Theoretical Computer Science 875 (2021) 1–27

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

A logical and graphical framework for reaction systems ☆

Linda Brodo [b,*], Roberto Bruni [a,*], Moreno Falaschi [c,*]

[a] Dipartimento di Informatica, Università di Pisa, Italy
[b] Dipartimento di Scienze Economiche e Aziendali, Università di Sassari, Italy
[c] Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Univ. di Siena, Italy

ARTICLE INFO

Article history:
Received 22 January 2021
Received in revised form 19 March 2021
Accepted 22 March 2021
Available online 26 March 2021
Communicated by I. Petre

Keywords:
SOS rules
Reaction systems
Logic programming
Assertions
Bisimulation
Hennessy-Milner logic

ABSTRACT

Reaction Systems (RSs) are a successful computational framework inspired by biological systems. A RS pairs a set of entities with a set of reactions over them. Entities can be used to enable or inhibit each reaction, and are produced by reactions. Entities can also be provided by an external context sequence to simulate *in silico* biological experiments. In this paper we define an extension of RSs considering nondeterministic and recursive context operators, and give an original labelled transition system (LTS) for extended RSs in the structural operational semantics (SOS) style. Thanks to extended contexts, a single LTS can now account for several biological experiments. The rich information recorded in transition labels is useful to guarantee the compositionality of SOS inference rules as well as to define an assertion language to tailor behavioural and logical equivalences on some specific properties or entities. The SOS rules have been also exploited to design a flexible prototype implementation in logic programming that allows to inspect the LTS and to extract useful information when performing experiments on a RS. Our implementation provides a rapid prototyping tool for (extensions of) RSs, with a user friendly online interface to our interpreter. A parser allows to introduce the logical formulas and the contexts using the usual comfortable concrete syntax. The user can visualise and inspect the LTS for a RS and make some analysis of its underlying computation patterns, can check if the main RS satisfies a given property and if it is equivalent to a second adversarial RS. Finally, the SOS approach is suited to drive additional enhancements of RSs.

1. Introduction

Labelled Transition Systems (LTSs) are a powerful structure to model the behaviour of interacting processes. An LTS can be conveniently defined following the Structural Operational Semantics (SOS) approach [1,2]. Given a signature, an SOS system assigns some inference rules to each operator of the language: the conclusion of each rule is the transition of a composite term, which is determined by those of its constituents (appearing as premises of the rule). The SOS approach has been particularly successful in the area of process algebras [3–5].

☆ Research partially supported by Università degli Studi di Sassari Project FAR_2020 *Fondi di Ateneo per la ricerca 2020*, by MIUR PRIN Project 201784YSZ5 *ASPRA: Analysis of program analyses*, and by Università di Pisa Project PRA_2018_66 *DECLWARE: Metodologie dichiarative per la progettazione e il deployment di applicazioni*.

* Corresponding author.
E-mail addresses: brodo@uniss.it (L. Brodo), bruni@di.unipi.it (R. Bruni), moreno.falaschi@unisi.it (M. Falaschi).

https://doi.org/10.1016/j.tcs.2021.03.024
0304-3975/

Reaction Systems (RSs) [6] are a computational framework inspired by systems of living cells. Its constituents are 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 presences 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 and that will be made available at the next step). The behaviour of a RS is then defined as a discrete time interactive process: a finite context sequence describes the entities provided by the environment at each step, the current state is determined by the union of the entities coming from the environment with those produced from the previous step and the state sequence is determined by applying all and only the enabled reactions to the set of entities available in the current state. Since their introduction, RSs have shown to be a quite general computational model whose application ranges from the modelling of biological phenomena [7–10], and molecular chemistry [11] to theoretical foundations of computing [12,13].

Given the context sequence, the semantics of RSs is uniquely determined and can be represented as a finite, deterministic and unlabelled transition system. When a biological system is modelled as a RS, *in silico* experiments can then be conducted by synthesizing a specific context sequence to represent the external stimuli and then observing the resulting state sequence. One limitation of such approach is the difficulty in representing 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.

Here we define, for the first time, an LTS semantics for RSs in the SOS style that is able to faithfully represent the ordinary semantics of RS as well as allowing to conduct more general experiments and thus overcome the above mentioned limitation. First we fix a process signature whose operators pinpoint the basic structure of a RS. We have operators for entities and reactions. For contexts we exploit some classic process algebraic operators (action prefix, sum and recursion). This way we can recursively define contexts that possibly exhibit nondeterministic behaviour, as sometimes have already appeared in the literature [14,15]. Even though we enrich the expressiveness of contexts, the overall LTS still remains finite.

The SOS approach has several advantages: 1) compositionality, the behaviour of each composite system is defined in term of the behaviours of its constituents; 2) each transition label conveys all the activities connected to that rewrite step; 3) the definition of contexts is better integrated in the framework; 4) different kinds of contexts (recursive, nondeterministic) can be considered, so to combine different experiments in a single structure and to account for (regular) possibly infinite computations; 5) it is now easier to change or extend the concept of RSs by adding new operators; 6) SOS rules facilitate implementation in a declarative language and the application of standard techniques for defining equivalences between processes.

The transition labels of our LTS are so rich of information that standard notion of behavioural equivalence (like traces or bisimulation) are too fine grain and would distinguish too much. For studying RSs, one is often interested in focusing on some entities and disregard others, like exploiting a microscope to enhance certain details and ignore others that fall out of the picture. To this aim, following the ideas in our previous paper [16], we propose an assertion language built over the transition labels, and we make the definition of behavioural and logical equivalences parametric w.r.t. such assertions. This way, it is possible to consider different RSs as equivalent for some purposes or to distinguish them for other purposes. Then, the results in [16] can be immediately transferred to our setting to prove the correspondence between a coinductive definition in terms of bisimilarity and its logical counterpart *à la* Hennessy-Milner.

To experiment with the theory developed in this paper, we have implemented in logic programming a first prototype interpreter of our semantic framework, which we have then extended with additional features to increase its performance, modularity and extensibility. The interpreter is written in SWI-Prolog and is freely available for download at http://pages.di.unipi.it/bruni/LTSRS/. It consists of a main file `BioReSolve.pl` (of around 2000 code lines, fully commented) that can load custom Reaction System specifications. A template for writing custom RS specifications is also available on the web site, together with a short document with usage instructions. Alternatively, the web site includes an online version of the tool based on Tau Prolog[1]: no installation is required as it can be experimented via the browser, but the code is not optimized and the underlying engine is much less efficient. Our interpreter allows the user to derive the LTS of a Reaction System as well as checking behavioural equivalences of different Reaction Systems and the validity of formulas expressed in our assertion-based variant of the Hennessy-Milner logic. The experimentation conducted with the tool is widely discussed in Section 5.

A preliminary version of this paper was presented in [17]. With respect to the workshop version, we have added full proofs of main results, added original examples and many explanations. In particular we have considered a real world example of large size, which we discuss and analyse, also introducing a new graph analysis methodology. We have improved in several ways our implementation, making it much more user friendly and providing several new functionalities, including the above mentioned online interface. We have defined a parser which allows us to introduce the logical formulas and the contexts using the usual comfortable concrete syntax. We added the generation of the graph of the LTS for a RS in a portable graph description language, called DOT format,[2] and we added a tool for checking if the main RS and a second adversarial reaction system are bisimilar.

[1] http://tau-prolog.org/.

[2] See the documentation pages of the graph visualization software Graphviz (https://graphviz.org) for information about the syntax of DOT format. Other visualization tools include Gephi https://gephi.org, Vis.js https://visjs.github.io/vis-network/examples/network/data/dotLanguage/dotEdgeStyles.html and Graphviz Visual Editor http://magjac.com/graphviz-visual-editor/.

Related work. The work by Kleijn et al. [14] presents an LTS for RS over 2^S states, where S is the set of entities. Two labelled transition system versions have been proposed: state-oblivious context controller, and state-aware context controller. In the first version, the transition labels only record the entities provided by the context, and in the second one the transition labels also provide the entities composing the actual state. The last choice allows one to decide which entities the context should provide. Differently, we give a process algebra-style definition of the RS, where the SOS rules produce informative transition labels, including context specification, allowing different kinds of analysis.

The SOS approach to RS has already been proposed in the definition of the Reaction Algebra by Pardini et al. [18], however the emphasis was there on another operator inspired by process algebras, namely (entities) hiding and the resulting LTS was essentially deterministic, so that the usual notion of bisimulation coincided with trace equivalence. In this paper, we introduce the definition of recursive and non deterministic environments that allows to set up more complex and interesting *in silico* experiments.

There are some previous works based on bisimulation applied to models for biological systems. Barbuti et al. [19] define a classical setting for bisimulation for two formalisms: the Calculus of Looping Sequences, which is a rewriting system, and the Brane Calculi, which is based on process calculi. Bisimulation is used to verify properties of the regulation of lactose degradation in Escherichia coli and the EGF signalling pathway. These calculi allow the authors to model membranes' behaviour. Cardelli et al. [20] present two quantitative behavioural equivalences over species of a chemical reaction network with semantics based on ordinary differential equations. Bisimulation identifies a partition where each equivalence class represents the exact sum of the concentrations of the species belonging to that class. Bisimulation also relates species that have identical solutions at all time points when starting from the same initial conditions. Both the mentioned formalisms [19,20] adopt a classical approach to bisimulation.

In Brodo et al. [15,16] we derived a similar LTS to the one presented here by encoding RSs into cCNA, a more general multi-party process algebra (a variant of the `link`-calculus [21,22]). In comparison with the encoding of RS in cCNA, here we give an SOS semantics tailored for RSs, without relying on an ad hoc translation. Consequently, any term of our process algebra corresponds to a Reaction System, which is not the case for cCNA: for example, a cCNA process representing a Reaction System can exhibit a totally unexpected behaviour when it is composed with other generic cCNA processes, moreover, even if the fragment of cCNA processes that are images of some Reaction System is closed under transitions, when inspecting the states of the LTS it is much more difficult to recover the corresponding states of the original Reaction System. Another difference is that the synchronization algebra of transition labels in cCNA relies on particular sequences of symbols, called link chains, that required the introduction of several different symbols for each entity and reaction. Each symbol represents a way in which the entity was used (as a reactant, as an inhibitor, as a product, as a context provision, etc.) or which reaction was considered, while the labels of the LTS in this paper are much more abstract (tuples with four sets of entities). For these reasons, it would be notably more difficult to develop an implementation or a graphical tool to directly analyze RS computations according to the encoding in [15,16]. Overall, the advantage is that we get here a much simpler computational model, closely related to the syntax of RSs, easier to analyze, to implement and to possibly extend with new operators, enhancing the expressivity of RSs. Still, the contribution in [16] has been fundamental for the design of the assertion language and the parametric notions of behavioural and logical equivalences presented here, namely the definition of bio-similarity and BioHML, and the fact that we have been able to transfer them to the setting of this paper is a witness of their generality.

Structure of the paper. In Section 2 we recall the basics of RSs. The original contribution starts from Section 3, where: 1) we introduce the syntax and operational semantics of a novel process algebra for RSs, 2) we show how to encode RSs as processes, 3) we state a tight correspondence between the classical semantics of RSs and the operational semantics of their corresponding processes. Section 4 shows the correspondence between a coinductive definition in terms of bisimilarity and its logical counterpart *à la* Hennessy-Milner. A prototype implementation in logic programming of our semantic framework is described in Section 5, together with a discussion on related implementation tools. Further extensions of RSs that build on our theory are sketched in Section 6. Some concluding remarks are in Section 7.

2. Reaction systems

The theory of Reaction Systems (RSs) [6] was born in the field of Natural Computing to model the behaviour of biochemical reactions in living cells.

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).* 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, non empty sets and $R \cap I = \emptyset$.

The sets R, I, P are the sets of *reactants*, *inhibitors*, and *products*, respectively. All reactants are needed for the reaction to take place. Any inhibitor blocks the reaction. Products are the outcome of the reaction. Since R and I are not empty,

all products are produced from at least one reactant and every reaction can be inhibited. We let $rac(S)$ be the set of all reactions in S.

Definition 2 *(Reaction System).* A Reaction System (RS) is a pair $\mathcal{A} = (S, A)$ s.t. S is a finite set, and $A \subseteq rac(S)$ is a finite set of reactions in S.

The theory of RSs is based on three assumptions: **no permanency**, any entity vanishes unless it is sustained by a reaction. In fact, a living cell would die for lack of energy, without chemical reactions; **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 taken into account; **threshold nature of resources**, we assume that either an entity is available for all reactions, or it is not available at all.

Definition 3 *(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 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: $res_A(W) \triangleq \bigcup_{a \in A} res_a(W)$.

Living cells are seen as open systems that react with the external environment. The behaviour of a RS is formalized in terms of *interactive processes*.

Definition 4 *(Interactive Process).* Let $\mathcal{A} = (S, A)$ be a RS and let $n \geq 0$. An n-steps *interactive process* in $\mathcal{A}$ is a pair $\pi = (\gamma, \delta)$ 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. The result sequence δ is entirely determined by γ and A. Each state W_i in τ is the union of two sets: the context C_i at step i and the result set $D_i = res_A(W_{i-1})$ from the previous step.

Given a context sequence γ, we denote by γ^k the shift of γ starting at the k-th step. The shift notation will come in handy to draw a tight correspondence between the classic semantics of RS and the newly proposed SOS specification.

Definition 5 *(Sequence shift).* Let $\gamma = \{C_i\}_{i \in [0,n]}$ a context sequence. Given a positive integer $k \leq n$ we let $\gamma^k = \{C_{i+k}\}_{i \in [0, n-k]}$.

We conclude this section with a simple example of RS.

Example 6. Here we consider a toy RS defined as $\mathcal{A} = (S, A)$ where the set $S = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ only contains three entities, and the set of reactions $A = \{a_1\}$ only contains the reaction $a_1 = (\{\mathsf{a}, \mathsf{b}\}, \{\mathsf{c}\}, \{\mathsf{b}\})$, to be written more concisely as $(\mathsf{ab}, \mathsf{c}, \mathsf{b})$. Then, we consider a $4-steps$ interactive process $\pi = (\gamma, \delta)$, where $\gamma = \{C_0, C_1, C_2, C_3\}$, with $C_0 = \{\mathsf{a}, \mathsf{b}\}$, $C_1 = \{\mathsf{a}\}$, $C_2 = \{\mathsf{c}\}$, and $C_3 = \{\mathsf{c}\}$; and $\delta = \{D_0, D_1, D_2, D_3\}$, with $D_0 = \emptyset$, $D_1 = \{\mathsf{b}\}$, $D_2 = \{\mathsf{b}\}$, and $D_3 = \emptyset$. Then, the resulting state sequence is

$$\tau = W_0, W_1, W_2, W_3 = \{\mathsf{a}, \mathsf{b}\}, \{\mathsf{a}, \mathsf{b}\}, \{\mathsf{b}, \mathsf{c}\}, \{\mathsf{c}\}.$$

In fact, it is easy to check that, e.g., $W_0 = C_0$, $D_1 = res_A(W_0) = res_A(\{\mathsf{a}, \mathsf{b}\}) = \{\mathsf{b}\}$ because $en_a(W_0)$, and $W_1 = C_1 \cup D_1 = \{\mathsf{a}\} \cup \{\mathsf{b}\} = \{\mathsf{a}, \mathsf{b}\}$.

3. SOS rules for reaction systems

Inspired by classic process algebras, such as CCS [3], we introduce a syntax for RSs that resembles their original presentation and then equip each operator with some SOS inference rules that define its behaviour. This way: (1) we establish a strong correspondence between terms of the signature and RSs; (2) we derive an LTS semantics for each RS, where the states are terms, each transition corresponds to a step of the RS and transition labels retain some information needed for compositionality; (3) we pave the way to the RS enhancements in Section 6.

Definition 7 *(RS processes).* Let S be a set of entities. An *RS process* P is any term defined by the following grammar:

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

where $R, I, P \subseteq S$ are non empty sets of entities, $C, D \subseteq S$ are possibly empty set of entities, and X is a process variable.

An RS process P embeds a *mixture* process M obtained as the parallel composition of some reactions (R, I, P), some set of currently present entities D (possibly the empty set $\emptyset$), and some *context* process K. We write $\prod_{i\in I} \mathsf{M}_i$ for the parallel composition of all M_i with $i \in I$. For example, $\prod_{i\in\{1,2\}} \mathsf{M}_i = \mathsf{M}_1 \mid \mathsf{M}_2$.

A process context K is a possibly nondeterministic and recursive system: the nil context $\mathbf{0}$ stops the computation; the prefixed context $C.\mathsf{K}$ says that the entities in C are immediately available to be consumed by the reactions, and then K is the context offered at the next step; the non deterministic choice $\mathsf{K}_1 + \mathsf{K}_2$ allows the context to behave either as K_1 or K_2; X is a process variable, and rec $X.\ \mathsf{K}$ is the usual recursive operator of process algebras. We write $\sum_{i\in I} \mathsf{K}_i$ for the nondeterministic choice between all K_i with $i \in I$.

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 $\emptyset$ 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 $D_1, D_2 \subseteq S$.

