Executable Biochemical Space
for Specification and Analysis of Biochemical Systems

Matej Troják, David Šafránek, Luboš Brim

Systems Biology Laboratory, Masaryk University, Brno, Czech Republic

Jakub Šalagovič, Jan Červený

Global Change Research Centre AS CR, v. v. i., Brno, Czech Republic

Abstract

We present the second generation of a rule-based language called Biochemical Space Language (BCSL) that combines the advantages of different approaches and thus makes an effort to overcome several problems with existing solutions. The key aspect of the language is the level of abstraction it uses, which allows scalable and compact hierarchical specification of biochemical entities. This abstraction enables unique analysis techniques to reason about properties of models written in the language on the semantic and syntactic level.

Keywords: rule-based modelling, formal specification, static analysis

1 Introduction

Modelling complex systems in systems biology has to be conducted at several levels of abstraction that reflect well the known information [14]. At every level, the system has to be described rigorously in a formal language that allows avoiding misunderstood and ambiguous interpretations. The more complex the system is, the harder it is to describe it rigorously while not losing human-readability and compactness of the description at the same time. A modern biochemical system specification language that can be sufficiently employed in systems biology practice has to be *hierarchical* and *executable*. Hierarchical description allows expressing individual system components at different levels of detail. Since not all biochemical structures are known in detail, the language has to support the expression of partial knowledge. On the other end, executability allows automatic assigning the description with appropriate formal (mathematical or programming) structures that enable simulation and exhaustive analysis of desired properties or revealing bugs in the description.

Traditional approaches used to describe biochemical systems are: (i) a *chemistry approach* employing "mechanical" descriptions by chemical reactions or (ii) a *mathematical approach* using ordinary differential equations or other mathematical formalisms. The problem of both approaches is *scalability* in the description of the model and in its execution: even when the formulation of a model does not run into scalability issues, the execution or simulation might still be infeasible [24]. To that end, computer science offers a *computational approach* based on abstract languages with a variety of rigorous executable semantics. Relations among these approaches have been discussed in [4] and [12].

A promising computational approach is provided by *rule-based modelling* [7,9] and process-algebraic frameworks [4,5,23]. Rule-based models make a natural extension of the mechanical reaction-based models used in chemistry. Instead of operating with objects, rule-based frameworks operate with *types* that allow avoiding the combinatorial explosion that occurs when underlying objects are specified directly. The semantics of the

model is given in terms of *rules* defined on given types. An important advantage of rule-based approach is that mathematical models can be automatically generated from them. In particular, instead of relying on a single mathematical formalism, different mathematical models can thus be obtained for a given model (e.g., ODEs [3], PDEs [1], chemical master equation or continuous-time Markov chains [19,25], reaction-diffusion systems [26], etc.).

Although rule-based models make a great alternative to mathematical models, they are not yet sufficiently used in practice. The reason is that existing formalisms rely on cryptic (symbolic) syntax and they are limited to a specific subset of interactions or are too abstract: BNGL [9] and Kappa [7] target protein-protein binding; BioSPI [23] and SPiM [22] use very elemental asymmetric binary synchronisation primitives; BioPEPA [5] adapts process-algebraic framework to chemical reactions while relaxing the compactness of combinatorial interactions; Chromar [13] utilises functional programming. These languages can be thus understood as low-level formalisms that allow precise formal description and analysis of biological processes. Several high-level frameworks have been developed based on principles of these formalisms: rxncon [24] focuses on regulatory interactions and allows construction of rules from experimental evidence, LBS [21] and LBS-κ [20] enrich rule-based framework with modularity, PySB [17] embeds Kappa and BNGL into Python, MetaKappa [6] extends Kappa language by hierarchical inheritance of agent sites, BioCHAM [2] explicitly separates rules from their mathematical semantics. None of these frameworks provides a sufficiently universal solution for description and annotation of heterogeneous biophysical processes integrated at the cellular level. Apparently, different approaches need to be combined accordingly to make a universal hierarchical modelling and annotation base that supports executability. The work presented in [18] targets bringing annotation standards into rule-based frameworks.

On the other end, SBML multi [29] transfers rule-based description into a universal XML format that fixes the hierarchical structure of objects and modularity of rules. It moves the rule-based paradigm towards a standard technique of describing biological systems. However, it does not directly solve the executability and advanced analysis issues that make an important aspect of rule-based frameworks.

Our long-term aim is the development of a general modelling framework [16,27]. Together with general annotation format Biochemical Space [15], it respects the need for maintaining existing ODE models but allows to align them with a mechanistic rule-based description that is understandable by biologists, compact in size, executable in terms of allowing basic analysis tasks ensuring consistency of the description, and provides links to existing bioinformatics annotation databases. Such a comprehensive solution allows supporting modellers effort in building mathematical models that have clear biochemical meaning and can be easily integrated. Moreover, mechanistic descriptions can be later used as computational models having all advantages of rule-based modelling. To that end, we have pioneered an idea of combining advantages of rule-based modelling with the simplicity of chemical reactions by introducing the first prototype of a high-level rule-based language called *Biochemical Space Language* (BCSL), introduced in [8]. The language has been defined at the top of Kappa. BCSL aims at higher-level abstraction than Kappa that focuses on morphisms between protein binding sites. Therefore the Kappa-based formulation of BCSL has limited expressiveness and does not fit well the aims of our framework. Additionally, Kappa does not provide hierarchical description which is one of the key aspects of BCSL.

