** (until) and **R** (release). These operators are quantified in CTL over the paths of the computation tree (there **E**xists or for **A**ll paths). Temporal logic formulas can then be model-checked using the command `check` of `umaudemc` whose parameters include the file with the Maude specification, the initial state (a term of sort `State`) and the temporal formula.

The results of [27] that we reproduced using the `search` commands above can alternatively be obtained via model-checking. More concretely the presence/absence of the attractor AKT in steady states of the BT747 cell line can be checked

by feeding `umaudemc` with the LTL formula $\Diamond(\Box(\texttt{io-state} \rightarrow \texttt{akt}))$ and an initial state that specifies the stimulus and drug combination to be considered:

```
$> umaudemc check BT474 "{ process: ! (hrg, egf, e, t) ; input: empty }"
   "<> [] (io-state ->  akt)" --- Stimulus: HRG+EGF Drug combination: e+t

The property is not satisfied in the initial state ... counterexample ...
```

Using temporal formulas, we can go beyond the results in [27] and test other hypotheses. For instance, we can check that it is always the case that once the growth factor receptor ErbB-2 is present in a time-instant t, then it is also present in the time-instant $t + 1$, thus highlighting a persistent activity of the ErbB-2 receptor across the pathway. We cannot model check directly the formula $\Box(\texttt{erbb2} \rightarrow \circ\texttt{erbb2})$ since the *next* state of a term of sort `IOState` is a term of sort `PState`. This can be remediated by model checking, instead, the formula $\Box(\texttt{erbb2} \rightarrow \circ(\neg\texttt{io-state}\ \mathbf{U}\ \texttt{erbb2}))$, where `erbb2` holds in the next `IOState` (after a sequence of `PState`s).

We can use CTL to check the behavior of a drug treatment on different runs. For instance, in order to observe the interactions when either erlotinib or pertuzumab are supplied, it suffices to consider the process `{(hrg,egf,e)}.(rec 'X . ({e} .'X) + ({p} .'X))`. For this process, Maude reports that the formula $\mathbf{E} \diamond (\texttt{isSteady} \wedge \neg\texttt{akt})$ is satisfied (there **E**xists at least one path where that treatment is successful) but $\mathbf{A} \diamond (\texttt{isSteady} \wedge \neg\texttt{akt})$ is not (not **A**ll paths avoid a steady state where AKT is present).

We can also check properties about the transient states, describing the evolution of the RS before reaching the steady state. Consider e.g., the process

```
! (hrg, egf)) || (rec 'X . ( ({e} . 'X) + ({ p } . 'X) + ({ t } . 'X) )))
```

Maude returns that, for BT474, the following two formulas are satisfied: $\Box(\texttt{pdk1} \rightarrow (\diamond\Box(\texttt{io-state} \rightarrow \texttt{akt})))$ and $\texttt{erbb1}\ \mathbf{R}\ \neg\texttt{pdk1}$. This (positively) answers the following questions: (i) regardless the drug used, once PDK1 is present, inevitably the steady state includes AKT; and (ii) PDK1 never appears before ErbB-1 is produced (which basically means that PDK1 is a product of the activation of the ErbB-1 receptor).

The results for the short-term experiments in [27] show that by *permanently* providing the drug erlotinib and the stimulus (EGF and HRG), the attractor AKT is never produced. By using **ccReact** conditional prefixes we can ask whether the drug needs to be provided in *all* the time-instants or can be provided only when an EGF receptor is activated. For that, consider the following process:

```
rec 'X . ([ erbb1   or  erbb2   --> { e } ]     . 'X) +
         ([ erbb1 ^ and erbb2 ^ --> { empty } ] . 'X)
```

Maude checks that the property $\mathbf{E} \diamond \mathbf{E}\Box(\texttt{io-state} \rightarrow \neg\texttt{akt})$ is satisfied, i.e., the production of AKT can be also inhibited by providing erlotinib only when receptor ErbB-1 and ErbB-2 are active.

6 Related Work

There are some frameworks implemented in Maude that can be compared to ours. [10] provides a Maude formalization of RSs which is suitable for trace slicing, a technique used to simplify computation traces. The methodology in [10] needs context sequences to be manually given by the user, whereas our approach uses processes to generate (conditional as well as nondeterministic) contexts, automatically.

Pathway Logic (PL) [35] is a Maude framework that allows signaling pathways to be simulated and formally verified. However, PL is limited to the analysis of cell models, while our framework is more abstract and can be applied to multiple application domains (even beyond the biological context).

As for the formal verification of RSs, [29] introduces a CTL logic specifically tailored to context-restricted RSs, i.e., a computation model that generalizes standard RSs by controlling their context sequences. [32] proposes an LTL logic for a version of RSs with discrete concentrations, where collections of biological entities are modeled as multisets. Differently to our work, [29,32] use ad-hoc implemented model checkers, while we verify specifications by means of the standard model checkers. In [11] we have reproduced some of the experiments in [29,32] within our framework.