Remark 8. Note that the processes $\emptyset$ and $\mathbf{0}$ are not interchangeable: as it will become clear from the operational semantics, the process $\emptyset$ can perform just a trivial transition to itself, while the process $\mathbf{0}$ cannot perform any transition.

Definition 9 *(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:

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

where the context process $\mathsf{K}_{\gamma^i} \triangleq C_i.C_{i+1}.\cdots.C_n.\mathbf{0}$ is the sequentialization of the entities offered by γ^i. We write $[\![\mathcal{A}, \pi]\!]$ as a shorthand for $[\![\mathcal{A}, \pi]\!]_0$.

Example 10. Here, we give the encoding of the reaction system, $\mathcal{A} = (S, A)$, defined in Example 6. The resulting RS process is as follows:

$$\mathsf{P} = [\![\mathcal{A}, \pi]\!] = [\![(\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}, \{(\mathsf{ab}, \mathsf{c}, \mathsf{b})\}), \pi]\!] = [(\mathsf{ab}, \mathsf{c}, \mathsf{b}) \mid \emptyset \mid \mathsf{K}_\gamma] \equiv [(\mathsf{ab}, \mathsf{c}, \mathsf{b}) \mid \mathsf{K}_\gamma]$$

where $\mathsf{K}_\gamma = \{\mathsf{a}, \mathsf{b}\}.\{\mathsf{a}\}.\{\mathsf{c}\}.\{\mathsf{c}\}.\mathbf{0}$, written more concisely as $\mathsf{ab.a.c.c.}\mathbf{0}$. Note that $\emptyset$ is inessential and can be discarded thanks to structural congruence.

In Definition 9 we have not exploited the entire potentialities of the syntax. In particular, the context K_γ is just a finite sequence of action prefixes induced by the set of entities provided by γ at the various steps. Our syntax allows for more general kinds of contexts as shown in the example below. Nondeterministic contexts can be used to collect several experiments, while recursion can be exploited to extract some regularity in the longterm behaviour of a Reaction System. Together they offer any combination of in-breadth/in-depth analysis.

Example 11. Let us consider our running example. Suppose we want to enhance the behaviour of the context by defining a process $\mathsf{K}' = \mathsf{K}_1 + \mathsf{K}_2$ that non-deterministically can behave as K_1 or as K_2, where $\mathsf{K}_1 = \mathsf{ab.a.c.c.}\mathbf{0}$ (as in Example 10), and $\mathsf{K}_2 = \text{rec } X.\ \mathsf{ab.a.}X$ (which is a recursive behaviour that allows the reaction to be always enabled). Then we simply define $\mathsf{P}' \equiv [(\mathsf{ab}, \mathsf{c}, \mathsf{b}) \mid \mathsf{K}']$.

Definition 12 *(Label).* A label is a tuple $\langle W \rhd R, I, P\rangle$ with $W, R, I, P \subseteq S$.

In a transition label $\langle W \rhd R, I, P\rangle$, we record the set W of entities currently in the system (produced in the previous step or provided by the context), the set R of entities whose presence is assumed (either because they are needed as reactants on an applied reaction or because their presence prevents the application of some reaction); the set I of entities whose absence is assumed (either because they appear as inhibitors for an applied reaction or because their absence prevents the application of some reaction); the set P of products of all the applied reactions.

Definition 13 *(Operational semantics).* The operational semantics of processes is defined by the set of SOS inference rules in Fig. 1.

The process $\mathbf{0}$ has no transition. The rule (*Ent*) makes available the entities in the (possibly empty) set D, then reduces to $\emptyset$. As a special instance of (*Ent*), $\emptyset \xrightarrow{\langle\emptyset\rhd\emptyset,\emptyset,\emptyset\rangle} \emptyset$. The rule (*Cxt*) says that a prefixed context process $C.\mathsf{K}$ makes available the

$$\frac{}{D \xrightarrow{\langle D \rhd \emptyset,\emptyset,\emptyset\rangle} \emptyset}\ (Ent) \qquad \frac{}{C.\mathsf{K} \xrightarrow{\langle C \rhd \emptyset,\emptyset,\emptyset\rangle} \mathsf{K}}\ (Cxt) \qquad \frac{\mathsf{K}[^{\mathsf{rec}\ X.\ \mathsf{K}}/_X] \xrightarrow{\langle W \rhd R,I,P\rangle} \mathsf{K}'}{\mathsf{rec}\ X.\ \mathsf{K} \xrightarrow{\langle W \rhd R,I,P\rangle} \mathsf{K}'}\ (Rec)$$

$$\frac{\mathsf{K}_1 \xrightarrow{\langle W \rhd R,I,P\rangle} \mathsf{K}_1'}{\mathsf{K}_1 + \mathsf{K}_2 \xrightarrow{\langle W \rhd R,I,P\rangle} \mathsf{K}_1'}\ (Suml) \qquad \frac{\mathsf{K}_2 \xrightarrow{\langle W \rhd R,I,P\rangle} \mathsf{K}_2'}{\mathsf{K}_1 + \mathsf{K}_2 \xrightarrow{\langle W \rhd R,I,P\rangle} \mathsf{K}_2'}\ (Sumr)$$

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

$$\frac{\mathsf{M}_1 \xrightarrow{\langle W_1 \rhd R_1,I_1,P_1\rangle} \mathsf{M}_1' \quad \mathsf{M}_2 \xrightarrow{\langle W_2 \rhd R_2,I_2,P_2\rangle} \mathsf{M}_2' \quad (W_1 \cup W_2 \cup R_1 \cup R_2) \cap (I_1 \cup I_2) = \emptyset}{\mathsf{M}_1 \mid \mathsf{M}_2 \xrightarrow{\langle W_1 \cup W_2 \rhd R_1 \cup R_2,I_1 \cup I_2,P_1 \cup P_2\rangle} \mathsf{M}_1' \mid \mathsf{M}_2'}\ (Par)$$

$$\frac{\mathsf{M} \xrightarrow{\langle W \rhd R,I,P\rangle} \mathsf{M}' \quad R \subseteq W}{[\mathsf{M}] \xrightarrow{\langle W \rhd R,I,P\rangle} [\mathsf{M}']}\ (Sys)$$

Fig. 1. SOS semantics of the reaction system processes.

entities in the set C and then reduces to K. The rule (*Rec*) is the classical rule for recursion. Here, $\mathsf{K}[^{\mathsf{rec}\ X.\ \mathsf{K}}/_X]$ denotes the process obtained by replacing in K every free occurrence of the variable X with its recursive definition $\mathsf{rec}\ X.\ \mathsf{K}$. For example $\mathsf{rec}\ X.\ \mathsf{a.b}.X \xrightarrow{\langle \mathsf{a} \rhd \emptyset,\emptyset,\emptyset\rangle} \mathsf{b.rec}\ X.\ \mathsf{a.b}.X$ The rules (*Suml*) and (*Sumr*) select a move of either the left or the right component, resp., discarding the other process. The rule (*Pro*), executes the reaction (R, I, P) (its reactants, inhibitors, and products are recorded the label), which remains available at the next step together with P. The rule (*Inh*) applies when the reaction (R, I, P) should not be executed; it records in the label the possible causes for which the reaction is disabled: possibly some inhibiting entities ($J \subseteq I$) are present or some reactants ($Q \subseteq R$) are missing, with $J \cup Q \neq \emptyset$, as at least one cause is needed for explaining why the reaction is not enabled.[3] The rule (*Par*) puts two processes in parallel by pooling their labels and joining all the set components of the labels; a sanity check is required to guarantee that there is no conflict between reactants and inhibitors of the applied reactions. Finally, the rule (*Sys*) requires that all the processes of the systems have been considered, and also checks that all the needed reactants are actually available in the system ($R \subseteq W$). In fact this constraint can only be met on top of all processes. The check that inhibitors are absent ($I \cap W = \emptyset$) is not necessary, as it is embedded in rule (*Par*).

Example 14. Let us consider the RS process $\mathsf{P}_0 \triangleq [(\mathsf{ab}, \mathsf{c}, \mathsf{b}) \mid \mathsf{ab.a.c.c.}\mathbf{0}]$ from Example 10. The process P_0 has a unique outgoing transition, whose formal derivation is given below:

$$\dfrac{\dfrac{\dfrac{}{(\mathsf{ab},\mathsf{c},\mathsf{b}) \xrightarrow{\langle \emptyset \rhd \mathsf{ab},\mathsf{c},\mathsf{b}\rangle} (\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{b}}\ (Pro) \qquad \dfrac{}{\mathsf{ab.a.c.c.}\mathbf{0} \xrightarrow{\langle \mathsf{ab} \rhd \emptyset,\emptyset,\emptyset\rangle} \mathsf{a.c.c.}\mathbf{0}}\ (Cxt)}{(\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{ab.a.c.c.}\mathbf{0} \xrightarrow{\langle \mathsf{ab} \rhd \mathsf{ab},\mathsf{c},\mathsf{b}\rangle} (\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{b} \mid \mathsf{a.c.c.}\mathbf{0}}\ (Par)}{[(\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{ab.a.c.c.}\mathbf{0}] \xrightarrow{\langle \mathsf{ab} \rhd \mathsf{ab},\mathsf{c},\mathsf{b}\rangle} [(\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{b} \mid \mathsf{a.c.c.}\mathbf{0}]}\ (Sys)$$

The target process $\mathsf{P}_1 \triangleq [(\mathsf{ab}, \mathsf{c}, \mathsf{b}) \mid \mathsf{b} \mid \mathsf{a.c.c.}\mathbf{0}]$ has also a unique outgoing transition, namely:

$$\mathsf{P}_1 = [(\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{b} \mid \mathsf{a.c.c.}\mathbf{0}] \xrightarrow{\langle \mathsf{ab} \rhd \mathsf{ab},\mathsf{c},\mathsf{b}\rangle} [(\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{b} \mid \mathsf{c.c.}\mathbf{0}] = \mathsf{P}_2$$

Instead the process P_2 has three outgoing transitions, each providing a different justification to the fact that the reaction $(\mathsf{ab}, \mathsf{c}, \mathsf{b})$ is not enabled:

1. $[(\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{b} \mid \mathsf{c.c.}\mathbf{0}] \xrightarrow{\langle \mathsf{bc} \rhd \mathsf{c},\mathsf{a},\emptyset\rangle} [(\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{c.}\mathbf{0}]$, where the label shows that the presence of c and the absence of a inhibit the reaction;
2. $[(\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{b} \mid \mathsf{c.c.}\mathbf{0}] \xrightarrow{\langle \mathsf{bc} \rhd \mathsf{c},\emptyset,\emptyset\rangle} [(\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{c.}\mathbf{0}]$, where it is only observed that the presence of c has played some role in inhibiting the reaction;
3. $[(\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{b} \mid \mathsf{c.c.}\mathbf{0}] \xrightarrow{\langle \mathsf{bc} \rhd \emptyset,\mathsf{a},\emptyset\rangle} [(\mathsf{ab},\mathsf{c},\mathsf{b}) \mid \mathsf{c.}\mathbf{0}]$, where it is only observed that the absence of a has played some role in inhibiting the reaction.

[3] Conceptually, one could extend labels to record J and Q in separate positions from R and I, respectively, like in $\langle W \rhd R, J, I, Q, P\rangle$. However, one would then need to rewrite the side conditions of all the rules by replacing R with $R \cup J$ and I with $I \cup Q$, because the distinction is never exploited in the SOS rules.

Notably, the three transitions have the same target process $\mathsf{P}_3 \triangleq [(\mathsf{ab}, \mathsf{c}, \mathsf{b}) \mid \mathsf{c}.\mathbf{0}]$.

Finally, the process P_3 has seven transitions all leading to $\mathsf{P}_4 \triangleq [(\mathsf{ab}, \mathsf{c}, \mathsf{b}) \mid \mathbf{0}]$. Their labels are of the form $\langle c \rhd J, Q, \emptyset \rangle$ with $J \subseteq \mathsf{c}$, $Q \subseteq \mathsf{ab}$ and $J \cup Q \neq \emptyset$. Each label provides a different explanation why the reaction is not enabled.

The following technical lemmas express some relevant properties of the transition system and can be proved by rule induction (see the Appendix).

Lemma 15. *If* $\mathsf{M} \xrightarrow{\langle W \rhd R, I, P \rangle} \mathsf{M}'$ *then* $\mathsf{M}' \equiv \mathsf{M}'' \mid P$ *for some* M''.

Lemma 16. *If* $\prod_{a \in A} a \xrightarrow{\langle W \rhd R, I, P \rangle} \mathsf{M}$ *then* $W = \emptyset$ *and* $\mathsf{M} \equiv \prod_{a \in A} a \mid P$.

Lemma 17. *If* $\mathsf{M} \xrightarrow{\langle W \rhd R, I, P \rangle} \mathsf{M}'$ *then* $(W \cup R) \cap I = \emptyset$.

Lemma 18. *If* $\mathsf{P} \xrightarrow{\langle W \rhd R, I, P \rangle} \mathsf{P}'$ *then* $R \subseteq W$ *and* $W \cap I = \emptyset$.

The main theorem shows that the rewrite steps of a RS exactly match the transitions of its corresponding RS process. For the proof see the appendix.

Theorem 19. *Let* $\mathcal{A} = (S, A)$ *be a RS, and* $\pi = (\gamma, \delta)$ *an* n*-step interactive process in* $\mathcal{A}$ *with* $\gamma = \{C_i\}_{i \in [0,n]}$, $\delta = \{D_i\}_{i \in [0,n]}$, *and let* $W_i \triangleq C_i \cup D_i$ *and* $\mathsf{P}_i \triangleq [\![\mathcal{A}, \pi]\!]_i$ *for any* $i \in [0, n]$. *Then:*

1. $\forall i \in [0, n-1]$, $\mathsf{P}_i \xrightarrow{\langle W \rhd R, I, P \rangle} \mathsf{P}$ *implies* $W = W_i$, $P = D_{i+1}$ *and* $\mathsf{P} \equiv \mathsf{P}_{i+1}$;
2. $\forall i \in [0, n-1]$, *there exists* $R, I \subseteq S$ *such that* $\mathsf{P}_i \xrightarrow{\langle W_i \rhd R, I, D_{i+1} \rangle} \mathsf{P}_{i+1}$.

Remark 20. Note that the process $\mathsf{P}_n = [\![\mathcal{A}, \pi]\!]_n = [\prod_{a \in A} a \mid D_n \mid C_n.\mathbf{0}]$ has one more transition available (the $(n+1)$-th step from P_0), even if the standard theory of RSs stops the computation after n steps. We thus have additional steps

$$\mathsf{P}_n \xrightarrow{\langle W_n \rhd R_n, I_n, res_A(W_n) \rangle} \left[\prod_{a \in A} a \mid res_A(W_n) \mid \mathbf{0} \right]$$

for suitable $R_n, I_n \subseteq S$. The target process contains $\mathbf{0}$ and therefore is deadlock.

Example 14 shows that we can have redundant transitions because of rule (*Inh*). However, they can be easily detected and eliminated by considering a notion of dominance. To this aim we introduce an order relation $\sqsubseteq$ over pairs of set of entities defined as follows:

$$(R', I') \sqsubseteq (R, I) \quad \text{if} \quad R' \subseteq R \wedge I' \subseteq I.$$

As usual, we write $(R', I') \sqsubset (R, I)$ if $(R', I') \sqsubseteq (R, I)$ but $(R', I') \neq (R, I)$.

Definition 21 *(Dominance).* A transition $\mathsf{P} \xrightarrow{\langle W \rhd R', I', P \rangle} \mathsf{P}'$ is *dominated* if there exists another transition $\mathsf{P} \xrightarrow{\langle W \rhd R, I, P \rangle} \mathsf{P}'$ such that $(R', I') \sqsubset (R, I)$.

Note that in the definition of dominance we require the dominated transition to have the same source and target processes as the dominant transition, and that their labels carry also the same sets W and P.

Finally, we can immediately derive an LTS, whose transitions are written using double arrows, where only dominant transitions are considered. The LTS is defined by the additional SOS rule (*Dom*) below:

$$\frac{\mathsf{P} \xrightarrow{\langle W \rhd R, I, P \rangle} \mathsf{P}' \quad (R, I) = \max_{\sqsubseteq} \{(R', I') \mid \mathsf{P} \xrightarrow{\langle W \rhd R', I', P \rangle} \mathsf{P}'\}}{\mathsf{P} \xRightarrow{\langle W \rhd R, I, P \rangle} \mathsf{P}'} \ (Dom)$$

In other words, a transition $\mathsf{P} \xRightarrow{\langle W \rhd R, I, P \rangle} \mathsf{P}'$ guarantees that any instance of the rule (*Inh*) is applied in a way that maximizes the sets J and Q (given the overall available entities W).

Example 22. Looking back at Example 14, both transitions $\mathsf{P}_2 \xrightarrow{\langle \mathsf{bc} \rhd \mathsf{c}, \emptyset, \emptyset \rangle} \mathsf{P}_3$ and $\mathsf{P}_2 \xrightarrow{\langle \mathsf{bc} \rhd \emptyset, \mathsf{a}, \emptyset \rangle} \mathsf{P}_3$ are dominated by $\mathsf{P}_2 \xrightarrow{\langle \mathsf{bc} \rhd \mathsf{c}, \mathsf{a}, \emptyset \rangle} \mathsf{P}_3$. Therefore, the process $\mathsf{P}_2 = [(\mathsf{ab}, \mathsf{c}, \mathsf{b}) \mid \mathsf{b} \mid \mathsf{c}.\mathsf{c}.\mathbf{0}]$ has a unique (double-arrow) transition $\mathsf{P}_2 \xRightarrow{\langle \mathsf{bc} \rhd \mathsf{c}, \mathsf{a}, \emptyset \rangle} \mathsf{P}_3$.