In this paper, BCSL is redefined and significantly improved with respect to the primary prototype presented in [8]: (i) hierarchical and composable object types and rules are defined without the need to encode them in an existing rule-based framework thus avoiding any loss of information, (ii) executable semantics of rules is defined directly at the level of the language thus making a base for unique analysis tasks specific for the considered level of abstraction, (iii) software tool is available to maintain and analyse BCSL specifications – BCSgen [1] . The new version of BCSL emphasises the following aspects: (i) *human-readability* (easy to read, write, and maintain), (ii) *executability* (formal executable semantics is defined allowing efficient static analysis and consistency checking), (iii) *universality* (principally different cellular processes can be sufficiently combined in a single specification), (iv) *scalability* (combinatorial explosion of the description is avoided), (v) *hierarchy* (object types are described hierarchically allowing compositional assembly from simpler structures). Moreover, we provide several static analysis techniques which take the advantage from the specific level of abstraction. They are aimed primarily at consistency checking, model reduction and reachability analysis. Particularly, *rule redundancy elimination* allows detecting unnecessary rules in the models, *context-based reduction* and *static non-reachability analysis* uniquely deal with non-reachability in terms of preventing expensive transition system enumeration in cases when it is not necessarily needed. These techniques are demonstrated on a model of *fibroblast growth factor* (FGF) signalling pathway and show practical impact in the field of static analysis.

---

[1] https://github.com/sybila/BCSgen

2 Formal definition of Biochemical Space Language

In this section, we formally define Biochemical Space Language. At first, we define all the required objects (so called *agents*) and interactions among them (so called *rules*; for an example, see Figure 1), then we define syntax of the language and semantics of the BCSL models.

KaiC protein   KaiBC dimer

$$S\{u\} :: KaiC :: KaiBC :: cyt \Rightarrow S\{p\} :: KaiC :: KaiBC :: cyt$$

Serine residue
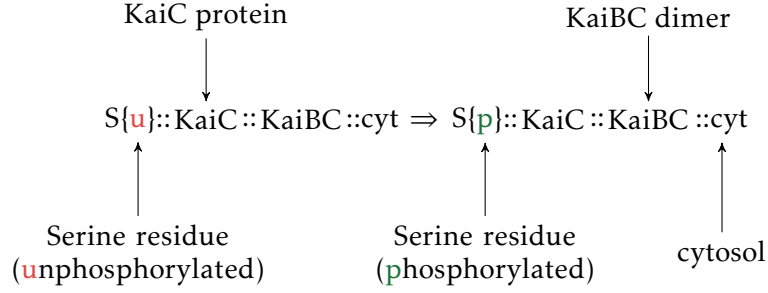(unphosphorylated)   Serine residue
(phosphorylated)   cytosol

Fig. 1. An example of a rule. The rule describes the change of serine (S) amino acid residue from an unphosphorylated to phosphorylated state. Additionally, such phosphorylation can happen only when the serine is part of a KaiC protein, which occurs inside a protein complex of KaiC and KaiB proteins. The entire process is allowed only inside of cytosol (cyt) compartment.

2.1 Formal preliminaries

Before we proceed, we provide some basic definitions and notations in order to build the formal definition for the language.

Definition 2.1 (*Multiset*) *Multiset* Ω is a pair $(\mathcal{A}, \mathfrak{m})$ where $\mathcal{A}$ is a set and $\mathfrak{m} : \mathcal{A} \to \mathbb{N}$ is a function from $\mathcal{A}$ to the set of natural numbers. The set $\mathcal{A}$ is called the *reference set* of elements. For each element a in $\mathcal{A}$ the *multiplicity* (that is, number of occurrences) of a is the number $\mathfrak{m}(a)$.

Notation 2.2

- *Let $\mathcal{S}$ be a set. By $\Omega^{\mathcal{S}}$ we denote the set of all possible finite multisets $(\mathcal{A}, \mathfrak{m})$ such that $\mathcal{A} \subseteq \mathcal{S}$.*
- *Let $O = (o_1, \ldots, o_n)$ be a tuple.*
  - *By $\Omega(O)$ we denote a multiset constructed from tuple O.*
  - *By $\sigma(O)$ we denote a set of all possible permutations of length n of the tuple O.*
- *By $|Y|$ we denote* (i) *dimension of tuple Y or* (ii) *cardinality of (multi)set Y.*

Definition 2.3 (*Labelled transition system*) *Labelled transition system* (LTS) $\mathcal{L}$ is a quadruple (S, A, T, s_0) where S is a set of states, A is a set of labels, $T \subseteq S \times A \times S$ is a transition relation, and $s_0 \in S$ is an initial state.

Definition 2.4 (*Path in LTS*) Let $\mathcal{L} = (S, A, T, s_0)$ be an LTS. We define *path* as a sequence of states $s_1 s_2 s_3 \ldots$ such that $\forall s_i, s_{i+1} : (s_i, a, s_{i+1}) \in T$ for some $a \in A$.

Definition 2.5 (*Tuples concatenation*) Let $X = (x_1, \ldots, x_n), Y = (y_1, \ldots, y_m)$ be two tuples for some $n, m \in \mathbb{N}$. *Concatenation* of two tuples, written $X + Y$, is defined as: $X + Y = (x_1, \ldots, x_n, y_1, \ldots, y_m)$.