Nondeterministic contexts have already appeared in the literature [28,20,21], but here we are able to represent a collection of experiments within a single semantic object, which is easier to analyze and model-check. On the other hand, the RS semantics in [20,21] is compositional and hence well-suited to modular analysis, while our semantics is not, since we handle RSs as monolithic objects.

7 Conclusions and Future Work

Modeling and analyzing biological phenomena are complex tasks which require the support of adequate languages and tools. In this paper, we presented **ccReact**, a language of processes that formally captures the dynamic behavior of RSs. **ccReact** is endowed with a rich process algebra that allows for the definition of multiple external contexts that can drive the execution of a given RS w.r.t. different environment stimuli. Our language has been implemented in Maude, a rewriting-based language that offers both fast performance and an easy interconnection with several analysis tools. To this respect, we show how Maude built-in reachability capability and the Unified Maude Model-Checking tool can be directly employed to efficiently check properties related to resistance of different combinations of therapeutics in the HER2-positive breast cancer subtype.

As future work we want to consider other case studies, such as the rare disease Alkaptonuria [17], for which we are interested to analyze the evolution of the diseases under different treatments.We want also to investigate a possible integration of static analysis and slicing techniques [15,16,6,5,3,4,14] within our framework, to study an extension to a distributed system [19], and the extension of the set of entities in the reactions with more structured values, exploiting some ideas from [1,2].

References

1. Alpuente, M., Ballis, D., Cuenca-Ortega, A., Escobar, S., Meseguer, J.: ACUOS^2: A High-Performance System for Modular ACU Generalization with Subtyping and Inheritance. In: Proceedings of the 16th European Conference on Logics in Artificial Intelligence (JELIA 2019). Lecture Notes in Computer Science, vol. 11468, pp. 171–181. Springer (2019). https://doi.org/10.1007/978-3-030-19570-0_11
2. Alpuente, M., Ballis, D., Escobar, S., Sapiña, J.: Variant-Based Equational Anti-unification. In: Proc. of Logic-Based Program Synthesis and Transformation - 32nd International Symposium, LOPSTR 2022. Lecture Notes in Computer Science, vol. 13474, pp. 44–60. Springer (2022). https://doi.org/10.1007/978-3-031-16767-6_3
3. Alpuente, M., Ballis, D., Frechina, F., Sapiña, J.: Combining Runtime Checking and Slicing to improve Maude Error Diagnosis. In: Logic, Rewriting, and Concurrency - Festschrift Symposium in Honor of José Meseguer. Lecture Notes in Computer Science, vol. 9200, pp. 72–96. Springer (2015). https://doi.org/10.1007/978-3-319-23165-5_3
4. Alpuente, M., Ballis, D., Frechina, F., Sapiña, J.: Assertion-based Analysis via Slicing with ABETS. Theory and Practice of Logic Programming **16**(5–6), 515–532 (2016). https://doi.org/10.1017/S1471068416000375
5. Alpuente, M., Ballis, D., Romero, D.: A Rewriting Logic Approach to the Formal Specification and Verification of Web Applications. Science of Computer Programming **81**, 79–107 (2014). https://doi.org/https://doi.org/10.1016/j.scico.2013.07.014
6. Alpuente, M., Ballis, D., Sapiña, J.: Static Correction of Maude Programs with Assertions. Journal of Systems and Software **153**, 64–85 (2019). https://doi.org/10.1016/j.jss.2019.03.061
7. 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
8. Azimi, S., Iancu, B., Petre, I.: Reaction system models for the heat shock response. Fundam. Informaticae **131**(3-4), 299–312 (2014). https://doi.org/10.3233/FI-2014-1016
9. Bae, K., Meseguer, J.: The linear temporal logic of rewriting maude model checker. In: Ölveczky, P.C. (ed.) Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Held as a Satellite Event of ETAPS 2010, Paphos, Cyprus, March 20-21, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6381, pp. 208–225. Springer (2010). https://doi.org/10.1007/978-3-642-16310-4_14
10. Ballis, D., Brodo, L., Falaschi, M.: Modeling and Analyzing Reaction Systems in Maude. Electronics **13**(6,1139) (2024). https://doi.org/https://doi.org/10.3390/electronics13061139
11. Ballis, D., Brodo, L., Falaschi, M., Olarte, C.: Process calculi and rewriting techniques for analyzing reaction systems. Companion tool. Available at `https://depot.lipn.univ-paris13.fr/olarte/reaction-systems-maude.git` (2024)
12. Barbuti, R., Gori, R., Levi, F., Milazzo, P.: Investigating dynamic causalities in reaction systems. Theor. Comput. Sci. **623**, 114–145 (2016). https://doi.org/https://doi.org/10.1016/j.tcs.2015.11.041
13. 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/https://doi.org/10.1016/j.tcs.2020.07.031, special Issue on Reaction Systems