4. Bio-simulation

Bisimulation equivalences [23] play a central role in process algebras. They can be defined in terms of coinductive games, of fixpoint theory and of logics. The bisimulation game is played by an attacker and a defender: the former wants to disprove the equivalence between two processes p and q, the latter tries to show that p and q are equivalent. The game is turn based: at each turn the attacker picks one process, e.g., p, and one transition $p \xrightarrow{\lambda} p'$ and the defender must reply by picking one transition $q \xrightarrow{\lambda} q'$ of the other process with exactly the same label λ; then the game continues challenging the equivalence between p' and q'. The game ends when the attacker has no transition available, and the defender wins, or when defender cannot match the move of the attacker, and the attacker wins. The defender also wins if the game doesn't end. The processes p and q are not equivalent iff the attacker has a winning strategy.

In the case of biological systems, the classical notion of bisimulation can be too concrete. In fact, in a biological soup, a high number of interactions occur every time instant, and generally, biologists are only interested to analyse a small subset of them and to focus on a subset of entities. In the case of RS processes, the labels that we used for the LTS consider too many details and convey too much information: they record the entire information about all the reactions that have been applied in one transition, the entities that acted as reactants, as inhibitors or as products, or that were available in the state. All this information stored in the label is necessary to compose a transition in a modular way. Depending on the application, only a suitable abstraction over the label can be of interest. For this reason, following the approach introduced in Brodo et al. [16], we exploit an alternative notion of bisimulation, called *bio-simulation*, that compares two biological systems by restricting the observation to only a limited set of events that are of particular interest. With respect to the work in Brodo et al. [16], here the labels are easier to manage and simpler to parse, because the underlying process algebra is tailored to Reaction Systems (whereas in [16] Reaction Systems were encoded in a fragment of a much more general process algebra).

In a way, at each step of the bisimulation game, we want to query our labels about some partial information. To this goal, we define an assertion language to express detailed and partial queries about what happened in a single transition.

Example 23. For instance we would like to express properties about each step of the bio-simulation of a system like the ones below:

1. Has the presence of the entity a been exploited by some reaction?
2. Have the entities a and b been produced by some reaction?
3. Have the entities a or c been provided by the state?
4. Has the reaction $(\mathsf{ab}, \mathsf{c}, \mathsf{b})$ been applied or not?

As detailed before, we remark the importance of dealing with non-deterministic contexts, as bisimulation takes into account the branching structure of system dynamics.

The bio-simulation approach works as follows: first we introduce an assertion language to abstract away some information from the labels; then we define a bisimilarity equivalence that is parametric to a given assertion, called bio-similarity; finally we give a logical characterisation of bio-similarity, called bio-logical equivalence, by tailoring the classical HML to the given assertion.

4.1. Assertion language

An assertion is a formula that predicates on the labels of our LTS. The assertion language that we propose is very basic, but can be extended if necessary.

Definition 24 *(Assertion Language).* Given a set of entities S, assertions F on S are built from the following syntax, where $E \subseteq S$ and $Pos \in \{\mathcal{W}, \mathcal{R}, \mathcal{I}, \mathcal{P}\}$:

$$\mathsf{F} ::= E \subseteq Pos \mid ? \in Pos \mid \mathsf{F} \vee \mathsf{F} \mid \mathsf{F} \wedge \mathsf{F} \mid \neg \mathsf{F}$$

Roughly, *Pos* distinguishes different positions in the labels: $\mathcal{W}$ stands for entities provided by current state, $\mathcal{R}$ stands for reactants, $\mathcal{I}$ stands for inhibitors, and $\mathcal{P}$ stands for products. An assertion F is either the membership of a subset of entities E in a given position *Pos*, $E \subseteq Pos$, the test of *Pos* for non-emptyness, $? \in Pos$, the disjunction of two assertions $\mathsf{F}_1 \vee \mathsf{F}_2$, their conjunction $\mathsf{F}_1 \wedge \mathsf{F}_2$, or the negation of an assertion $\neg\mathsf{F}$. To improve readability, we assume that negation binds stronger than conjunction and disjunction, so that $\neg\mathsf{F}_1 \wedge \mathsf{F}_2$ stands for $(\neg\mathsf{F}_1) \wedge \mathsf{F}_2$. Of course, all the remaining usual logical operators can be derived as expected, e.g. we write exclusive or $\mathsf{F}_1 \veebar \mathsf{F}_2$ as a shorthand for the assertion $(\mathsf{F}_1 \wedge \neg\mathsf{F}_2) \vee (\neg\mathsf{F}_1 \wedge \mathsf{F}_2)$ and the implication $\mathsf{F}_1 \rightarrow \mathsf{F}_2$ as a shorthand for the assertion $\neg\mathsf{F}_1 \vee \mathsf{F}_2$.

Definition 25 *(Satisfaction of Assertion).* Let $\upsilon = \langle W \rhd R, I, P \rangle$ be a transition label, and F be an assertion. We write $\upsilon \models \mathsf{F}$ (read as the transition label υ satisfies the assertion F) if and only if the following hold:

$$
\begin{array}{lll}
\upsilon \models E \subseteq Pos & \text{iff} & E \subseteq \mathsf{select}(\upsilon, Pos) \\
\upsilon \models ? \in Pos & \text{iff} & \mathsf{select}(\upsilon, Pos) \neq \emptyset \\
\upsilon \models \mathsf{F}_1 \wedge \mathsf{F}_2 & \text{iff} & \upsilon \models \mathsf{F}_1 \wedge \upsilon \models \mathsf{F}_2 \\
\upsilon \models \mathsf{F}_1 \vee \mathsf{F}_2 & \text{iff} & \upsilon \models \mathsf{F}_1 \vee \upsilon \models \mathsf{F}_2 \\
\upsilon \models \neg \mathsf{F} & \text{iff} & \upsilon \not\models \mathsf{F}
\end{array}
$$

$$
\text{where} \qquad \mathsf{select}(\langle W \rhd R, I, P\rangle, Pos) \triangleq \begin{cases} W & \text{if } Pos = \mathcal{W} \\ R & \text{if } Pos = \mathcal{R} \\ I & \text{if } Pos = \mathcal{I} \\ P & \text{if } Pos = \mathcal{P} \end{cases}
$$

Given two transition labels v, w we write $v \equiv_{\mathsf{F}} w$ if $v \models \mathsf{F} \Leftrightarrow w \models \mathsf{F}$, i.e. if both v, w satisfy F or they both do not.

Example 26. Some assertions matching the queries listed in Example 23 are:

1. $\mathsf{F}_1 \triangleq \mathsf{a} \subseteq \mathcal{R}$
2. $\mathsf{F}_2 \triangleq \mathsf{ab} \subseteq \mathcal{P}$
3. $\mathsf{F}_3 \triangleq \mathsf{a} \subseteq \mathcal{W} \vee \mathsf{c} \subseteq \mathcal{W}$
4. $\mathsf{F}_4 \triangleq \mathsf{ab} \subseteq \mathcal{R} \wedge \mathsf{c} \subseteq \mathcal{I}$ checks if the reaction has been applied, while $\mathsf{F}_5 \triangleq \mathsf{a} \subseteq \mathcal{I} \vee \mathsf{b} \subseteq \mathcal{I} \vee \mathsf{c} \subseteq \mathcal{R}$ the opposite case. Alternatively, we can set $\mathsf{F}_5 \triangleq \neg \mathsf{F}_4$.

If we take the label $\upsilon = \langle \mathsf{ab} \rhd \mathsf{ab}, \mathsf{c}, \mathsf{b}\rangle$ it is immediate to check that

$$
\upsilon \models \mathsf{F}_1 \qquad \upsilon \not\models \mathsf{F}_2 \qquad \upsilon \models \mathsf{F}_3 \qquad \upsilon \models \mathsf{F}_4 \qquad \upsilon \not\models \mathsf{F}_5
$$

With respect to the assertion language proposed in our previous paper [16], the new one has less expressive power as it is not possible to immediately distinguish the reactants, the inhibitors and the products referred to each reaction applied, or to know the reason why a reaction has not been applied. However, these informations can be retrieved by the reaction definition. The main interest of this proposal is that it is directly applied to the LTS tailored for RSs.

4.2. Bio-similarity and bio-logical equivalence

The notion of bio-simulation builds on the above language of assertions to parameterize the induced equivalence on the property of interest. Please recall that we have defined the behaviour of the context in a non deterministic way, thus at each step, different possible sets of entities can be provided to the system and different sets of reaction can be enabled/disabled. Bio-simulation can thus be used to compare the behaviour of different systems that share some of the reactions or entities or also to compare the behaviour of the same set of reaction rules when different contexts are provided.

Definition 27 *(Bio-similarity $\sim_{\mathsf{F}}$ [16]).* Given an assertion F, a *bio-simulation* $\mathbf{R}_{\mathsf{F}}$ that respects F is a binary relation over RS processes s.t., if $\mathsf{P}\ \mathbf{R}_{\mathsf{F}}\ \mathsf{Q}$ then:

- $\forall \upsilon, \mathsf{P}'$ s.t. $\mathsf{P} \stackrel{\upsilon}{\Rightarrow} \mathsf{P}'$, $\exists w, \mathsf{Q}'$ s.t. $\mathsf{Q} \stackrel{w}{\Rightarrow} \mathsf{Q}'$ with $\upsilon \equiv_{\mathsf{F}} w$ and $\mathsf{P}' \mathbf{R}_{\mathsf{F}} \mathsf{Q}'$.
- $\forall w, \mathsf{Q}'$ s.t. $\mathsf{Q} \stackrel{w}{\Rightarrow} \mathsf{Q}'$, $\exists \upsilon, \mathsf{P}'$ s.t. $\mathsf{P} \stackrel{\upsilon}{\Rightarrow} \mathsf{P}'$ with $\upsilon \equiv_{\mathsf{F}} w$ and $\mathsf{P}' \mathbf{R}_{\mathsf{F}} \mathsf{Q}'$.

We let $\sim_{\mathsf{F}}$ denote the largest bio-simulation and we say that P is *bio-similar* to Q, with respect to F, if $\mathsf{P} \sim_{\mathsf{F}} \mathsf{Q}$.

Remark 28. An alternative way to look at a bio-simulation that respects F is to define it as an ordinary bisimulation over the transition system labelled over $\{\mathsf{F}, \neg\mathsf{F}\}$ obtained by transforming each transition $\mathsf{P} \stackrel{\upsilon}{\Rightarrow} \mathsf{P}'$ such that $\upsilon \models \mathsf{F}$ into $\mathsf{P} \stackrel{\mathsf{F}}{\Rightarrow} \mathsf{P}'$ and each transition $\mathsf{P} \stackrel{\upsilon}{\Rightarrow} \mathsf{P}'$ such that $\upsilon \not\models \mathsf{F}$ into $\mathsf{P} \stackrel{\neg\mathsf{F}}{\Longrightarrow} \mathsf{P}'$.

It can be easily shown that the identity relation is a bio-simulation and that bio-simulations are closed under (relational) inverse, composition and union and that, as a consequence, bio-similarity is an equivalence relation.

Example 29. Let us consider some variants of our working example. The behaviour of $\mathsf{P}_0 \triangleq [(\mathsf{ab}, \mathsf{c}, \mathsf{b}) \mid \mathsf{ab.a.ac.}\mathbf{0}]$ is deterministic, and its unique trace of labels is:

$$
\mathsf{P}_0 \xrightarrow{\langle \mathsf{ab} \rhd \mathsf{ab}, \mathsf{c}, \mathsf{b}\rangle} \mathsf{P}_1 \xrightarrow{\langle \mathsf{ab} \rhd \mathsf{ab}, \mathsf{c}, \mathsf{b}\rangle} \mathsf{P}_2 \xrightarrow{\langle \mathsf{abc} \rhd \mathsf{c}, \emptyset, \emptyset\rangle} [(\mathsf{ab}, \mathsf{c}, \mathsf{b}) | \mathbf{0}]
$$

Instead, the behaviour of $\mathsf{P}'_0 \triangleq [(\mathsf{ab}, \mathsf{c}, \mathsf{b}) \mid (\mathsf{ab.a.ac.}\mathbf{0} + \mathsf{ab.a.a.}\mathbf{0})]$ is non deterministic. Now there are two possible traces of labels: the first trace is equal to the above one, and the other one follows:

$$\begin{array}{l} P_0' \xrightarrow{\langle ab \triangleright ab, c, b\rangle} P_1 \xrightarrow{\langle ab \triangleright ab, c, b\rangle} P_2 \xrightarrow{\langle abc \triangleright c, \emptyset, \emptyset\rangle} [(ab, c, b)|\mathbf{0}] \\ \quad\quad \xrightarrow{\langle ab \triangleright ab, c, b\rangle} P_1' \xrightarrow{\langle ab \triangleright ab, c, b\rangle} P_2' \xrightarrow{\langle ab \triangleright ab, c, b\rangle} [(ab, c, b)|b|\mathbf{0}] \end{array}$$

Now, it is easy to check that the two processes P_0, P_0' are not bio-similar w.r.t. the assertion $F_1 \triangleq c \in \mathcal{E}$, requiring that in the state configuration entity c is present, and are bio-similar w.r.t. the assertion $F_2 \triangleq (a \in \mathcal{R}) \barwedge (c \in \mathcal{R})$, requiring that either c or a are used as reactants.

Now, we introduce a slightly modified version of the Hennessy-Milner Logic [24], called bioHML; due to the reasons we explained above, we do not want to look at the complete transition labels, thus we rely on our simple assertion language to make it parametric to the assertion F of interest:

Definition 30 *(BioHML [16]).* Let F be an assertion, then the set of bioHML formulas G that respects F are built by the following syntax, where $\chi \in \{F, \neg F\}$:

$$G, H ::= \mathtt{t} \mid \mathtt{f} \mid G \wedge G \mid G \vee G \mid \langle\chi\rangle G \mid [\chi]G$$

Remark 31. An alternative way to look at bioHML formulas is as ordinary HML formulas over the set of labels $\{F, \neg F\}$.

The semantics of a bioHML formula is the set of processes that satisfy it.

Definition 32 *(Semantics of BioHML).* Let $\mathbb{P}$ denote the set of all RS processes over S. For a BioHML formula G, we define $[\![G]\!] \subseteq \mathbb{P}$ inductively on G:

$$\begin{array}{llll} [\![\mathtt{t}]\!] & \triangleq \mathbb{P} & [\![\mathtt{f}]\!] & \triangleq \emptyset \\ [\![G \wedge H]\!] & \triangleq [\![G]\!] \cap [\![H]\!] & [\![G \vee H]\!] & \triangleq [\![G]\!] \cup [\![H]\!] \end{array}$$

$$[\![\langle\chi\rangle G]\!] \triangleq \{P \in \mathbb{P} : \exists \upsilon, P'.\ P \overset{\upsilon}{\Rightarrow} P' \text{ with } \upsilon \models \chi \text{ and } P' \in [\![G]\!]\}$$
$$[\![[\chi]G]\!] \triangleq \{P \in \mathbb{P} : \forall \upsilon, P'.\ P \overset{\upsilon}{\Rightarrow} P' \text{ implies } \upsilon \models \chi \text{ and } P' \in [\![G]\!]\}$$

We write $P \models G$ (P satisfies G) if $P \in [\![G]\!]$.

Negation is not included in the syntax, but the converse $\overline{G}$ of a bioHML formula G can be easily defined inductively in the same way as for HML logic.

We let $\mathcal{L}_F$ be the set of all bioHML formulas that respects F.

Definition 33 *(Bio-logical equivalence).* We say that P, Q are *bio-logically* equivalent w.r.t. F, written $P \equiv_{\mathcal{L}_F} Q$, when P and Q satisfy the exactly the same bioHML formulas in $\mathcal{L}_F$, i.e. when for any $G \in \mathcal{L}_F$ we have $P \models G \Leftrightarrow Q \models G$.

Finally, we extend the classical result establishing the correspondence between the logical equivalence induced by HML with bisimilarity for proving that bio-similarity coincides with bio-logical equivalence. The proof is essentially the same as the one in [16] and thus omitted.

Theorem 34 *(Correspondence [16]).* $\sim_F = \equiv_{\mathcal{L}_F}$

Example 35. We continue by considering our running example in Example 29. There already is the evidence that the two processes $P_0 \triangleq [(ab, c, b) \mid ab.a.ac.\mathbf{0}]$ and $P_0' \triangleq [(ab, c, b) \mid (ab.a.ac.\mathbf{0} + ab.a.a.\mathbf{0})]$ are not bio-similar w.r.t. the assertion $F_1 \triangleq c \in \mathcal{W}$. Here, we give a bioHML formula that distinguishes P_0 and P_0':

$$G \triangleq \langle\neg F_1\rangle[\neg F_1]\langle\neg F_1\rangle\mathtt{t}.$$

In fact, G is not satisfied by P_0, written $P_0 \not\models G$, because, along the unique possible path, the labels of the first two transitions satisfy $\neg F_1$ but P_2 cannot perform any transition whose label satisfies $\neg F_1$.