Definition 2.6 (*Sum of concatenations*) Let $T = (T_1, T_2, \ldots, T_n)$ be sequence of tuples for some $n \in \mathbb{N}$. *Concatenation* of sequence of tuples $+_{i=1}^{n} T_i$ is defined as: $+_{i=1}^{n} T_i = T_1 + T_2 + \ldots + T_n$

2.2 Objects definition

Let $\mathcal{N}_A, \mathcal{N}_T, \mathcal{N}_\delta, \mathcal{N}_c$ be mutually exclusive finite sets of atomic names, structure names, states, and compartments respectively. Moreover, ε is a reserved symbol and does not belong to any of these sets.

For better readability, we provide examples of syntax for the most important objects with their definitions. The formal definition of syntax and the relation to the objects are given below (Sections 2.3 and 2.4).

2.2.1 Signature

Definition 2.7 (*Signature*) *Atomic signature* is a function $\Sigma_A : \mathcal{N}_A \to 2^{\mathcal{N}_\delta}$ that associates each atomic name to a set of state names. Similarly, *structure signature* is a function $\Sigma_T : \mathcal{N}_T \to 2^{\mathcal{N}_A}$ that associates each structure name to a set of atomic names.

Signatures define a set of allowed states for an atomic name and an allowed set of atomic names for a structure name. For example, $\left\{ S \rightarrow \{u,p\}, Q \rightarrow \{a,i\} \right\}$ is an atomic signature and $\left\{ KaiC \rightarrow \{S,Q\}, KaiB \rightarrow \emptyset \right\}$ is a structure signature.

2.2.2 Atomic agent

Definition 2.8 (*Atomic agent*) An *atomic agent* A is a pair (η, δ) where $\eta \in \mathcal{N}_A$ is a name and $\delta \in \mathcal{N}_\delta \cup \{\varepsilon\}$ is a state. The name and the state of the agent A is usually denoted by $\eta(\mathsf{A})$ and $\delta(\mathsf{A})$, respectively.

Atomic agents are the simplest objects used for describing biological entities. Each atomic agent has its name and state. Allowed set of admissible states for the atomic agent (with additional empty ε state) is given by signature $\Sigma_{\mathsf{A}}(\eta)$.

Definition 2.9 (*Equality relation of atomic agents*) Let A, A' be atomic agents. A is *equal* to A', written $\mathsf{A} = \mathsf{A}'$, *iff* $\eta(\mathsf{A}) = \eta(\mathsf{A}') \wedge \delta(\mathsf{A}) = \delta(\mathsf{A}')$.

Intuitively, the defined equality on atomic agents is an equivalence relation.

Notation 2.10 *We use the symbol $\mathbb{A}$ to denote the universe of all possible atomic agents.*

Atomic agents are usually used to express small biological entities which can change their state, for example, amino acids, small inorganic molecules, etc. Examples of atomic agents are $\mathsf{A}_1 = (S, u)$, written as $S\{u\}$, and $\mathsf{A}_2 = (Q, \varepsilon)$, written as $Q\{\varepsilon\}$. Note the meaning of ε is the state is unknown or not important to be considered in a given context.

Definition 2.11 (*Compatibility of atomic agents*) Let A_1, A_2 be atomic agents. The agent A_1 is *compatible with* agent A_2, written $\mathsf{A}_1 \lhd \mathsf{A}_2$, if either $\mathsf{A}_1 = \mathsf{A}_2$ or $\eta(\mathsf{A}_1) = \eta(\mathsf{A}_2) \wedge \delta(\mathsf{A}_1) = \varepsilon$.

Compatibility of atomic agents is a key property defined between agents. An agent is compatible with anothet agent if they have the same name and they are in the same state or the first agent is in the unknown state. It provides a formal way to compare which agent is more detailed, i.e. its state is more specified.

Definition 2.12 (*Fully specified atomic agent*) Let $\mathsf{A} \in \mathbb{A}$ be an atomic agent. We say the agent A is *fully specified*, written $\triangle \mathsf{A}$, *iff* $\forall \mathsf{A}' \in \mathbb{A}$ such that $\mathsf{A}' \neq \mathsf{A} : \neg(\mathsf{A}' \lhd \mathsf{A})$.

2.2.3 Structure agent

Definition 2.13 (*Structure agent*) We define a *structure agent* T as a pair (η, γ) where $\eta \in \mathcal{N}_T$ is a name and $\gamma \subseteq \mathbb{A}$ is a set of atomic agents called partial composition such that $\forall \mathsf{A}, \mathsf{A}' \in \gamma : \eta(\mathsf{A}) \neq \eta(\mathsf{A}')$. The name and the partial composition of the agent T is usually denoted by $\eta(\mathsf{T})$ and $\gamma(\mathsf{A})$, respectively.

A structure agent represents a biochemical object that is composed of several known atomic agents while we know that a composition is abstract and not necessarily complete. To incorporate this kind of abstraction into our language, a structure agent is defined to be labelled with a unique name and a set of atomic agents. This set is restricted according to the given structure signature with the same name as the structure agent.

Definition 2.14 (*Equality relation of structure agents*) Let T, T' be structure agents. T is *equal* to T', written $\mathsf{T} = \mathsf{T}'$, *iff* $\eta(\mathsf{T}) = \eta(\mathsf{T}') \wedge \gamma(\mathsf{T}) = \gamma(\mathsf{T}')$.