14. Bodei, C., Brodo, L., Degano, P., Gao, H.: Detecting and preventing type flaws at static time. Journal of Computer Security **18**(2), 229–264 (2010)
15. Bodei, C., Brodo, L., Focardi, R.: Static evidences for attack reconstruction. In: Bodei, C., Ferrari, G., Priami, C. (eds.) Programming Languages with Applications to Biology and Security - Essays Dedicated to Pierpaolo Degano on the Occasion of His 65th Birthday. Lecture Notes in Computer Science, vol. 9465, pp. 162–182. Springer (2015). https://doi.org/10.1007/978-3-319-25527-9_12
16. 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, `https://doi.org/10.1016/j.tcs.2017.07.008`
17. Braconi, D., Millucci, L., Spiga, O., Santucci, A.: Cella and tissue models of alkaptonuria. Drug Discovery Today: Disease Models **31**, 3–10 (2020). https://doi.org/10.1016/j.ddmod.2019.12.001
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/https://doi.org/10.1142/S0129054111008842
19. Brodo, L.: On the expressiveness of the π-calculus and the mobile ambients. In: Johnson, M., Pavlovic, D. (eds.) Algebraic Methodology and Software Technology. pp. 44–59. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17796-5_3
20. Brodo, L., Bruni, R., Falaschi, M.: Enhancing reaction systems: A process algebraic approach. In: Alvim, M., Chatzikokolakis, K., Olarte, C., Valencia, F. (eds.) The Art of Modelling Computational Systems. LNCS, vol. 11760, pp. 68–85. Springer Berlin (2019). https://doi.org/10.1007/978-3-030-31175-9_5
21. Brodo, L., Bruni, R., Falaschi, M.: A logical and graphical framework for reaction systems. Theoretical Computer Science **875**, 1–27 (2021). https://doi.org/https://doi.org/10.1016/j.tcs.2021.03.024
22. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L. (eds.): All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic, Lecture Notes in Computer Science, vol. 4350. Springer (2007). https://doi.org/10.1007/978-3-540-71999-1
23. Corolli, L., Maj, C., Marinia, F., Besozzi, D., Mauri, G.: An excursion in reaction systems: From computer science to biology. Theor. Comput. Sci. **454**, 95–108 (2012). https://doi.org/https://doi.org/10.1016/j.tcs.2012.04.003
24. Durán, F., Eker, S., Escobar, S., Martí-Oliet, N., Meseguer, J., Rubio, R., Talcott, C.L.: Programming and symbolic computation in Maude. J. Log. Algebraic Methods Program. **110** (2020). https://doi.org/10.1016/J.JLAMP.2019.100497
25. 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
26. 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
27. der Heyde, S.V., Bender, C., Henjes, F., Sonntag, J., Korf, U., Beißbarth, T.: Boolean ErbB network reconstructions and perturbation simulations reveal individual drug response in different breast cancer cell lines. BMC Systems Biology **8**(1), 75 (2014). https://doi.org/10.1186/1752-0509-8-75

28. Kleijn, J., Koutny, M., Mikulski, Ł., Rozenberg, G.: Reaction systems, transition systems, and equivalences. In: Böckenhauer, H., Komm, D., Unger, W. (eds.) Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday. LNCS, vol. 11011, pp. 63–84. Springer (2018). https://doi.org/10.1007/978-3-319-98355-4_5
29. Męski, A., Penczek, W., Rozenberg, G.: Model checking temporal properties of reaction systems. Information Sciences **313**, 22–42 (2015). https://doi.org/10.1016/j.ins.2015.03.048
30. Meseguer, J.: Conditioned rewriting logic as a united model of concurrency. Theor. Comput. Sci. **96**(1), 73–155 (1992). https://doi.org/10.1016/0304-3975(92)90182-F
31. Meseguer, J.: Twenty years of rewriting logic. J. Log. Algebraic Methods Program. **81**(7-8), 721–781 (2012). https://doi.org/10.1016/J.JLAP.2012.06.003
32. Meski, A., Koutny, M., Penczek, W.: Verification of linear-time temporal properties for reaction systems with discrete concentrations. Fundam. Informaticae **154**(1-4), 289–306 (2017). https://doi.org/10.3233/FI-2017-1567
33. 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
34. Rubio, R., Martí-Oliet, N., Pita, I., Verdejo, A.: Strategies, model checking and branching-time properties in maude. J. Log. Algebraic Methods Program. **123**, 100700 (2021). https://doi.org/10.1016/J.JLAMP.2021.100700
35. Talcott, C.: Pathway Logic. In: Proceedings of the 8th International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM 2008). Lecture Notes in Computer Science, vol. 5016, pp. 21–53. Springer (2008). https://doi.org/10.1007/978-3-540-68894-5_2