Differently, $P_0' \models G$. In fact, P_0' can move to P_1' with a transition whose label satisfies $\neg F_1$, then P_1' has a unique transition to P_2' whose label satisfies $\neg F_1$ and finally the target state P_2' can perform a transition whose label satisfies $\neg F_1$.

5. Implementation and experimentation

In Falaschi and Palma [25] we have presented some preliminary work on how to implement RS formalism in a logic programming language (Prolog). Our implementation did not aim to be highly performing. We aimed to obtain a rapid prototyping tool for implementing extensions of RSs. Our initial prototype allowed to perform finite computations on RSs, in the form of interactive processes. Here we have extended the implementation by including the more general notion of contexts, and have exploited transition labels to derive the corresponding LTSs. Then we have added the predicates for formulating expressions of our assertion language that acts on the transition labels. On the basis of this assertion language we have implemented a slightly modified version of the Hennessy-Milner Logic to make it parametric on the specific assertion specified by the user. As explained in the Introduction, our interpreter is available for download,[4] together with a template for writing RS specifications and usage instructions.

5.1. Tool description

The interpreter has been developed and tested under SWI-Prolog[5] and makes use of a few library predicates for handling efficiently association lists and ordered sets. DCG Grammar rules are used to ease the writing of custom RS specifications.

In the current version of our implementation an RS is represented by a term `sys(Delta,E,Ks,Rs)`, where `Delta` is the environment, which represents a set of constant definitions for defining the recursive contexts. It is implemented as an association list (constant - context process). `E` is the current (ordered) set of entities, `Ks` is the list of context processes and `Rs` is (ordered) set of reactions. All background entities and constants are represented by different atoms (actually, they can only include letters, digits and underscore and must begin with a small cap letter). Exploiting DCG clauses, the concrete syntax for expressing contexts, environments, assertions and BioHML formulas is completely similar to the one presented in this paper: dedicated predicates perform the parsing analysis and translate it to the abstract syntax for execution in the interpreter.

The following predicates can be used to define a custom RS of interest:
`myentities/1`, `myreactions/1`, `mycontext/1`, `myenvironment/1`.

Normally, the predicate `myentities/1` defines the empty list of reactants, as the context sequence will provide other reactants, including the initial ones. However, the predicate can be updated to define a (non empty) list of reactants, like in `myentities([a1,...,an])`, from which the computation of the RS will start. The list is transformed to an ordered set before being used by any other predicate.

The predicate `myreactions/1` defines the list of reactions. A reaction in the list must be a term of the form `react(R,I,P)`, like in `myreactions([react(R1,I1,P1),...,react(Rn,In,Pn)])`. Each term `react(Ri,Ii,Pi)` contains a list of reactants `Ri`, a list of inhibitors `Ii` and a list of products `Pi`. Each list is transformed to an ordered set before being used by any other predicate.

The predicate `mycontext/1` defines the list of context processes in a suitable DCG grammar. Instead of using the rec X. K construct, recursion is made possible by relying on a finite set of context process constants to be defined in the environment.

The predicate `myenvironment/1` defines the list of context processes constants that can be exploited in the context. Thanks to DCG clauses, each constant declaration is written as `X = K`, where `X` is the atom representing the constant and `K` is the corresponding context process.

The predicate `mybhml/1` is used to define a BioHML formula G and to check if it is satisfied by the main Reaction System RS.

Finally, the predicate `myassert/1`, is used to define a default assertion F and the predicates `adventities/1`, `advreactions/1`, `advcontext/1` to define an adversary Reaction System ARS and to check if RS and ARS are F-biosimilar.

For performance reasons and in conformance with the double-arrow transition system, our implementation uses the (*InH*) rule in a *deterministic* way by maximising the sets of present inhibitors and lacking reactants in the current computation. This improves the efficiency of the tool.

Even if we prefer to leave out most details about the full list of currently available functionalities, we mention that the labels of the LTS generated from a RS specification carry even more information than those reported in Section 3. In particular, a transition label of the form $\langle W \rhd R, I, P\rangle$ is represented as a term `obs(E,C,W,R,RI,I,IR,P)`, where `E` is the set of entities available in the system before the transition, `C` is the set of entities provided by the context for that transition, `W` is just the union of `E` and `C` (it is included to avoid the need of computing it every time) and coincides with the set W of the shorter label, `R` is the set of reactants exploited by enabled reactions, `RI` is the set of reactants whose presence inhibited some reaction, so that R is the union of `R` and `RI`, similarly `I` is the set of entities whose absence enabled some reactions and `IR` is the set of entities whose absence inhibited some reactions, so that I is the union of `I` and `IR`, finally `P` coincides with P and is the set of entities produced by some reactions. Note that `E` and `C` are not necessarily disjoint, as well as `R` and `RI` and also `I` and `IR`.

[4] http://pages.di.unipi.it/bruni/LTSRS/.
[5] https://www.swi-prolog.org/.

Most functionalities are made easily available for experimentation by dedicated options of the `main/2` predicate, which can be invoked using the syntax:

```
?- main(option,Time).
```

Any option is just a constant. Among the available options, we cite just a few[6]: `stat` computes some general information about the RS; `target` computes the terminal result set of the RS; `run` computes the result sequence of the RS; `rundigraph` draws the result sequence as an LTS; `digraph` computes the whole LTS of the RS; `biohml` checks if the RS satisfies a BioHML formula; `biosim` checks biosimilarity of a RS and its adversary.

The Prolog interpreter will respond the query with

```
Time = <execution time>
```

and will save the result in a newly created file of the workspace.

As explained in the online instructions, the tool can be easily customised by instantiating the above mentioned few predicates providing, respectively, the RS specification and a BioHML formula to be verified. The graphical interface for using our interpreter online allows to select several facilities. We have: 1) the visualisation of the graph of the LTS in DOT format, using Vis.js; 2) a tool which checks whether a BioHML formula is true w.r.t. the given main RS process; 3) a tool for checking if the main RS and a second adversarial RS are bisimilar; and 4) the visualisation as a graph of the LTS corresponding to the adversarial RS.

We have run and checked all the examples in this paper, by using our interpreter, together with other examples from the literature, such as the 'heat-shock-response' presented in [7]. Due to space limitation, we report here a toy example about RS specification of Non-deterministic Finite Automata (NFA) and a much larger and challenging case study about the RS translation of the model of ErbB receptor signal transduction in human mammary epithelial cells in [26], with 6720 reactions over 246 entities and a 1000-steps interactive process.

5.2. RSs and NFA

We recall that a NFA [27,28] is a tuple $N = (Q, \Sigma, \delta, q_0, F)$ where Q is the finite set of states, Σ is a finite alphabet of symbols, $\delta : Q \times \Sigma \to \wp(Q)$ is the transition function that, given the current state of the automaton and the observed alphabet symbol, returns the (possibly empty) set of target states, q_0 is the initial state and $F \subseteq Q$ is the set of final states. Without loss of generality, we assume that the set of states S is disjoint from the set Σ of alphabet symbols.

The transition function δ can be extended to $\widehat{\delta} : Q \times \Sigma^* \to \wp(Q)$ that, given the current state of the automaton and a finite string of alphabet symbols, returns the (possibly empty) set of target states. This way the formal language accepted by the automaton N is defined as $L(N) = \{w \mid \widehat{\delta}(q_0, w) \cap F \neq \emptyset\}$, i.e. is the set of strings w such that $\widehat{\delta}(q_0, w)$ contains some final state of the automaton. It is a standard result of Computer Science that NFAs recognise regular languages and that, using the *subset construction* algorithm (also called *Rabin-Scott powerset construction*), each NFA can be translated to a Deterministic Finite Automaton (DFA) that recognises the same formal language.

We show that, by a straight transformation of the NFA to a RS we can define a process system with a standardised context process such that the corresponding LTS shown by our interpreter coincides with the version of the powerset construction that creates only the states that are actually reachable (instead of the whole powerset $\wp(S)$).

The idea is transform the NFA $N = (Q, \Sigma, \delta, q_0, F)$ to the RS $\mathcal{A}_N = (S_N, A_N)$ where[7]:

- $S_N = Q \cup \Sigma \cup \{\text{void}\}$;
- $A_N = \{(\{q, a\}, \{\text{void}\}, \delta(q, a)) \mid q \in Q, a \in \Sigma, \delta(q, a) \neq \emptyset\}$.

Then, we define a simple, recursive (perpetual) context process definition

$$x = a_1.x + \cdots + a_n.x$$

that nondeterministically can provide, at any step, a singleton context for any alphabet symbol in $\Sigma = \{a_1, ..., a_n\}$. Finally, we set the initial set of entities to $\{q_0\}$ and the list of context processes to $[x]$.

Exploiting the facilities of our tool, we can easily customise the appearance of the LTS, by deciding which labels display for the nodes and for the transitions. In the case of the powerset construction, we decide to display just the set of current entities for each node (not the context, which will always be x) and the set of entities provided by the context for the transitions. Additionally, we exploit the predicate `nodeStyle/3` to set a different colour for the states that contain an entity in the set F of final states of the original NFA.

As a concrete example, we take the well-known NFA over the binary alphabet $\Sigma = \{a, b\}$ that has $n + 1$ states and for which there is no equivalent DFA with less than s^n states (see Fig. 2, for the automaton with $n = 3$).

[6] Options `target`, `run` and `rundigraph` all require a terminating context.

[7] We insert a distinct entity void as a dummy inhibitor in reactions, which will never be provided by the context.

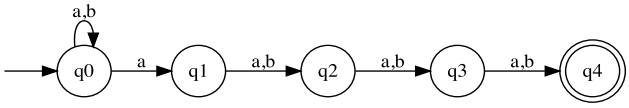

Fig. 2. A simple NFA.

```
% contents of the file spec-NFA.pl
myentities([q0]).
myreactions([react([q0,a],[void],[q0]),
             react([q0,b],[void],[q0]),
             react([q0,a],[void],[q1]),
             react([q1,a],[void],[q2]),
             react([q1,b],[void],[q2]),
             react([q2,a],[void],[q3]),
             react([q2,b],[void],[q3]),
             react([q3,a],[void],[q4]),
             react([q3,b],[void],[q4]),
             react([q4,a],[void],[q4]),
             react([q4,b],[void],[q4]) ]).
mycontext("[x]").
myenvironment("[x=({a}.x + {b}.x)]").
% plus some default trivial definitions of other predicates
% ...
```

Fig. 3. RS specification of the NFA in Fig. 2.

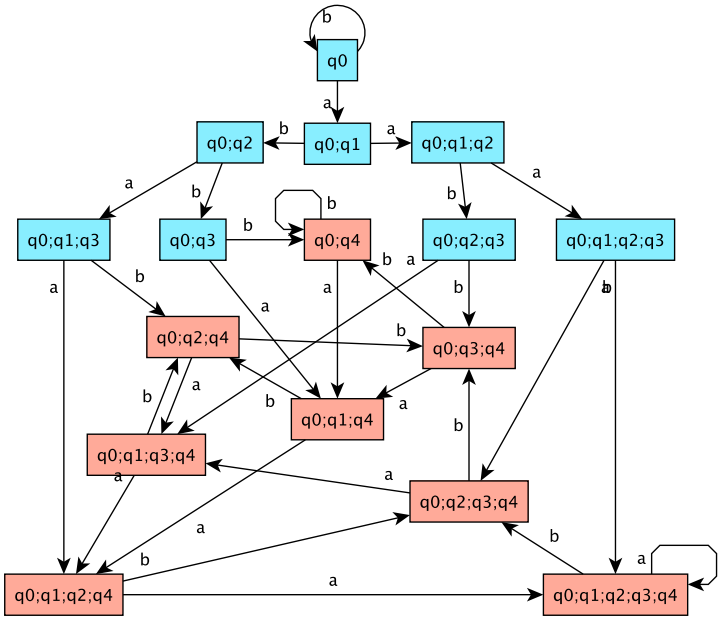

Fig. 4. DFA powerset construction as LTS of a RS. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

The corresponding RS specification is shown in Fig. 3.

Then, the query below produces the LTS in Fig. 4,[8] that corresponds to the (reachable fragment of) the DFA obtained as powerset construction. Note that final states are represented using a different colour.

```
?- main(digraph,Time).
```

[8] The layout of all graphs shown in the paper has been generated exploiting the graph editor yEd (https://www.yworks.com/products/yed), after translating the files in DOT format, as generated by our tool, in Graphml format thanks to the Python script `dottoxml.py` available at https://github.com/dirkbaechle/dottoxml.

```
PLA2, , AA
ECM Gas Gbg_i Integrins, , AC
IQGAP1 Myosin, Arp_23 alpha_catenin, Actin
IQGAP1 Myosin alpha_catenin, Arp_23, Actin
Arp_23 Myosin alpha_catenin, IQGAP1, Actin
Arp_23 IQGAP1 Myosin, alpha_catenin, Actin
Arp_23 IQGAP1 Myosin alpha_catenin, , Actin
...
Cdc42 Crk Fak Grb2 Nck PIP2_45 Src, PTPPEST, WASP

---

Ras AA Erk PIP2_34 PIP3_345 Fak Src RKIP PAK Mekk1 PP2A PA EGFR_Contr IL1_TNF alpha_sL Stress alpha_qL alpha_1213L ECM ExtPump alpha:iL EGF
Ras AA Erk PIP2_34 PIP3_345 Fak Src RKIP PAK Mekk1 PP2A PA EGFR_Contr IL1_TNF alpha_sL Stress alpha_qL alpha_1213L ECM ExtPump alpha:iL EGF
Ras AA Erk PIP2_34 PIP3_345 Fak Src RKIP PAK Mekk1 PP2A PA EGFR_Contr IL1_TNF alpha_sL Stress alpha_qL alpha_1213L ECM ExtPump alpha:iL EGF
...
Ras AA Erk PIP2_34 PIP3_345 Fak Src RKIP PAK Mekk1 PP2A PA EGFR_Contr
```

Fig. 5. An excerpt from file `erbb.rsy`.

```
% contents of the file spec-ERBB.pl
myreactions([react([diov],[void],[diov]),
          react([pla2],[void],[aa]),
          react([ecm,gas,gbg_i,integrins],[void],[ac]),
          react([iqgap1,myosin],[arp_23,alpha_catenin],[actin]),
          react([iqgap1,myosin,alpha_catenin],[arp_23],[actin]),
          react([arp_23,myosin,alpha_catenin],[iqgap1],[actin]),
          react([arp_23,iqgap1,myosin],[alpha_catenin],[actin]),
          react([arp_23,iqgap1,myosin,alpha_catenin],[void],[actin]),
          ...
          react([cdc42,crk,fak,grb2,nck,pip2_45,src],[ptppest],[wasp]) ]).

mycontext("[{ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,il1_tnf,alpha_sl,stress,alpha_ql,alpha_12131,ecm,extpump,alpha_il,egf}.
{ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,il1_tnf,alpha_sl,stress,alpha_ql,alpha_12131,ecm,extpump,alpha_il,egf}.
{ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,il1_tnf,alpha_sl,stress,alpha_ql,alpha_12131,ecm,extpump,alpha_il,egf}.
...
{ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr}.nil]").

...
```

Fig. 6. An excerpt from file `spec-ERBB.pl`.

5.3. ErbB receptor signal transduction in human mammary epithelial cells

As we wanted to challenge our experimentation on a large benchmark, we looked for RS specifications over the web and came across a very interesting model[9] available on the GitHub page of HERESY (Highly Efficient REaction SYstem simulator) [26]. HERESY is a powerful Python tool with a friendly user interface and it is able to leverage the GPU to offload reactions' calculations (cf. Section 5.4 for more details).

The file `erbb.rsy` contains 6.720 reactions and the sequence of 1000 contexts used to replicate one of the experiments in [29] (see Table 1, and more precisely the column addressing the conditions for the experiment in Fig. 2b). Of course the definition of such a large model for a real case study required a tremendous effort as well as domain-specific knowledge, so we have been very glad to find it available for experimentation.

The first step was to translate the specification in the file `erbb.rsy` to make it compatible with our Prolog predicates: in HERESY, reactions are entered as single lines containing the sets of reactants, inhibitors, and products (the elements in the same set are separated by spaces, while the three sets are separated by commas) while the set of chemicals for each iteration of the context sequence is entered as a single line separated by spaces. Reactions and contexts were separated by `---`. To give an idea of the correspondence, we report some samples from the original specification (see Fig. 5 and the online supplementary material of [30] for the input conditions of the experiment) and its translation to our setting (see Fig. 6). For example, note that we added dummy entities `diov` and `void`, together with the (always enabled) reaction `react([diov],[void],[diov])` to respect the constraint about non-empty set of reactants and inhibitors in the reactions.