Intuitively, the defined equality on structure agents is an equivalence relation. The key construct of a structure agent is *partial composition* defined as a set of atomic agents which are considered to be relevant parts of the structure agent. We allow this set to be empty with the meaning of a biological structure for which we do not know its composition.

Notation 2.15 *We use symbol $\mathbb{T}$ to denote the universe of all possible structure agents.*

A typical example of a structure agent is a protein where the atomic agents are amino acids that are of interest in the particular setting. Imagine that in our modelled system only three out of a few hundred amino acids are able to undergo some post-translational modifications, such as phosphorylation, metylation etc. It is suitable to model only these three amino acids instead of entire primary structure of the protein. Examples of structure agent are $\mathsf{T}_1 = (K, \{(S,p),(Q,i)\})$, written as $K(S\{p\}, Q\{i\})$, and $\mathsf{T}_2 = (K, \{(Q,a)\})$, written as $K(Q\{a\})$.

We define difference on the level of partial compositions of structure agents, which is necessary for definition of semantics below.

Definition 2.16 (*Difference of partial compositions*) Let γ, γ' be partial compositions. We define *difference of partial compositions* $\gamma \ominus \gamma' = \{\mathsf{A} \mid \mathsf{A} \in \gamma \wedge \mathsf{A} \notin \gamma \cap \gamma'\}$ where $\gamma \cap \gamma' = \{\mathsf{A} \mid \mathsf{A} \in \gamma \wedge \exists \mathsf{A}' \in \gamma' : \eta(\mathsf{A}') = \eta(\mathsf{A})\}$.

Definition 2.17 (*Compatibility of structure agents*) Let T_1, T_2 be structure agents. The agent T_1 is *compatible with* agent T_2, written $T_1 \lhd T_2$, *iff* either $T_1 = T_2$ or $\eta(T_1) = \eta(T_2) \wedge \forall A_1 \in \gamma(T_1) \, \exists A_2 \in \gamma(T_2) : A_1 \lhd A_2$.

Structure agents are compatible if it is possible to create pairs from atomic agents of composition of the first agent with the second ones such that these atomic agents are all unique. For such pairs, the agents in each pair must be compatible. It provides a formal way to compare which agent is more specified, i.e. particular states of atomic agents in partial composition are given or not.

Definition 2.18 (*Fully specified structure agent*) Let $T \in \mathbb{T}$ be a complex agent. We say the agent T is *fully specified*, written $\triangle T$, *iff* $\forall T' \in \mathbb{T}$ such that $T' \neq T : \neg(T' \lhd T)$.

2.2.4 Complex agent

A complex agent represents a non-trivial composite biochemical object that is inductively constructed from already known biological objects. In rule-based languages, this is usually defined by introducing bonds between individual biochemical objects. In BCSL we abstract from the detailed specification of bonds and we rather assume a complex as a coexistence of certain objects in a particular group. Moreover, a complex agent resides in a compartment which gives it a spatial position.

Definition 2.19 (*Complex agent*) We define a *complex agent* X as a pair (μ, com) where $\mu \in (\mathbb{A} \cup \mathbb{T})^n$ is a sequence of agents, $\text{com} \in \mathcal{N}_c$ is a compartment, and $n \in \mathbb{N}$. The sequence and the compartment of the agent X is usually denoted by $\mu(X)$ and $\text{com}(X)$, respectively.

The key element of a complex agent is *sequence* inductively constructed from existing agents. In contrast to partial composition in structure agent, we allow replication at the level of sequence (an agent of a certain name can appear more than once in a sequence). The order in the sequence is necessary to uniquely identify agents which are equal. On the other hand, when comparing two sequences, we do it regardless the order.

Definition 2.20 (*Equality relation of complex agents*) Let X, X' be complex agents. X is *equal* to X', written $X = X'$, *iff* $\text{com}(X) = \text{com}(X') \wedge \Omega(\mu(X)) = \Omega(\mu(X'))$.

Intuitively, the defined equality on complex agents is an equivalence relation. Example of a complex agent is $X = \big(((K, \{(S, p), (Q, i)\}), (S, p)), cell\big)$, written as $K(S\{p\}, Q\{i\}).S\{p\} :: cell$.

Notation 2.21 *We use the symbol $\mathbb{X}$ to denote the universe of all possible complex agents.*

The complex agents encapsulate other agents – an atomic or a structure agent cannot exist on its own (the case when only one item is in its sequence can occur). This guarantees each atomic and structure agent has indirectly given spatial location – the compartment.

Definition 2.22 (*Compatibility of complex agents*) Let X_1, X_2 be complex agents. The complex agent X_1 is *compatible with* complex agent X_2, written $X_1 \lhd X_2$, *iff* either $X_1 = X_2$ or $\text{com}(X_1) = \text{com}(X_2) \wedge \exists \mu' \in \sigma(\mu(X_2))$ such that $\forall i \in [1, n] : \mu_i(X) \lhd \mu_i'$, where n is length of sequence which is the same for both sequences.

Complex agents are compatible if there exists a permutation of the sequence of the first agent such that individual agents on the same position in both sequences are compatible. It provides a formal way to compare which agent is more specified.

Definition 2.23 (*Fully specified complex agent*) Let $X \in \mathbb{X}$ be a complex agent. We say the agent X is *fully specified*, written $\triangle X$, *iff* $\forall X' \in \mathbb{X}$ such that $X' \neq X : \neg(X' \lhd X)$.