By comparing the context sequence and the corresponding experiment in [29,30], we realised that the sequence was constructed according to the (%ON) activity levels of each stimulus. For example, as the activity level of the stimulus `alpha_12131` was 72%, the corresponding entity was present in the first 720 lines of the sequence. This way the shape of the RS experiment becomes quite regular, which can have important consequences for the behavioural analysis, as we are going to show later. The complete description of the different sets of entities provided by the context sequence is reported in Fig. 7. In the following, we use the term *stage* to refer to the provision of a specific context and the term *duration* to denote the number of steps during which each context is provided: so the experiment is divided in nine stages, the first stage has duration 20, the second 10, and so on.

Because we are storing a lot of information in the LTS transition labels, our first attempt to simulate the execution of the sequence and build the corresponding LTS encountered some difficulties with the default memory allocation of the stack and we needed to extend it. Then we decided to do a more in-depth inspection of the experiment in order to see if it was possible to simplify it and discovered that, given the context sequence under consideration, there were

[9] The file `erbb.rsy` at https://github.com/aresio/HERESY/tree/master/models.

```
[ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump,alpha_il,egf,il1_tnf,stress] % first 20 steps
[ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump,alpha_il,egf]               % next  80 steps
[ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump,alpha_il]                % next  10 steps
[ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump]                         % next  10 steps
[ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql]                                 % next 250 steps
[ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl]                                          % next 110 steps
[ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131]                                                   % next 240 steps
[ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm]                                                               % next 160 steps
[ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr]                                                                   % last 120 steps
```

Fig. 7. A concise description of the context sequence.

```
Some statistics about your custom RS:
============================================================
the initial state has 1 entities:
[diov]

the reactants are 211:
[aa,ac,actin,ag,akt,alix,alpha_12131,alpha_1213r,alpha_catenin,alpha_il,alpha_ir,alpha_ql,alpha_qr,alpha_sl,alpha_sr,amsh,and_34,ap2,arf,arno, ... ,wasp]

the inhibitors are 199:
[aa,ac,actin,akt,alix,alpha_12131,alpha_1213r,alpha_catenin,alpha_il,alpha_ir,alpha_ql,alpha_qr,alpha_sl,alpha_sr,and_34,ap2,arf,arno,arp_23, ... ,vps4]

the products are 210:
[aa,ac,actin,akt,alix,alpha_1213r,alpha_catenin,alpha_ir,alpha_qr,alpha_sr,amsh,and_34,ap2,arf,arno,arp_23,ask1,b_arrestin,b_parvin,ca,calm,cam, ... ,wasp]

the reactions involve 248 entities:
[aa,ac,actin,ag,akt,alix,alpha_12131,alpha_1213r,alpha_catenin,alpha_il,alpha_ir,alpha_ql,alpha_qr,alpha_sl,alpha_sr,amsh,and_34,ap2,arf,arno, ... ,wasp]

the environment involves 22 entities:
[aa,alpha_12131,alpha_il,alpha_ql,alpha_sl,ecm,egf,egfr_contr,erk,extpump,fak,il1_tnf,mekk1,pa,pak,pip2_34,pip3_345,pp2a,ras,rkip,src,stress]

the context involves 22 entities:
[aa,alpha_12131,alpha_il,alpha_ql,alpha_sl,ecm,egf,egfr_contr,erk,extpump,fak,il1_tnf,mekk1,pa,pak,pip2_34,pip3_345,pp2a,ras,rkip,src,stress]

the whole RS involves 248 entities:
[aa,ac,actin,ag,akt,alix,alpha_12131,alpha_1213r,alpha_catenin,alpha_il,alpha_ir,alpha_ql,alpha_qr,alpha_sl,alpha_sr,amsh,and_34,ap2,arf,arno, ... ,wasp]

there are 24 reactants that will never be available:
[ag,b_catenin,cdc42,dgk,erbb2_contr,erbb2deg_contr,erbb3_contr,erbb4_contr,exte_cadherin,gab1,mekk2,mkk3,nck,nrg,p120_catenin,pertuzumab,pi3k,plc_g,ptp1b,
 rac,shc,spry2,tgfa,trastuzumab]

the context can provide 0 entities that will never be used:
[]

the overall number of reactions is 6721:
- the applicable reactions are 4847
- the are 1874 reactions that will never be enabled
============================================================
```

Fig. 8. Automatically generated statistics about the ERBB specification.

many reactions that would never be enabled (because some of their reactants were appearing neither as products of other reactions nor in the context sequence). To automate the simplification on the RS specification in such cases, we developed suitable predicates to discover dead reactions and eliminate them before running the experiment. A report about the RS with some useful information of this kind can now be automatically generated by running the query below (see Fig. 8 for the ErbB specific output). We remark that 28% of the reactions were not applicable and that by discarding them before running the experiment it was no longer necessary to extend the stack size.

```
?- main(stat,Time).
```

The first experiment we made was to compute the target state. This is achieved by running the query:

```
?- main(target,Time).
```

and the result (computed in 186203 ms[10]) is:

```
[aa,akt,alpha_1213r,alpha_ir,alpha_qr,alpha_sr,and_34,ap2,arf,arno,b_parvin,ca,calm,camk,camkk,camp,cas,cbp,clathrin,cortactin,crk,csk,dag,diov,dock180,dynamin,
 eea1,egfr_t669,egfr_y1045,egfr_y1068,egfr_y1086,egfr_y1101,egfr_y1148,egfr_y1173,egfr_y891,egfr_y992,endophilin,epsin,fak,gbg_1213,gbg_s,graf,grb2,hip1r,hsc70,
 ilk,ip3,ip3r1,iqgap1,mek,mekk1,mkps,myosin,p115rhogef,p190rhogap,p90rsk,pa,pdk1,pi5k,pip2_45,pip3_345,pip_4,pix_cool,pkc_primed,pla2,rab7,rabaptin_5,rabenosyn_5,
 rabex_5,raf,ral,ralbp1,rap1,rgs,rin,sos,tiam,vinc]
```

More interestingly, we can generate the LTS of the underlying computation by running the (optimised for deterministic, terminating computations) query.

```
?- main(rundigraph,Time).
```

The corresponding graph is in Fig. 9: it consists of 129 nodes and (of course) 1000 edges and was computed in 125851 ms. We decided to use trivial labels for nodes (hiding the current entities and the process context) and to enumerate the transitions progressively to show the progress of the computation (instead of showing, e.g., the entities provided by the context at each step as was done for the NFA example). The `nodeStyle/3` predicate was used to set different colours for each node, depending on the available context (the target state is coloured in white because it has the `nil` context).

[10] All the experiments were conducted on a MacBook Pro with a 2,6 GHz Intel Core i7 6 core processor and 16 GB 2400 MHz DDR4 memory, running MacOS Catalina (v.10.15.7) and SWI-Prolog (threaded, 64 bits, version 8.0.0).

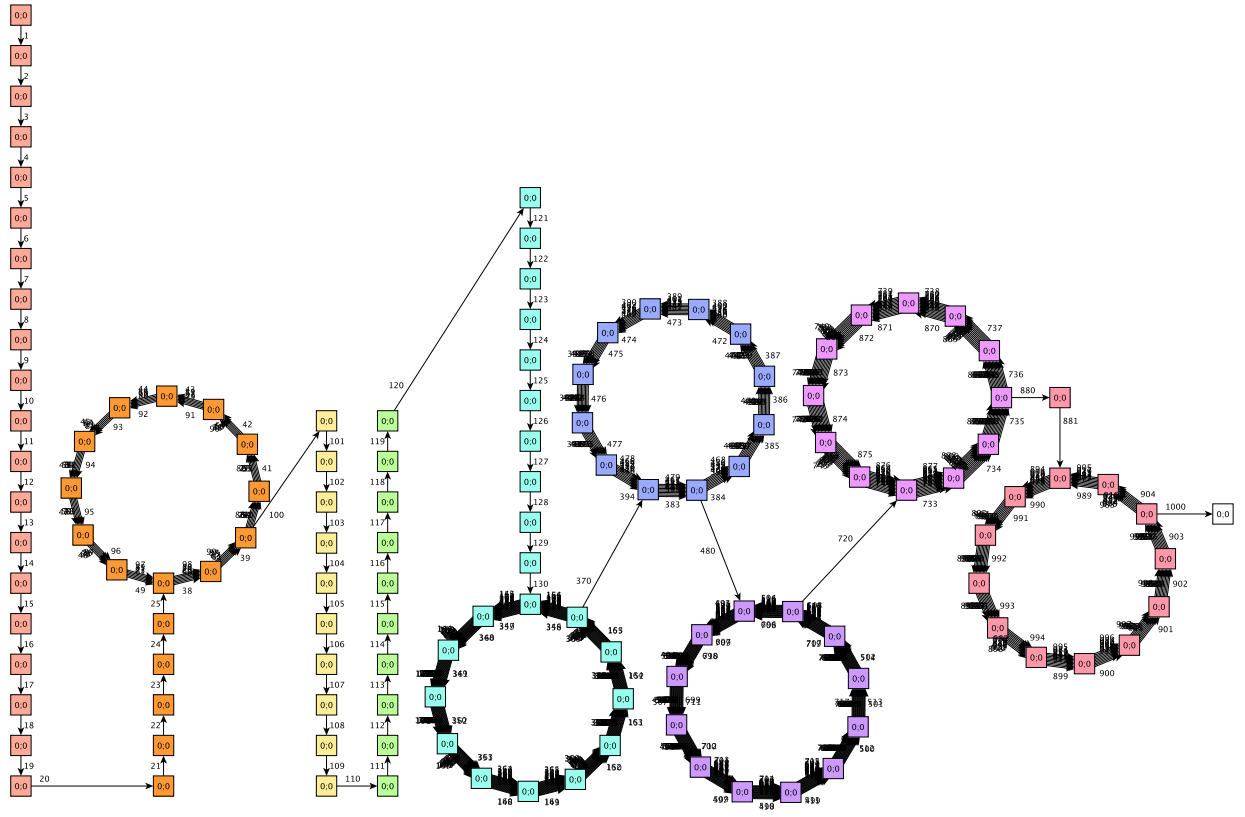

Fig. 9. The LTS of the original run.

```
100% - [aa,alpha_1213r,alpha_ir,alpha_qr,alpha_sr,and_34,ap2,arno,b_parvin,calm,camp,cas,cbp,clathrin,cortactin,crk,csk,dag,diov,dock180,dynamin,egfr_t669,egfr_y1101,
        egfr_y891,egfr_y992,endophilin,epsin,fak,graf,grb2,hip1r,hsc70,ilk,mekk1,mkps,myosin,p190rhogap,pdk1,pi5k,pip2_45,pip3_345,pix_cool,pla2,raf,ral,ralbp1,rin,tiam]
 75% - [ca,cam,camkk,egfr_free,gbg_1213,gbg_i,gbg_q,ip3r1,pa,pi4k,pkc_primed,pld,rgs]
 66% - [eea1,rab5,rab7,rabaptin_5,rabenosyn_5,rabex_5]
 50% - [actin,akt,arf,camk,cbl_rtk,cin85,egfr_t654,erk,ga_1213,gai,gaq,gbg_s,ip3,iqgap1,mek,p115rhogef,p90rsk,pip_4,pka,pkc,plc_b,ptpa,rap1,ras,rkip,sos,talin,vinc]
 25% - [egfr_egfr,egfr_y1045,egfr_y1068,egfr_y1086,egfr_y1148,egfr_y1173]
```

Fig. 10. Activity levels for the entities appearing in the states of the target attractor of the experiment.

We think the picture conveys a lot of interesting information, as it makes evident that:

- the graph contains 129 nodes, which suggests the fact the same target can be reached with a shorter experiment;
- if the same context is provided long enough, the system tends to stabilise in a sort of loop (also called *attractor*);
- this is not the case for the first, third and fourth stages, whose durations are too short: a few more steps would be necessary to reach their corresponding attractors;
- all attractors have the same period (12 nodes) and in fact most of them differ just for the available context;
- it is maybe more meaningful to focus the attention to the target attractor instead of the target state, because if the last stage had a different duration (e.g., one more step or one step less) a different target state would have been returned.

Regarding the last aspect, we have defined a predicate to automatically extract the activity levels of the entities appearing in the state of the target attractor, as reported in Fig. 10.

The LTS of the experiment also opens some important questions:

- Are we guaranteed that the same target attractor will be reached if we change the stage durations as far as the activity levels are preserved (e.g. by performing experiments over a scale of an 700-steps interactive process or a 10000-steps one)?
- What does it happen if we extend the experiment so that the durations of the first, third and fourth stages are long enough to stabilise the system in some attractor loop?
- What does it happen if at some intermediate stage we leave an attractor loop from a different state (by performing, e.g. one additional step with the same context)?

```
myenvironment("[
  x1a = {ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump,alpha_il,egf,il1_tnf,stress}.x1b,
  x1b = ({ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump,alpha_il,egf,il1_tnf,stress}.x1b
      + {ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump,alpha_il,egf}.x2),
  x2 = ({ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump,alpha_il,egf}.x2
      + {ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump,alpha_il}.x3),
  x3 = ({ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump,alpha_il}.x3
      + {ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump}.x4),
  x4 = ({ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql,extpump}.x4
      + {ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql}.x5),
  x5 = ({ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl,alpha_ql}.x5
      + {ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl}.x6),
  x6 = ({ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131,alpha_sl}.x6
      + {ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131}.x7),
  x7 = ({ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm,alpha_12131}.x7
      + {ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm}.x8),
  x8 = ({ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr,ecm}.x8
      + {ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr}.x9),
  x9 = {ras,aa,erk,pip2_34,pip3_345,fak,src,rkip,pak,mekk1,pp2a,pa,egfr_contr}.x9
]").

mycontext("[x1a]").
```

Fig. 11. Specification for the extended experiment: at least one context provision before moving to the next context in the original sequence.

To address these questions, we decided to perform an extended version of the experiment: exploiting nondeterminism and recursion we can define a context process that can decide, *at each step*, whether to provide the same context or move to a different context. This way it is possible to collect *inside a single LTS* the behaviour that explores *all possible combinations for different activity levels of the stimuli and different lengths of the context sequence*. The environment and the context process that we defined to run the extended experiment are in Fig. 11. Notably, the context process allows for infinite computations, but the set of reachable states is finite.

The LTS for the extended experiment is obtained by running the query.

```
?- main(rundigraph,Time).
```

The corresponding graph is in Fig. 12: it has 1783 nodes, 3229 edges and was computed in 672974 ms. To neatly separate states at different stages of the experiment we have surrounded them by large light-blue boxes. Since the graph is very large we find it useful to zoom on a stage of the experiment: Fig. 13 focuses of the states visited when the second context is provided, independently from the number of steps along which the first context has been provided before. They are arranged in a different shape to give a more intuitive idea of the possible computational paths.

Even a superficial inspection of the LTS is enough to conclude that: 1) independently from the (non-zero) activity levels of the stimuli, *the same attractor is always reached at the fourth stage*,[11] and 2) when moving from the fourth stage to the fifth one (i.e., when the context no longer provides the entity `extpump`), six different attractors can be reached; 3) after stage five, for any of the six possible cases, independently from the (non-zero) activity levels of the stimuli, *the same attractor is always reached at the ninth stage*; 4) at each stage, it is now possible to count the least duration that allows to visit all nodes of the corresponding attractor; 5) in principle, *all the attractors present at stage nine can be computed by performing much shorter experiments*.

Further analyses can be done by verifying BioHML formulas. In this particular setting, as a simple example, we can check that, starting from the initial state, in every possible path the entity `mek` appears exactly after four steps. If we set the assertion $\mathsf{F} \triangleq \{\mathtt{mek}\} \subseteq \mathcal{P}$, such property is expressed by the following BioHML formula:

$$[\mathsf{F}]\mathtt{f} \wedge [\neg\mathsf{F}][\mathsf{F}]\mathtt{f} \wedge [\neg\mathsf{F}][\neg\mathsf{F}][\mathsf{F}]\mathtt{f} \wedge [\neg\mathsf{F}][\neg\mathsf{F}][\neg\mathsf{F}]\langle\mathsf{F}\rangle\mathtt{t}$$

The corresponding specification, in the syntax expected by our tool, is in Fig. 14. Then, the query loads the formula and confirms its validity by providing a successful answer in 4.212 ms.

```
?- main(biohml,Time).
```

Since the graph in Fig. 12 is too large to be inspected more closely, we can exploit the above considerations to refine the extended experiment in order to guarantee that an attractor is always reached before moving to the next stage. The corresponding LTS is shown in Fig. 15, where the initial states of each stage are coloured in cyan to ease readability. We think Fig. 12 is well suited to expose the main patterns found in the system.

Finally we can combine the fact that (1) the attractor at stage three can always be reached independently from the duration of stages one and two, and (2) all the attractors at stage nine can be reached from the attractor at stage three independently from the duration of stages four, five, six, seven and eight, to synthesize an even smaller experiment that reveals all the attractors at stage. It is enough to guarantee that the duration of stage three is at least 25 and the duration of stage nine is at least 20, while all the other stages have length one (see Fig. 16). This way each target attractor can be reached in about 50 steps.

[11] To be precise, the experiment is constrained by the requirement that the order of the activity levels among different stimuli in the original experiment is always preserved.

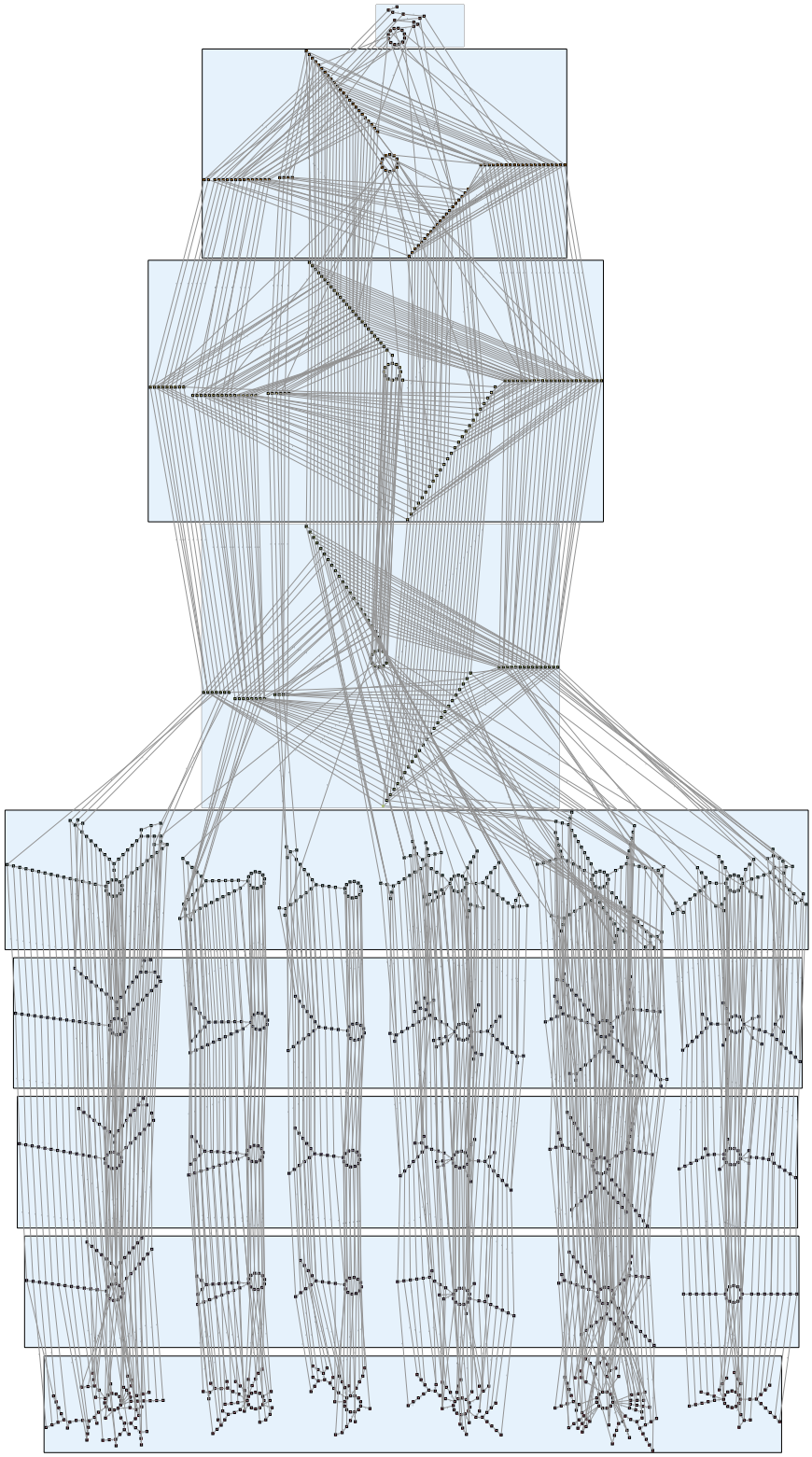

Fig. 12. The LTS of the extended experiment.

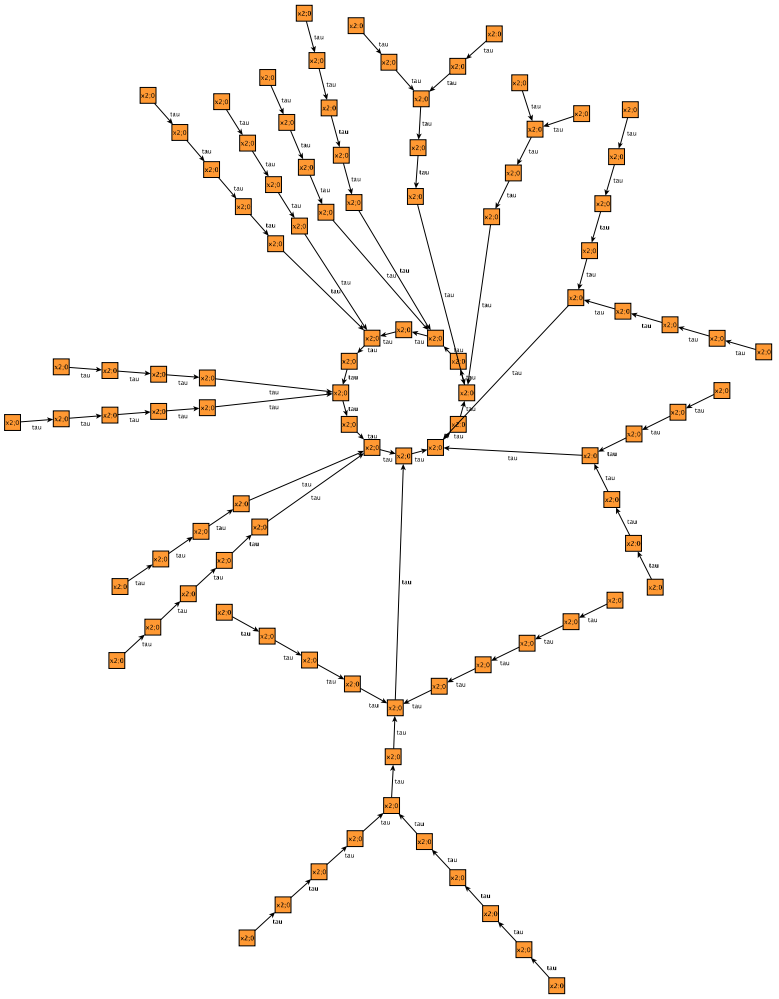

Fig. 13. Zoom on the provision of the second context.

```
mybhml("([{mek} inP]false /\\ [-{mek} inP][{mek} inP]false /\\ [-{mek} inP][-{mek} inP][{mek} inP]false /\\ [-{mek} inP][-{mek} inP][-{mek} inP]<{mek} inP>true)").
```

Fig. 14. Specification for the BioHML formula to be verified.

Notably, most attractors involve the same entities (except `gas` that is present in only one attractor). Table 1 summarises the different activity levels of entities in each attractor. Entities not reported in the table have the same activity levels in each attractor (see Fig. 10 for their activity levels).

The conclusion that we draw from the discussion in this section is that our tool can really help to optimize a biological experiment. In particular for complex experiments, without crossing the data of many interactive processes, it is difficult to check whether all reactants are necessary and used, how many steps should be considered to design an interactive process for a given biological experiment, or if it would be possible to achieve the same results by saving some of the steps and reactants. The strategies illustrated on the ErbB case study can thus have a significant impact in terms of time and cost of expensive entities. The concept of attractor was introduced to facilitate the overall analysis of an experiment, by emphasizing the looping behaviour of subparts of the experiment, and detecting some regularities. As future work, we think that our graphical analysis tool can be further enhanced, by combining it with the extensions described in Section 6.

5.4. Related tools

In the literature there are several simulation tools available for the ordinary semantics of RSs based on interactive processes.

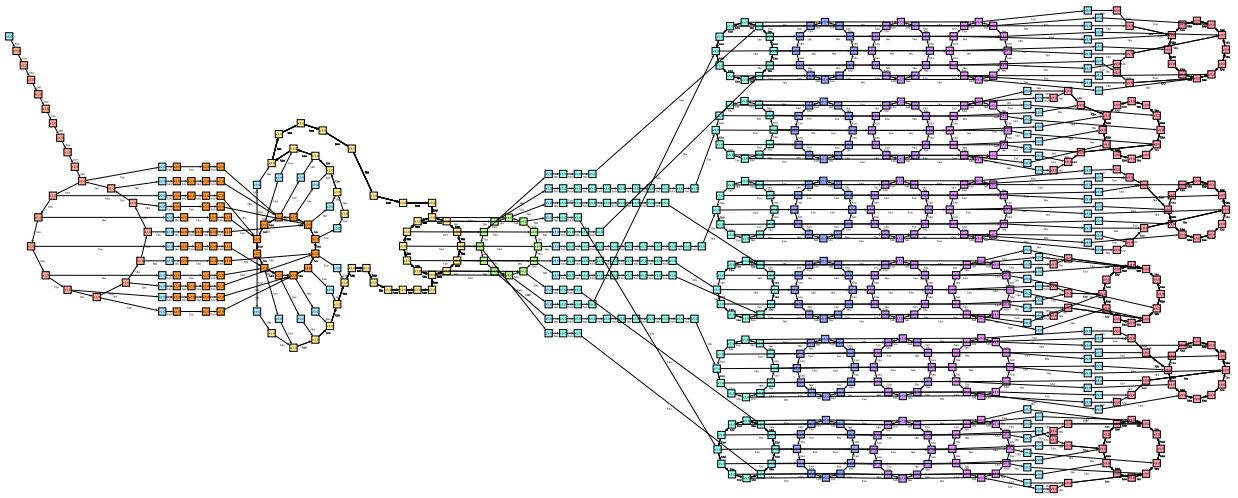

Fig. 15. Extended experiment where an attractor is always reached before moving to the next stage.

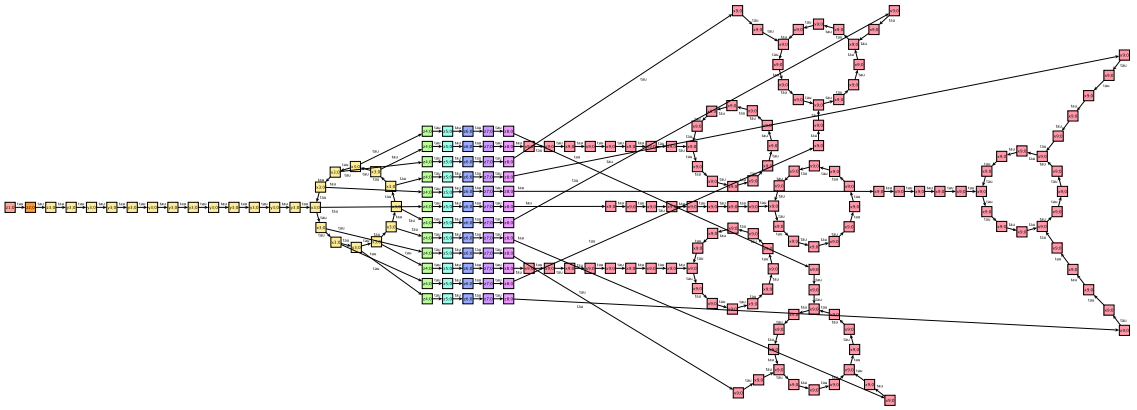

Fig. 16. Extended experiment with least durations.

The first simulator to be made publicly available was `brsim`[12] (Basic Reaction System Simulator, written in Haskell and distributed under the terms of GNU GPLv3 license) [31]. Given the reactions of the RS and a context sequence, `brsim` can compute the result sequence as well as produce further annotations for each computation step (like the result, the enabled reactions or the conserved sets). Alternatively, `brsim` can be run in a user-interactive mode, in the sense that the user can input the next context to use at every step. Interestingly, the online version of the tool, called `WEBRSIM`,[13] makes all functionalities of `brsim` available through a friendly web interface [32].

`HERESY`[14] (Highly Efficient REaction SYstem simulator) is a GPU-based simulator of RSs, written using CUDA, equipped with an intuitive Graphic User Interface and able to deal with very large-scale systems, thanks to the possibility to exploit the large number of computational units inside GPUs [33]. When GPUs are not available, a CPU-based version of `HERESY` written in Python 2 can be used, but in that case it is in general less performant than `brsim`.

Finally, Ferretti et al. [34] presented the tool `cl-rs`,[15] which is an optimized Common Lisp simulator for RSs. The performances of all these tools have been compared in [34], where it was shown that `cl-rs` is able to offer performances comparable with the GPU-based simulator `HERESY` on the already discussed ErbB model. `cl-rs` can employ the direct (basic) simulation method, or can also use other more performant simulations. Thus, an optimised simulation can consider the graph of dependencies between reactions, avoiding the simulation of parts of the reactions that cannot produce effects on its dynamics. Another alternative simulation methodology is obtained by first rewriting the dynamical evolution of a

[12] Available at https://github.com/scolobb/brsim/.
[13] Available at https://combio.org/portfolio/webrsim-reaction-system-simulator/.
[14] Available at https://github.com/aresio/HERESY/.
[15] Available at https://github.com/mnzluca/cl-rs.

Table 1
A rough comparison of activity levels in the six possible target attractors.

	1	2	3	4	5	6
actin	50%			66%		
ca	75%			66%		
cam	75%			66%		
camk	50%			33%		
camkk	75%			66%		
cbl_rtk	50%			33%		
cin85	50%			33%		
egfr_egfr	25%			33%		
egfr_free	75%			66%		
egfr_t654	50%			66%		
egfr_y1045	25%			33%		
egfr_y1068	25%			33%		
egfr_y1086	25%			33%		
egfr_y1148	25%			33%		
egfr_y1173	25%			33%		
gai	50%			33%		
gaq	50%			33%		
gas	0%		50%	0%		
gbg_i	75%			66%		
gbg_q	75%			66%		
gbg_s	50%		75%	50%		
ip3	75%	50%				
ip3r1	75%			66%		
iqgap1	50%			66%		
pa	75%			83%		
pi4k	75%			83%		
pkc	50%			66%		
pkc_primed	75%			66%		
plc_b	75%	50%				
pld	75%			83%		
ptpa	50%			66%		
rap1	75%	50%		66%		
ras	50%			66%		
rgs	75%			66%		
rkip	50%			66%		
sos	50%			66%		

reaction system in terms of matrix-vector multiplications, vector additions, and clipping operations. This way Ferretti et al. [34] defined a proof-of-concept implementation in Python 3 exploiting the efficient linear algebra libraries of Python.

Shang et al. in [35] presented the first attempt to implement RSs in hardware. They describe algorithms for translating RSs into synchronous digital circuits keeping the same behaviour. They also developed a compiler translating a RS description into an hardware circuit description using field-programming gate arrays. Obviously the performance is the best possible one, as execution is at the hardware level. On the other hand for the same reason flexibility of this realization is at the least level.

While it would be ungenerous to compare simply the performances of our proof-of-concept Prolog implementation with the above listed highly-performant simulators, we remind that our implementation introduces several novel features not covered in the literature and it has been designed as a tool for verification, as well as for rapid prototyping extensions of

RSs, not just for their simulations. Of course, an advantage of declarative (logic) programming is that the actual code is very close to the mathematical description of the framework, which facilitate its development, documentation, maintenance and updates. In particular, the correctness of most predicates trivially follows from their definitions. It is also an environment on which we will base future extensions and analyses of RSs, e.g. quantitative extensions of RSs in the style described in the following section.

As explained at the beginning of this section, our implementation is loosely rooted on the first prototype developed in [25]. This first prototype was designed around ordinary interactive processes (i.e. context sequences) and exploited a simple technique for memoizing the state sequence and checking for repetitions of states during a computation. We have first extended the work in [25] by including nondeterministic contexts and BioHML formula verification (see [17]) and then we have added the capabilities to deal with recursive contexts, bio-similarity check and LTS visualization, to be fully compliant to the theory presented in this paper. The current implementation has added also several other useful features such as a friendly user interface, a parser, and a graphical tool.

6. Two extensions

Here we present two extensions: a numeric extension that takes into account the number of times an entity is used as a reactant in a single transition, and an extension that introduces an operator for letting two RSs be *connected*.

Reactant occurrences.
The first idea is to introduce some naive measure for the number of entities that are needed by the reactions. Now, we assume that the number associated to entities in the sets R (reactants) and P (products) are the stoichiometric numbers, as specified in the corresponding biochemical equation. This amounts to use multisets instead of sets (for R and P) within the labels. The set I (of inhibitors) remains a simple set. At the level of notation, we write a multiset as a formal sum $\bigoplus_{\mathsf{a}\in S} n_{\mathsf{a}}\mathsf{a}$, where $n_{\mathsf{a}} \in \mathbb{N}$ is the number of occurrences of a. For simplicity, we write just a instead of 1a and we omit any term of the form 0a. For example, the multiset $2\mathsf{a} \oplus \mathsf{b}$ has two instances of a and one of b. Overloading the notation we use $\cup$ as multiset union, i.e.

$$(\bigoplus_{\mathsf{a}\in S} n_{\mathsf{a}}\mathsf{a}) \cup (\bigoplus_{\mathsf{a}\in S} m_{\mathsf{a}}\mathsf{a}) = \bigoplus_{\mathsf{a}\in S} (n_{\mathsf{a}} + m_{\mathsf{a}})\mathsf{a}$$

If $R = \bigoplus_{\mathsf{a}\in S} n_{\mathsf{a}}\mathsf{a}$ we let $R(a) = n_{\mathsf{a}}$.