It worth noting that the complexes have no binding topology. While it provides many advantages, specifically when it comes to combinatorial explosion, it also has several drawbacks. The most important one is that we are not able to express structural modifications on the level of complexes. These have to be encoded using states.

2.2.5 Rule

Let us have a simple example of a rule:

$$K(S\{u\}).B(\varnothing) :: cyt \Rightarrow K(S\{p\}) :: cyt + B(\varnothing) :: cyt.$$

This rule dissociates a complex of K and B (both structure agents) to two separate agents while the structure agent K is changing the state of its atomic agent S from u to p. In order to describe the rule formally, we need to capture the relation between so-called *left-hand side* (the part *before* $\Rightarrow$ symbol) and *right-hand side* (the part *after* $\Rightarrow$ symbol). It is achieved by indexing the individual positions in the rule and creating index maps between them.

Definition 2.24 (*Rule*) We define a *rule* R as a quintuple $(\chi, \omega, \iota, \varphi, \psi)$ where:

- $\chi \in \mathbb{X}^n$ is a sequence of complex agents,
- $\omega \in (\mathbb{A} \cup \mathbb{T})^m$ is a sequence of atomic and structure agents,
- $\iota \in \{0, \dots, n\}$ is an index determining the end of the *left-hand side* (*LHS*) of χ,
- $\varphi \in \mathbb{N}^m$ is an index map from ω to χ,
- $\psi \in ((\{-\} \cup \mathbb{N})^2)^n$ is an index map from *LHS* to *RHS*

where $n, m \in \mathbb{N}$, $LHS = (\chi_1, \dots, \chi_\iota)$ is the *left-hand side*, and $RHS = (\chi_{\iota+1}, \dots, \chi_n)$ is the *right-hand side*.

The reason for this particular definition is that it is necessary to capture the relationship between the left-hand side and the right-hand side of the rule. This is done by enumerating all atomic and structure agents ω from sequence of complex agents χ. The index map ψ between the agents in ω determines pairs of agents from the left-hand side and the right-hand side which correspond to each other. It is possible that there are agents which do not have a pair (denoted by $-$) in the situation when the rule is modelling *inflow* from (resp. *outflow* to) the system. Another index map φ serves for relating agents from ω back to the original sequence of complexes χ. Finally, by index ι we determine the end of the left-hand side of the rule. Note the index is zero in the situation when there are no agents on the left-hand side.

Notation 2.25 *We use symbol $\mathbb{R}$ to denote the universe of all possible rules.*

Example of a rule is R $= (\chi, \omega, \iota, \varphi, \psi)$ where:

- $\chi = \begin{bmatrix} \big(((K, \{(S, u)\}), (B, \emptyset)), cyt\big), \\ \big(((C, \emptyset), (D, i)), cyt\big), \\ \big(((A, \varepsilon)), cyt\big), \\ \big(((K, \{(S, p)\}), (B, \emptyset), (C, \emptyset)), cyt\big), \\ \big(((D, a), (A, \varepsilon)), cyt\big), \\ \big(((H, u)), cyt\big) \end{bmatrix}$

- $\omega = \begin{bmatrix} (K, \{(S, u)\}), (B, \emptyset), (C, \emptyset), \\ (D, i), (A, \varepsilon), (K, \{(S, p)\}), \\ (B, \emptyset), (C, \emptyset), (D, a), (A, \varepsilon), (H, u) \end{bmatrix}$
- $\iota = 3$
- $\varphi = (2, 4, 5, 8, 10, 11)$
- $\psi = [(1, 6); (2, 7); (3, 8); (4, 9); (5, 10); (-, 11)]$

written as:

$K(S\{u\}).B(\emptyset) :: cyt + C(\emptyset).D\{i\} :: cyt + A\{\varepsilon\} :: cyt \Rightarrow K(S\{p\}).B(\emptyset).C(\emptyset) :: cyt + D\{a\}.A\{\varepsilon\} :: cyt + H\{u\} :: cyt$

Not every rule makes sense. For example, a rule where not a single agent is changed or a rule where the relation between the left-hand and the right-hand side would not be clear. In order to avoid such cases we need to specify when a rule is *well-formed*, i.e. it makes sense semantically.

Definition 2.26 (*Well-formed rule*) Let R be a rule and $i, j \in \mathbb{N}$. We say the rule R $= (\chi, \omega, \iota, \varphi, \psi)$ is *well-formed* if all the following conditions hold:

(i) at least one of conditions holds:
    (a) $\exists (i, j) \in \psi : \omega_i \neq \omega_j$,
    (b) $|LHS(R)| \neq |RHS(R)|$,
    (c) $\exists i \in [1, \iota] : \mathsf{com}(\chi_i) \neq \mathsf{com}(\chi_{\iota+i})$;

(ii) $\forall (i, j) \in \psi : \eta(\omega_i) = \eta(\omega_j)$;

(iii) $\forall (-, i) \in \psi : \triangle \omega_i$.

A rule is well-formed if it holds conditions given in Definition 2.26. The conditions basically claim that an agent has to change during the rule application. This is ensured by condition (i), where there are three options: (a) at least one pair of agents from *LHS* and *RHS* of the rule is different; (b) the lengths of the *LHS* and *RHS* are different, i.e. either a new agent is created or complex is formed/dissociated; (c) a compartment is changed. Any combination of these sub-conditions is allowed. The second condition (ii) guarantees that the pairs of structure and atomic agents in ω of the rule have the same name. Please note the conditions (i) and (ii) do not apply to those agents in ω which do not have a pair on the other side of the rule. Finally, the condition (iii) claims that if there is an agent which does not have defined a pair via index map ψ (denoted by $-$), it is required to be a fully specified agent (but only in case of agent creation, it is not necessary for agent degradation).