Similarly, we want to use multisets also for the contexts, but in this case we want the possibility to parameterize the context w.r.t. the number of entities it provides. To this purpose, fixed a finite set $X = \{x_1, ..., x_n\}$ of variables, we introduce some linear expressions of the form $e = \sum_{i=1}^{n} k_i x_i + h$ with coefficients $k_i, h \in \mathbb{N}$, such that a context C associates to each entity a a linear expression e_{a} and not just a number. Thus we write a context C as a formal sum $C = \bigoplus_{\mathsf{a}\in S} e_{\mathsf{a}}\mathsf{a}$. A multiset is just a particular case of the above expression where all variable coefficients are 0. For example, we can let $C = (x + y)\mathsf{a} \oplus (x + 1)\mathsf{b}$. The union of contexts is then defined as follows

$$\bigoplus_{\mathsf{a}\in S} e^1_{\mathsf{a}}\mathsf{a} \cup \bigoplus_{\mathsf{a}\in S} e^2_{\mathsf{a}}\mathsf{a} = \bigoplus_{\mathsf{a}\in S} (e^1_{\mathsf{a}} + e^2_{\mathsf{a}})\mathsf{a}$$

We assume that variables in X can only range over positive values, so that if $e_{\mathsf{a}} \neq 0$ then a is present in $\bigoplus_{\mathsf{a}\in S} e_{\mathsf{a}}\mathsf{a}$.

In the SOS rules we need to use the requirements $(W \cup R) \cap I = \emptyset$ and $R \subseteq W$. They are intended to be satisfied at the qualitative level, not necessarily at the quantitative one. Correspondingly, the disjointness condition $(W \cup R) \cap I = \emptyset$ is satisfied when $\forall \mathsf{a} \in I.\ (W \cup R)(\mathsf{a}) = 0$, and the inclusion condition $R \subseteq W$ is satisfied when $\forall \mathsf{a} \in S.\ R(\mathsf{a}) \neq 0 \Rightarrow W(\mathsf{a}) \neq 0$. Our new transition labels differ from the ones in Fig. 1 just because R, P, and W are now multisets. We keep the same SOS rules as before.

The advantage is that to each transition $\mathsf{P} \xrightarrow{\langle W \rhd R, I, P\rangle} \mathsf{P}'$ we can now assign a system of linear inequalities: $\forall \mathsf{a} \in S.\ R(\mathsf{a}) \leq W(\mathsf{a})$, where $R(\mathsf{a}) \in \mathbb{N}$ and $W(\mathsf{a})$ is an expression. The aim is to estimate, with no computational effort, the *relative quantities* of biological material which should be provided to the system to reach a desired configuration. This could be helpful during the setting phase of an *in vitro* experiment to avoid over-use of biological material, given its high cost. Please note that the qualitative nature of RS is unchanged, we only add some extra information that we elaborate by manipulating transition labels, only. Here we give an intuition with a short example.

Example 36. Let us consider the chemical reactions in Azimi et al. [7], Table 3, in particular reactions (i) and (vii); we will use their formalization in the syntax of RS, by keeping the stoichiometric numbers:

$$a_1 \triangleq (\{(\mathsf{hsf}, 3)\}, \{\mathsf{d_l}\}, \{\mathsf{hsf_3}\}) \qquad a_2 \triangleq (\{\mathsf{hsp}, \mathsf{hsf_3}\}, \{\mathsf{d_l}\}, \{\mathsf{hsp{:}hsf}, (\mathsf{hsf}, 2)\})$$

Reaction a_1 requires three copies of the entity hsf, while a_2 produces two copies of hsf. We assume that the context initially provides the set $C \triangleq x\mathsf{hsf} \oplus \mathsf{hsp} \oplus \mathsf{hsf_3}$ and then it provides the empty set, i.e. it is defined as $\mathsf{K} \triangleq C.\emptyset.\mathbf{0}$. The resulting system

$$\frac{\mathsf{P}_1 \xrightarrow{\langle W_1 \rhd R_1, I_1, P_1\rangle} \mathsf{P} \quad \mathsf{P}_2 \xrightarrow{\langle W_2 \rhd R_2, I_2, P_2\rangle} [\mathsf{M}]}{\mathsf{P}_1 \overset{L}{\Mapsto} \mathsf{P}_2 \xrightarrow{\langle W_1 \cup W_2 \rhd R_1 \cup R_2, I_1 \cup I_2, P_1 \cup P_2\rangle} \mathsf{P} \overset{L}{\Mapsto} [\mathsf{M}|(L \cap P_1)]} \ (Lnk)$$

Fig. 17. SOS semantics rule for the connector operator.

can only execute two transitions: in the first transition both reactions a_1 and a_2 are applied, in the second transition only reaction a_1 is applied:

$$[\mathsf{K}|a_1|a_2] \xrightarrow{\langle C \rhd R, I, P\rangle} [P|\emptyset.\mathbf{0}|a_1|a_2] \xrightarrow{\langle P \rhd R', I', P'\rangle} [P'|\mathbf{0}|a_1|a_2]$$

where

$$R = 3\mathsf{hsf} \oplus \mathsf{hsp} \oplus \mathsf{hsf}_3 \qquad I = \{\mathsf{d}_\mathsf{I}\} \qquad P = \mathsf{hsf}_3 \oplus \mathsf{hsp{:}hsf} \oplus 2\mathsf{hsf}$$
$$R' = 3\mathsf{hsf} \qquad I' = \{\mathsf{hsp}, \mathsf{d}_\mathsf{I}\} \qquad P' = \mathsf{hsf}_3$$

Now, from the first transition we extract the requirement $R(\mathsf{hsf}) = 3 \leq C(\mathsf{hsf}) = x$, while from the second transition we get $R'(\mathsf{hsf}) = 3 \leq P(\mathsf{hsf}) = 2$. If we would wanted a quantitative estimate of need of entity hsf, this comparison would reveal that the production of hsf is not sufficient to trigger the second reaction.

The connector operator.
In Bodei et al. [15] and Brodo et al. [16] we have presented the encoding of RS into the `link`-calculus and we have already discussed how to connect two encoded RS such that some of the entities produced by one RS are provided to the second one, similarly to what has been done in Bottoni et al. [36]. To this aim we introduce an operator, that we call "connector", written as $\mathsf{P}_1 \overset{L}{\Mapsto} \mathsf{P}_2$, meaning that when the RS process P_1 produces entities in the set L, these entities are available, at the next step, as reactants to the continuations of both RS processes. As a special case, when $L = \emptyset$, there cannot be any exchange of entities and P_1 and P_2 run in parallel, but in isolation.

For a simple example, we give the definition of a system composed by two RSs P_1 and P_2: P_1 is composed of two reactions $a_1 = (a, \ , c)$ and $a_2 = (a, \ , a)$ and we assume that the initial state only contains the entity a, hence $S_1 = \{a\}$. P_2 is composed of two reactions $a'_1 = (e, \ , d)$ and $a'_2 = (dc, \ , e)$ and we assume that the initial state only contains the entity e, hence $S_2 = \{e\}$. For simplicity, we omit the contexts and the inhibitors. Now, we want to describe a system where the entity c produced by P_1 is also made available for P_2. Then, the whole system is: $\mathsf{P}_1 \overset{c}{\Mapsto} \mathsf{P}_2$. Let assume that $\mathsf{P}_1 \xrightarrow{\langle\{a\} \rhd \{a\}, \emptyset, \{a,c\}\rangle} \mathsf{P}'_1$ and $\mathsf{P}_2 \xrightarrow{\langle\{e\} \rhd \{e\}, \emptyset, \{d\}\rangle} [a'_1|a'_2|\{d\}]$, then, by rule (*Lnk*) (see Fig. 17), we derive the transition of the whole system as follows:

$$\frac{\mathsf{P}_1 \xrightarrow{\langle\{a\} \rhd \{a\}, \emptyset, \{a,c\}\rangle} \mathsf{P}'_1 \quad \mathsf{P}_2 \xrightarrow{\langle\{e\} \rhd \{e\}, \emptyset, \{d\}\rangle} [a'_1|a'_2|\{d\}]}{\mathsf{P}_1 \overset{c}{\Mapsto} \mathsf{P}_2 \xrightarrow{\langle\{a,e\} \rhd \{a,e\}, \emptyset, \{a,c,d\}\rangle} \mathsf{P}'_1 \overset{c}{\Mapsto} [a'_1|a'_2|\{d, c\}]} \ (Lnk)$$

7. Conclusion and future work

We have presented an SOS semantics for an extension of RSs, considering non deterministic and recursive contexts, that generates a labelled transition system. We have revised RSs as processes, formulating a set of ad-hoc inference rules. We have defined a flexible framework that allows one to add new operators in a natural way. It is important to note that the transition labels play an interesting role, not only because they reflect the important aspect of the computations, but also because they can add expressivity at the computation allowing for additional analysis, as we did in Section 4. In Section 5 we have described a prototype implementation in logic programming. We have implemented several tools, which are all available online, by means of a user friendly interface. The interpreter is written in SWI-Prolog and can be run either on a web site based on Tau Prolog, or much more efficiently by using the desktop version of SWI-Prolog. A parser allows the user to make the input of formulas using concrete syntax. Our interpreter allows the user to derive the LTS of a Reaction System. A tool can verify the validity of BioHML formulas on computations of RS processes with extended contexts in our assertion-based variant of the Hennessy-Milner logic. A tool can also verify the biosimilarity of two adversarial RS processes. Moreover, the structure of a RS process can be shown graphically, thus helping the user to analyse the behaviour and evolution of the modelled system.

As future work we plan to apply our tool to show that it helps to analyse the behaviour of non trivial biological systems, helping the biologist to understand the interaction of the context with the modelled system. This can lead to understand how to choose a dosage of the reactants in a wet lab experiment in order to drive it and take paths which can lead to the target result by reducing the necessary quantities of reactants.

The SOS semantics paves the way for a systematic integration of other operators for combining Reaction Systems, like the ones described in Section 6 and many others available from the process algebra literature [37] (e.g. hiding, interleaving, external choice). Analogously, we plan to investigate de-synchronised versions of Reaction Systems, where some of the enabled reactions, but not necessarily all of them, can take place at each computation step, as well as Reaction Systems where reactions can occur at different speeds, which is often the case in many biological systems.

We also plan to apply our technique to define SOS semantics for other synchronous rewrite-rule systems (where all the rules are applied synchronously) to define a uniform computational framework. In order to make our implementation more efficient we can use constraint logic programs over sets [38] and with finite domains [39]. We also want to study the relation to analysis techniques [40–42] and slicing techniques [43]. As future investigation we think that the notion of periodicity and attractors that we have discussed in Section 5 can be related to the notions of extended RS and events introduced in [44].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Proofs

Here follow the proofs of the technical lemmas and the theorem stated in the main body of the paper.

Lemma 15. *If* $\mathsf{M} \xrightarrow{\langle W \rhd R, I, P\rangle} \mathsf{M}'$ *then* $\mathsf{M}' \equiv \mathsf{M}''|P$ *for some* M''.

Proof. The proof is by rule induction.

Ent) We need to prove that $\emptyset \equiv \mathsf{M}''|\emptyset$ for some M'', which is immediate by taking $\mathsf{M}'' = \emptyset$.
Cxt) We need to prove that $\mathsf{K} \equiv \mathsf{M}''|\emptyset$ for some M'', which is immediate by taking $\mathsf{M}'' = \mathsf{K}$.
Rec) We assume as inductive hypothesis that $\mathsf{K}' \equiv \mathsf{M}'''|P$ for some M''' and we need to prove that $\mathsf{K}' \equiv \mathsf{M}''|P$ for some M'', which is immediate by taking $\mathsf{M}'' = \mathsf{M}'''$.
Suml) We assume as inductive hypothesis that $\mathsf{K}'_1 \equiv \mathsf{M}_1|P$ for some M_1 and we need to prove that $\mathsf{K}'_1 \equiv \mathsf{M}''|P$ for some M'', which is immediate by taking $\mathsf{M}'' = \mathsf{M}_1$.
Sumr) Analogous to the previous case.
Pro) We need to prove that $(R, I, P)|P \equiv \mathsf{M}''|P$ for some M'', which is immediate by taking $\mathsf{M}'' = (R, I, P)$.
Inh) We need to prove that $(R, I, P) \equiv \mathsf{M}''|\emptyset$ for some M'', which is immediate by taking $\mathsf{M}'' = (R, I, P)$.
Par) We assume as inductive hypotheses that $\mathsf{M}'_1 \equiv \mathsf{M}''_1|P_1$ for some M''_1 and that $\mathsf{M}'_2 \equiv \mathsf{M}''_2|P_2$ for some M''_2. We need to prove that $\mathsf{M}'_1|\mathsf{M}'_2 \equiv \mathsf{M}''|(P_1 \cup P_2)$ for some M'', which is immediate by taking $\mathsf{M}''_1|\mathsf{M}''_2$ (thanks to the laws of commutative monoids and by $(P_1 \cup P_2) \equiv P_1|P_2$).
Sys) The conclusion of the rule deals with RS processes, not with mixture processes, so it can be ignored.

Lemma 16. *If* $\prod_{a\in A} a \xrightarrow{\langle W \rhd R, I, P\rangle} \mathsf{M}$ *then* $W = \emptyset$ *and* $\mathsf{M} \equiv \prod_{a\in A} a \mid P$.

Proof. The proof is by rule induction. Rules (*Ent*), (*Cxt*), (*Rec*), (*Suml*), (*Sumr*) and (*Sys*) can be ignored because their conclusions cannot match $\prod_{a\in A} a$.

Pro) It is immediate to check that we have indeed $W = \emptyset$ and $(R, I, P)|P \equiv (R, I, P)|P$.
Inh) It is immediate to check that we have indeed $W = \emptyset$ and $(R, I, P) \equiv (R, I, P)|\emptyset$.
Par) We assume as inductive hypotheses that $\mathsf{M}_1 = \prod_{a\in A_1} a$, $\mathsf{M}_2 = \prod_{a\in A_2} a$, $W_1 = W_2 = \emptyset$, $\mathsf{M}'_1 \equiv \mathsf{M}_1|P_1$, $\mathsf{M}'_2 \equiv \mathsf{M}_2|P_2$. We need to prove that $W_1 \cup W_2 = \emptyset$ and $\mathsf{M}'_1|\mathsf{M}'_2 \equiv \prod_{a\in(A_1\cup A_2)} a \mid (P_1 \cup P_2)$, which is immediate thanks to the laws of commutative monoids.

Lemma 17. *If* $\mathsf{M} \xrightarrow{\langle W \rhd R, I, P\rangle} \mathsf{M}'$ *then* $(W \cup R) \cap I = \emptyset$.

Proof. The proof is by rule induction.

Ent) We have indeed $(D \cup \emptyset) \cap \emptyset = \emptyset$.
Cxt) We have indeed $(C \cup \emptyset) \cap \emptyset = \emptyset$.
Rec) We assume as inductive hypothesis that $(W \cup R) \cap I = \emptyset$ that coincides with our goal.
Suml) Analogous to the previous case.
Sumr) Analogous to the previous case.
Pro) We need to prove that $(\emptyset \cup R) \cap I = \emptyset$, which is a consequence of the constraint that in any reaction (R, I, P) the set of reactants is disjoint from the set of inhibitors.
Inh) We need to prove that $(\emptyset \cup J) \cap Q = \emptyset$, which is a consequence of the constraint that in any reaction (R, I, P) the set of reactants R is disjoint from the set of inhibitors I, as $Q \subseteq R$ and $J \subseteq I$.
Par) We need to prove that $(W_1 \cup W_2 \cup R_1 \cup R_2) \cap (I_1 \cup I_2) = \emptyset$, which is in fact one premise of the rule.
Sys) The conclusion of the rule deals with RS processes, not with mixture processes, so it can be ignored.

Lemma 18. *If* $\mathsf{P} \xrightarrow{\langle W \rhd R, I, P\rangle} \mathsf{P}'$ *then* $R \subseteq W$ *and* $W \cap I = \emptyset$.

Proof. Let us assume that $\mathsf{P} \xrightarrow{\langle W \rhd R, I, P\rangle} \mathsf{P}'$. Since the only SOS rule applicable to RS processes is (*Sys*), it must be the case that $\mathsf{P} = [\mathsf{M}]$, with $[\mathsf{M}] \xrightarrow{\langle W \rhd R, I, P\rangle} [\mathsf{M}']$, $R \subseteq W$ and $\mathsf{P}' = [\mathsf{M}']$. By Lemma 17, we have $(W \cup R) \cap I = \emptyset$, i.e., $W \cap I = \emptyset$.

The main theorem shows that the rewrite steps of a RS exactly match the transitions of its corresponding RS process.