6

2.3 Syntax

In this section, we define the syntax for the language, i.e. how we usually write it in order to make the notation easily writeable and readable. It corresponds to the examples given while defining agents and rules above.

Definition 2.27 (*Grammar*)

Atomic expression	Structure expression	Complex expression
$\alpha ::= \eta\{s\} \mid \eta\{\varepsilon\}$	$\tau ::= \eta(\gamma) \mid \eta(\varnothing)$	$\Gamma ::= \beta_1 . \ldots . \beta_k :: c$
$\eta ::= n \in \mathcal{N}_A$	$\gamma ::= \alpha_1,\ldots,\alpha_k$	$\beta_i ::= \alpha \mid \tau$
$s ::= n \in \mathcal{N}_\delta$	$\eta ::= n \in \mathcal{N}_T$	$c ::= n \in \mathcal{N}_c$

Rule expression	$\rho ::= \Gamma_1 + \ldots + \Gamma_n \Rightarrow \Gamma_{n+1} + \ldots + \Gamma_m$

where $m, n \in \mathbb{N}_0 \wedge m > n$ and $k \in \mathbb{N}$.

2.4 Translation function

Once we defined BCSL agents and rules and syntax for the language, we need to connect them in order to give semantic meaning to a model written in the syntax. For this purpose, we define translation function F (Definition 2.28). It is defined recursively according to the expression given as an argument.

Definition 2.28 (*Translation function*) We define *translation function* F according to the expression given in double square brackets $[\![\ldots]\!]$ as follows:

$$
\begin{aligned}
F[\![\eta\{\varepsilon\}]\!] &= (\eta,\varepsilon) \in \mathbb{A} \\
F[\![\eta\{s\}]\!] &= (\eta,s) \in \mathbb{A} \\
F[\![\eta(\varnothing)]\!] &= (\eta,\emptyset) \in \mathbb{T} \\
F[\![\eta(a_1,\ldots,a_k)]\!] &= \left(\eta,\{ F[\![a_1]\!],\ldots,F[\![a_k]\!] \}\right) \in \mathbb{T} \\
F[\![\alpha_1 . \ldots . \alpha_k :: c]\!] &= \left((F[\![\alpha_1]\!],\ldots,F[\![\alpha_k]\!]), c \right) \in \mathbb{X} \\
F[\![\Gamma_1 + \ldots + \Gamma_n \Rightarrow \Gamma_{n+1} + \ldots + \Gamma_m]\!] &= (\chi,\omega,\iota,\varphi,\psi) \in \mathbb{R} \text{ such that:}
\end{aligned}
$$

- $\chi = \left(F[\![\Gamma_1]\!],\ldots,F[\![\Gamma_n]\!],F[\![\Gamma_{n+1}]\!],\ldots,F[\![\Gamma_m]\!]\right),$

- $\omega = +_{i=1}^{|\chi|} \mu(\chi_i),$

- $\iota = n,$

- $\varphi = (J_1,\ldots,J_m)$ where $J_k = \sum_{i=1}^{k} |\mu(\chi_i)|,$

- $\psi = \{ (i,j) \mid i \in [1,\varphi_\iota] \wedge j \in [\varphi_\iota+1,|\omega|] \wedge |i-j| = \varphi_\iota \} \cup$
$\{ (i,-) \mid i \in [k,\varphi_\iota] \wedge k = |\omega| - \varphi_\iota + 1 \} \cup$
$\{ (-,j) \mid j \in [k,|\omega|] \wedge k = 2 \times \varphi_\iota + 1 \}$

where ψ is defined together with an ordering such that symbol $'-' > k$ for every $k \in \mathbb{N}$ and all descending intervals in definition of ψ are ignored.

Note that the translation function works *only* on expressions defined in Definition 2.27. The function recursively creates objects from given expressions. Every rule expression is first decomposed to LHS and RHS, and consequently each agent expression is translated to an object. The appropriate index maps are created from sequence of complexes χ and sequence of atomic and structure agents ω.

2.5 BCSL model

We proceed to the BCSL model definition. We always consider an initialised model, which means the definition contains an initial state of the system (a solution, Definition 2.29). The definition of BCSL model also contains rules and signatures.

Definition 2.29 (*Solution*) *Solution* is a multiset $\mathcal{S} \in \Omega^{\mathbb{X}}$ such that $\mathcal{A}$ is the reference set of $\mathcal{S}$ and $\forall X \in \mathcal{A} : \triangle X$.

Definition 2.30 (*BCSL model*) We define *BCSL model* $\mathcal{M}$ as a quadruple $(\mathcal{R}, \Sigma_A, \Sigma_T, \mathcal{S})$ where $\mathcal{R}$ is a set of rules, Σ_A is an atomic signature, Σ_T is a structure signature, and $\mathcal{S}$ is an initial solution.

A BCSL model is formed by a set of rules $\mathcal{R}$, which define the behaviour of the model. The initial solution $\mathcal{S}$ defines the state of the model in the beginning. Atomic signature Σ_A defines allowed states for all atomic agents used in the rules. Finally, structure signature Σ_T defines allowed atomic agents for all structure agents used in the rules.