Theorem 19. *Let* $\mathcal{A} = (S, A)$ *be a RS, and* $\pi = (\gamma, \delta)$ *an n-step interactive process in* $\mathcal{A}$ *with* $\gamma = \{C_i\}_{i\in[0,n]}$, $\delta = \{D_i\}_{i\in[0,n]}$, *and let* $W_i \triangleq C_i \cup D_i$ *and* $\mathsf{P}_i \triangleq [\![\mathcal{A}, \pi]\!]_i$ *for any* $i \in [0, n]$. *Then:*

1. $\forall i \in [0, n-1]$, $\mathsf{P}_i \xrightarrow{\langle W \rhd R, I, P\rangle} \mathsf{P}$ *implies* $W = W_i$, $P = D_{i+1}$ *and* $\mathsf{P} \equiv \mathsf{P}_{i+1}$;
2. $\forall i \in [0, n-1]$, *there exists* $R, I \subseteq S$ *such that* $\mathsf{P}_i \xrightarrow{\langle W_i \rhd R, I, D_{i+1}\rangle} \mathsf{P}_{i+1}$.

Proof. We prove the two items separately.

1. Take $i \in [0, n-1]$ and suppose $\mathsf{P}_i \xrightarrow{\langle W \rhd R, I, P\rangle} \mathsf{P}$, with
$$\mathsf{P}_i = \left[\prod_{a\in A} a \mid D_i \mid C_i.C_{i+1}.\cdots.C_n.\mathbf{0}\right]$$
By Lemma 18 we know that $R \subseteq W$ and $W \cap I = \emptyset$. By rule (*Sys*) it must be the case that
$$\prod_{a\in A} a \mid D_i \mid C_i.C_{i+1}.\cdots.C_n.\mathbf{0} \xrightarrow{\langle W \rhd R, I, P\rangle} \mathsf{M}$$
with $\mathsf{P} = [\mathsf{M}]$. By Lemma 17 we know that $(W \cup R) \cap I = \emptyset$. By rule (*Par*) it must be the case that
$$\prod_{a\in A} a \xrightarrow{\langle W_1 \rhd R_1, I_1, P_1\rangle} \mathsf{M}_1 \quad \text{and} \quad D_i \mid C_i.C_{i+1}.\cdots.C_n.\mathbf{0} \xrightarrow{\langle W_2 \rhd R_2, I_2, P_2\rangle} \mathsf{M}_2$$
with $\mathsf{M} = \mathsf{M}_1 \mid \mathsf{M}_2$, $W = W_1 \cup W_2$, $R = R_1 \cup R_2$, $I = I_1 \cup I_2$, $P = P_1 \cup P_2$. By Lemma 16 it must be $W_1 = \emptyset$ and $\mathsf{M}_1 = \prod_{a\in A} a \mid P_1$. Observing that the only possible transition for D_i and $C_i.C_{i+1}.\cdots.C_n.\mathbf{0}$ are, respectively, $D_i \xrightarrow{\langle D_i \rhd \emptyset, \emptyset, \emptyset\rangle} \emptyset$ and $C_i.C_{i+1}.\cdots.C_n.\mathbf{0} \xrightarrow{\langle C_i \rhd \emptyset, \emptyset, \emptyset\rangle} C_{i+1}.\cdots.C_n.\mathbf{0}$, it must be the case that
$$D_i \mid C_i.C_{i+1}.\cdots.C_n.\mathbf{0} \xrightarrow{\langle W_i \rhd \emptyset, \emptyset, \emptyset\rangle} \emptyset \mid C_{i+1}.\cdots.C_n.\mathbf{0} \equiv C_{i+1}.\cdots.C_n.\mathbf{0}$$
Hence $\mathsf{M}_2 \equiv C_{i+1}.\cdots.C_n.\mathbf{0}$, $W_2 = W_i$ and $R_2 = I_2 = P_2 = \emptyset$, from which $W = W_i$, $R = R_1$, $I = I_1$ and $P = P_1$. Now observe that, for each rule $a = (R_a, I_a, P_a)$ there are two possibilities:
   - if $R_a \subseteq R$ and $I_a \subseteq I$ then $en_a(W_i)$ (because $R_a \subseteq R \subseteq W = W_i$ and $I_a \cap W \subseteq I \cap W = I \cap W_i = \emptyset$) and the rule (*Pro*) must have been applied to the process a and therefore $P_a \subseteq P$;
   - otherwise, the rules (*Inh*) must have been applied to a and therefore the transition label contributed with $\emptyset$ to P.

   From the two observations above it follows that $P = res_A(W_i) = D_{i+1}$. Summing up, we have $W = W_i$, $P = D_{i+1}$ and
$$\mathsf{P} \equiv [\mathsf{M}] \equiv [\mathsf{M}_1 \mid \mathsf{M}_2] \equiv \left[\prod_{a\in A} a \mid D_{i+1} \mid C_{i+1}.\cdots.C_n.\mathbf{0}\right] \equiv [\![\mathcal{A}, \pi]\!]_{i+1} \equiv \mathsf{P}_{i+1}$$
2. Take $i \in [0, n-1]$. Let $en(W_i) \triangleq \{a \in A \mid en_a(W_i)\}$. Observe that
   - By rule (*Par*) and (repeated applications of) rule (*Pro*) we have
$$\prod_{a\in en(W_i)} a \xrightarrow{\langle \emptyset \rhd R_1, I_1, P_1\rangle} \prod_{a\in en(W_i)} a \mid P_1$$
   with $R_1 = \bigcup_{a\in en(W_i)} R_a$, $I_1 = \bigcup_{a\in en(W_i)} I_a$, and $P_1 = \bigcup_{a\in en(W_i)} P_a = res_A(W_i) = D_{i+1}$.
   - For each rule $a \in A \setminus en(W_i)$ there must exist some sets $J_a \subseteq I_a \cap W_i$ (inhibitors that are present) and $Q_a \subseteq R_a \setminus W_i$ (missing reactants) with $J_a \cup Q_a \neq \emptyset$, so that by rule (*Par*) and (repeated applications of) rule (*Inh*) we have
$$\prod_{a\in A\setminus en(W_i)} a \xrightarrow{\langle \emptyset \rhd R_2, I_2, \emptyset\rangle} \prod_{a\in A\setminus en(W_i)} a$$
   with $R_2 = \bigcup_{a\in A\setminus en(W_i)} J_a$ and $I_2 = \bigcup_{a\in A\setminus en(W_i)} Q_a$.

Then, by rule (*Par*) we have

$$\prod_{a\in A} a \xrightarrow{\langle \emptyset \rhd R_1\cup R_2, I_1\cup I_2, D_{i+1}\rangle} \prod_{a\in A} a \mid D_{i+1}$$

because, by construction, $R_1 \cup R_2 \subseteq W_i$ and $W_i \cap (I_1 \cup I_2) = \emptyset$ and thus $(R_1 \cup R_2) \cap (I_1 \cup I_2) = \emptyset$. Moreover, by rules (*Par*), (*Ent*) and (*Cxt*), we have

$$D_i \mid C_i.C_{i+1}.\cdots.C_n.\mathbf{0} \xrightarrow{\langle W_i \rhd \emptyset,\emptyset,\emptyset\rangle} \emptyset \mid C_{i+1}.\cdots.C_n.\mathbf{0} \equiv C_{i+1}.\cdots.C_n.\mathbf{0}$$

Thus, by rules (*Par*) and (*Sys*), and letting $R = R_1 \cup R_2$ and $I = I_1 \cup I_2$, we have

$$\mathsf{P}_i \xrightarrow{\langle W_i \rhd R, I, D_{i+1}\rangle} \left[\prod_{a\in A} a \mid D_{i+1} \mid C_{i+1}.\cdots.C_n.\mathbf{0}\right] \equiv \mathsf{P}_{i+1}$$

References

[1] G.D. Plotkin, A structural approach to operational semantics, Tech. Rep. DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
[2] G.D. Plotkin, A structural approach to operational semantics, J. Log. Algebraic Methods Program. 60–61 (2004) 17–139, https://doi.org/10.1016/j.jlap.2004.05.001.
[3] R. Milner, A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92, Springer, 1980.
[4] G.D. Plotkin, An operational semantics for CSP, in: D. Bjørner (Ed.), Proceedings of the IFIP Working Conf. on Formal Description of Programming Concepts-II, Garmisch-Partenkirchen, North-Holland, 1982, pp. 199–226.
[5] J. Hillston, A compositional approach to performance modelling, Ph.D. thesis, University of Edinburgh, UK, 1994.
[6] R. Brijder, A. Ehrenfeucht, M. Main, G. Rozenberg, A tour of reaction systems, Int. J. Found. Comput. Sci. 22 (07) (2011) 1499–1517, https://doi.org/10.1142/S0129054111008842.
[7] S. Azimi, B. Iancu, I. Petre, Reaction system models for the heat shock response, Fundam. Inform. 131 (3–4) (2014) 299–312, https://doi.org/10.3233/FI-2014-1016.
[8] L. Corolli, C. Maj, F. Marinia, D. Besozzi, G. Mauri, An excursion in reaction systems: from computer science to biology, Theor. Comput. Sci. 454 (2012) 95–108, https://doi.org/10.1016/j.tcs.2012.04.003.
[9] S. Azimi, Steady states of constrained reaction systems, Theor. Comput. Sci. 701 (C) (2017) 20–26, https://doi.org/10.1016/j.tcs.2017.03.047.
[10] R. Barbuti, R. Gori, F. Levi, P. Milazzo, Investigating dynamic causalities in reaction systems, Theor. Comput. Sci. 623 (2016) 114–145, https://doi.org/10.1016/j.tcs.2015.11.041.
[11] F. Okubo, T. Yokomori, The computational capability of chemical reaction automata, Nat. Comput. 15 (2) (2016) 215–224, https://doi.org/10.1007/s11047-015-9504-7.
[12] A. Ehrenfeucht, M.G. Main, G. Rozenberg, Combinatorics of life and death for reaction systems, Int. J. Found. Comput. Sci. 21 (3) (2010) 345–356, https://doi.org/10.1142/S0129054110007295.
[13] A. Ehrenfeucht, M.G. Main, G. Rozenberg, Functions defined by reaction systems, Int. J. Found. Comput. Sci. 22 (1) (2011) 167–178, https://doi.org/10.1142/S0129054111007927.
[14] J. Kleijn, M. Koutny, Ł. Mikulski, G. Rozenberg, Reaction systems, transition systems, and equivalences, in: H. Böckenhauer, D. Komm, W. Unger (Eds.), Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday, in: LNCS, vol. 11011, Springer, 2018, pp. 63–84.
[15] L. Brodo, R. Bruni, M. Falaschi, Enhancing reaction systems: a process algebraic approach, in: M. Alvim, K. Chatzikokolakis, C. Olarte, F. Valencia (Eds.), The Art of Modelling Computational Systems, in: LNCS, vol. 11760, Springer, Berlin, 2019, pp. 68–85.
[16] L. Brodo, R. Bruni, M. Falaschi, A process algebraic approach to reaction systems, Theor. Comput. Sci. (2020), https://doi.org/10.1016/j.tcs.2020.09.001, in press.
[17] L. Brodo, R. Bruni, M. Falaschi, SOS rules for equivalences of reaction systems, in: M. Hanus, C.S. Coen (Eds.), Proc. of Functional and Constraint Logic Programming, WFLP 2020, in: LNCS, vol. 12560, Springer, 2020, in press.
[18] G. Pardini, R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, S. Tini, Compositional semantics and behavioural equivalences for reaction systems with restriction, Theor. Comput. Sci. 551 (2014) 1–21, https://doi.org/10.1016/j.tcs.2014.04.010.
[19] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, A. Troina, Bisimulations in calculi modelling membranes, Form. Asp. Comput. 20 (4) (2008) 351–377, https://doi.org/10.1007/s00165-008-0071-x.
[20] L. Cardelli, M. Tribastone, M. Tschaikowski, A. Vandin, Forward and Backward Bisimulations for Chemical Reaction Networks, Proc. of CONCUR 2015, vol. 42, Schloss Dagstuhl Publ., 2015, pp. 226–239.
[21] C. Bodei, L. Brodo, R. Bruni, A formal approach to open multiparty interactions, Theor. Comput. Sci. 763 (2019) 38–65, https://doi.org/10.1016/j.tcs.2019.01.033.
[22] C. Bodei, L. Brodo, R. Bruni, The link-calculus for open multiparty interactions, Inf. Comput. 275 (2020), https://doi.org/10.1016/j.ic.2020.104587.
[23] D. Sangiorgi, Introduction to Bisimulation and Coinduction, Cambridge University Press, USA, 2011.
[24] M. Hennessy, R. Milner, On observing nondeterminism and concurrency, in: ICALP'80, in: LNCS, vol. 85, Springer, 1980, pp. 299–309.
[25] M. Falaschi, G. Palma, A logic programming approach to reaction systems, in: DIP'20, in: OASIcs, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, vol. 86, 2020, pp. 6:1–6:15.
[26] M.S. Nobile, A.E. Porreca, S. Spolaor, L. Manzoni, P. Cazzaniga, G. Mauri, D. Besozzi, Efficient simulation of reaction systems on graphics processing units, Fundam. Inform. 154 (1–4) (2017) 307–321, https://doi.org/10.3233/FI-2017-1568.
[27] M.O. Rabin, D.S. Scott, Finite automata and their decision problems, IBM J. Res. Dev. 3 (2) (1959) 114–125, https://doi.org/10.1147/rd.32.0114.
[28] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, 3rd edition, Pearson international edition, Addison-Wesley, 2007.
[29] T. Helikar, N. Kochi, B. Kowal, M. Dimri, M. Naramura, S.M. Raja, V. Band, H. Band, J.A. Rogers, A comprehensive, multi-scale dynamical model of erbb receptor signal transduction in human mammary epithelial cells, PLoS ONE 8 (4) (2013) 1–9, https://doi.org/10.1371/journal.pone.0061757.
[30] T. Helikar, J. Konvalina, H. Jack, J.A. Rogers, Emergent decision-making in biological signal transduction networks, Proc. Natl. Acad. Sci. USA 105 (6) (2008) 1913–1918, https://doi.org/10.1073/pnas.0705088105.

[31] S. Azimi, C. Gratie, S. Ivanov, I. Petre, Dependency graphs and mass conservation in reaction systems, Theor. Comput. Sci. 598 (2015) 23–39, https://doi.org/10.1016/j.tcs.2015.02.014.
[32] S. Ivanov, V. Rogojin, S. Azimi, I. Petre, WEBRSIM: a web-based reaction systems simulator, in: C.G. Díaz, A. Riscos-Núñez, G. Paun, G. Rozenberg, A. Salomaa (Eds.), Enjoying Natural Computing - Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday, in: Lecture Notes in Computer Science, vol. 11270, Springer, 2018, pp. 170–181.
[33] M.S. Nobile, A.E. Porreca, S. Spolaor, L. Manzoni, P. Cazzaniga, G. Mauri, D. Besozzi, Efficient simulation of reaction systems on graphics processing units, Fundam. Inform. 154 (1–4) (2017) 307–321, https://doi.org/10.3233/FI-2017-1568.
[34] C. Ferretti, A. Leporati, L. Manzoni, A.E. Porreca, The many roads to the simulation of reaction systems, Fundam. Inform. 171 (1–4) (2020) 175–188, https://doi.org/10.3233/FI-2020-1878.
[35] Z. Shang, S. Verlan, I. Petre, G. Zhang, Reaction systems and synchronous digital circuits, Molecules 24 (10) (1961) 1–13, https://doi.org/10.3390/molecules24101961, 2019.
[36] P. Bottoni, A. Labella, G. Rozenberg, Networks of reaction systems, Int. J. Found. Comput. Sci. 31 (2020) 53–71, https://doi.org/10.1142/S0129054120400043.
[37] A. Bernini, L. Brodo, P. Degano, M. Falaschi, D. Hermith, Process calculi for biological processes, Nat. Comput. 17 (2) (2018) 345–373.
[38] A. Dovier, C. Piazza, E. Pontelli, G. Rossi, Sets and constraint logic programming, ACM Trans. Program. Lang. Syst. 22 (5) (2000) 861–931, https://doi.org/10.1145/365151.365169.
[39] J. Jaffar, M.J. Maher, Constraint logic programming: a survey, J. Log. Program. 19/20 (1994) 503–581, https://doi.org/10.1016/0743-1066(94)90033-7.
[40] M. Falaschi, C. Olarte, C. Palamidessi, A framework for abstract interpretation of timed concurrent constraint programs, in: PPDP'09, ACM, 2009, pp. 207–218.
[41] M. Falaschi, C. Olarte, C. Palamidessi, Abstract interpretation of temporal concurrent constraint programs, Theory Pract. Log. Program. 15 (3) (2015) 312–357, https://doi.org/10.1017/S1471068413000641.
[42] D. Chiarugi, M. Falaschi, C. Olarte, C. Palamidessi, Compositional modelling of signalling pathways in timed concurrent constraint programming, in: BCB'10, ACM, 2010, pp. 414–417.
[43] M. Falaschi, M. Gabbrielli, C. Olarte, C. Palamidessi, Dynamic slicing for concurrent constraint languages, Fundam. Inform. 177 (3–4) (2020) 331–357, https://doi.org/10.3233/FI-2020-1992.
[44] A. Ehrenfeucht, G. Rozenberg, Events and modules in reaction systems, Theor. Comput. Sci. 376 (1–2) (2007) 3–16, https://doi.org/10.1016/j.tcs.2007.01.008.