2.6 Matching

At this point, we define matching, which will be used in the definition of semantics for a BCSL model $\mathcal{M}$.

Definition 2.31 (*Matching*) Let $R = (\chi, \omega, \iota, \varphi, \psi)$, $r = (\chi', \omega', \iota', \varphi', \psi')$ be two rules, $\mathcal{S} \in \Omega^{\mathbb{X}}$ be a solution, and $i, j \in \mathbb{N}$. Let $\models\, \subseteq \mathbb{R} \times \Omega^{\mathbb{X}} \times \mathbb{R}$ be the *matching* relation such that a tuple $(R, \mathcal{S}, r) \in\, \models$, written $R \models_r \mathcal{S}$, iff

(i) $\iota = \iota' \wedge \varphi = \varphi' \wedge \psi = \psi'$,

(ii) $|\chi| = |\chi'| \wedge |\omega| = |\omega'|$,

(iii) $\forall i \in [1, |\chi|] : \chi'_i \lhd \chi_i$,

(iv) $\Omega(LHS(r)) = \mathcal{S}$,

(v) $\forall (i, j) \in \psi :$

    (a) $\omega_i \in \mathbb{A} \Rightarrow \begin{cases} \omega'_i = \omega'_j & if \quad \omega_i = \omega_j \\ \omega_i = \omega'_i \wedge \omega_j = \omega'_j & if \quad \omega_i \neq \omega_j \end{cases}$

    (b) $\omega_i \in \mathbb{T} \Rightarrow \gamma(\omega'_i) \ominus \gamma(\omega_i) = \gamma(\omega'_j) \ominus \gamma(\omega_j)$.

Remark 2.32 Note the rule r from the tuple $(R, \mathcal{S}, r) \in\, \models$ is so-called *reaction*, which is characterised as an instance of the rule R. For every rule in a model, it is possible to enumerate all potential reactions and this way convert a rule-based model to a reaction-based model.

2.7 Semantics

Definition 2.33 (*Replacement*) Let $\rightarrow\, \subseteq \Omega^{\mathbb{X}} \times \mathbb{R} \times \Omega^{\mathbb{X}}$ be the *replacement* relation such that a tuple $(\mathcal{S}, R, \mathcal{S}') \in\, \rightarrow$, written $\mathcal{S} \rightarrow_R \mathcal{S}'$, iff $\exists r \in \mathbb{R}\ \exists x \subseteq \mathcal{S}$ such that $R \models_r x \wedge\ \mathcal{S}' \setminus (\mathcal{S} \setminus x) = \Omega(RHS(r))$.

Replacement relation defines how a solution is transformed according to a given rule. For a BCSL model $\mathcal{M}$, rules yield a labelled transition system $LTS(\mathcal{M})$ between solutions containing an edge $\mathcal{S} \rightarrow_R \mathcal{S}'$. Note that we can achieve the equivalent behaviour if we first generate all possible reactions from the rules and apply replacement with them instead (a rule is just a generalised set of reactions).

3 Syntactic extensions

In this section, we define several syntactic extensions which increase the readability of the rule expressions. Note that each rule expression in an extended form can always be translated to basic form defined above (Section 2.3). All rule expressions containing the following extensions must be converted to basic form before the semantics can be applied. For better demonstration, we provide a running example, which will go through all syntactic extensions (Running example 3.1). Please note there is no biological sense of the example model, its only purpose is to effectively demonstrate all defined syntactic extensions.

Running example 3.1 (*The example model $\mathcal{M}$*)

(i) $KaiC(S\{u\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt \Rightarrow KaiC(S\{p\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt$

(ii) $KaiC(S\{u\}, T\{\varepsilon\}).KaiB(\varnothing) :: cyt \Rightarrow KaiC(S\{p\}, T\{\varepsilon\}).KaiB(\varnothing) :: cyt$

(iii) $KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt + KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt + KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt \Rightarrow KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt$

(iv) $KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt \Rightarrow KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt + KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt + KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt$

$$\Sigma_A = \left\{ S \rightarrow \{u, p\}, T \rightarrow \{a, i\} \right\}$$
$$\Sigma_T = \left\{ KaiC \rightarrow \{S, T\}, KaiB \rightarrow \varnothing \right\}$$

We omit the initial state definition just for simplicity of the example since all the extensions concern only rule expressions.

3.1 Partial composition context elimination

It is possible to omit all atomic expressions with unspecified state ε from partial compositions of structure agents (Running example 3.2). Such agent expressions do not give any additional information and whole partial composition can be reconstructed from the given signature.

8

Running example 3.2 (*The example model $\mathcal{M}$*)

(i) $KaiC(S\{u\}).KaiC(\varnothing).KaiC(\varnothing) :: cyt \Rightarrow KaiC(S\{p\}).KaiC(\varnothing).KaiC(\varnothing) :: cyt$

(ii) $KaiC(S\{u\}).KaiB(\varnothing) :: cyt \Rightarrow KaiC(S\{p\}).KaiB(\varnothing) :: cyt$

(iii) $KaiC(\varnothing) :: cyt + KaiC(\varnothing) :: cyt + KaiC(\varnothing) :: cyt \Rightarrow KaiC(\varnothing).KaiC(\varnothing).KaiC(\varnothing) :: cyt$

(iv) $KaiC(\varnothing).KaiC(\varnothing).KaiC(\varnothing) :: cyt \Rightarrow KaiC(\varnothing) :: cyt + KaiC(\varnothing) :: cyt + KaiC(\varnothing) :: cyt$

Additionally, this extension can go even further by omitting the ($\varnothing$) part from structure agents completely (Running example 3.3). Since we have the structure signature Σ_T defined, we can unambiguously determine which names belong to structure agents and this syntactic part can be easily reconstructed.

Running example 3.3 (*The example model $\mathcal{M}$*)

(i) $KaiC(S\{u\}).KaiC.KaiC :: cyt \Rightarrow KaiC(S\{p\}).KaiC.KaiC :: cyt$

(ii) $KaiC(S\{u\}).KaiB :: cyt \Rightarrow KaiC(S\{p\}).KaiB :: cyt$

(iii) $KaiC :: cyt + KaiC :: cyt + KaiC :: cyt \Rightarrow KaiC.KaiC.KaiC :: cyt$

(iv) $KaiC.KaiC.KaiC :: cyt \Rightarrow KaiC :: cyt + KaiC :: cyt + KaiC :: cyt$

This syntactic extension brings a lot of readability to the syntax while preserving all information in the context of the model $\mathcal{M}$.

3.2 Complex signature

We extend the model definition by complex signature Σ_X (Running example 3.4). In this signature, there are defined aliases for valid complex expressions. Then, the original complex expressions are substituted by the aliases.

Running example 3.4 (*The example model $\mathcal{M}$*)

Definition of complex signature $\Sigma_X = \left\{ \begin{array}{l} KaiC3 :: cyt \rightarrow KaiC.KaiC.KaiC :: cyt, \\ KaiBC :: cyt \rightarrow KaiC.KaiB :: cyt \end{array} \right\}$

(i) $KaiC(S\{u\}).KaiC.KaiC :: cyt \Rightarrow KaiC(S\{p\}).KaiC.KaiC :: cyt$

(ii) $KaiC(S\{u\}).KaiB :: cyt \Rightarrow KaiC(S\{p\}).KaiB :: cyt$

(iii) $KaiC :: cyt + KaiC :: cyt + KaiC :: cyt \Rightarrow KaiC3 :: cyt$

(iv) $KaiC3 :: cyt \Rightarrow KaiC :: cyt + KaiC :: cyt + KaiC :: cyt$

The usage of the complex signature has its limitations. Once a context is specified, the alias cannot be used. We will resolve this problem in the following extensions.

3.3 Directions

We allow rule expressions to be bi-directional – it is just a shortcut for two rule expressions and it can be converted to the basic rule expression form. A rule expression $\rho : l \Leftrightarrow r$ can be written as two rule expressions $\rho_1 : l \Rightarrow r$ and $\rho_2 : r \Rightarrow l$ (Running example 3.5).

Running example 3.5 (*The example model $\mathcal{M}$*)

(i) $KaiC(S\{u\}).KaiC.KaiC :: cyt \Rightarrow KaiC(S\{p\}).KaiC.KaiC :: cyt$

(ii) $KaiC(S\{u\}).KaiB :: cyt \Rightarrow KaiC(S\{p\}).KaiB :: cyt$

(iii) $KaiC :: cyt + KaiC :: cyt + KaiC :: cyt \Leftrightarrow KaiC3 :: cyt$

Definition of rules (iii) and (iv) from Running example 3.4 was replaced by one bi-directional rule (iii) in Running example 3.5.

3.4 Stoichiometry

For a rule expression of form:

$$\beta_1 :: c + \beta_2 :: c + \ldots + \beta_n :: c \Rightarrow \beta_1.\beta_2.\,\ldots\,.\beta_n :: c$$

we can reorder both sides such that we get non-crossing partition $\mathcal{P} = B_1/B_2/\ldots/B_k$ with $k \leq n$ from its indices $[1,\ldots,n]$ such that: $\forall B \in \mathcal{P} \; \forall \beta, \beta' \in B : \beta = \beta'$ and $\forall B, B' \in \mathcal{P} \; \forall \beta \in B \; \forall \beta' \in B' : \beta \neq \beta'$ such that $B \neq B'$.

For the left-hand side $\beta_1 :: \mathsf{c} + \beta_2 :: \mathsf{c} + \ldots + \beta_n :: \mathsf{c}$ of the reordered rule expression we can replace all rule expressions $[\beta_i, \ldots, \beta_j]$ which belong to the same non-crossing partition B by notation '$k\ \beta$', where β is a representative from $\beta_i, \ldots, \beta_j$ (they are all equivalent) and k is the number of the expressions in partition B (Running example 3.6). Note that this process is fully reversible – we can simply enumerate all expressions for each partition.

Running example 3.6 (*The example model* $\mathcal{M}$)
Definition of rule expressions:

  (i) $KaiC(S\{u\}).KaiC.KaiC :: cyt \Rightarrow KaiC(S\{p\}).KaiC.KaiC :: cyt$

 (ii) $KaiC(S\{u\}).KaiB :: cyt \Rightarrow KaiC(S\{p\}).KaiB :: cyt$

(iii) $3\ KaiC :: cyt \Leftrightarrow KaiC3 :: cyt$

Definition of rule expression (iii) from Running example 3.5 was replaced by a new rule expression using stoichiometry.

3.5 Locations

The localisation operator is intended for allowing an alternative way of expressing the hierarchically constructed agent expressions (Running example 3.8). The main idea is to allow zooming into individual parts of complex and structure expressions. For this purpose, we use $a :: b$ notation such that a, b are arbitrary agents which satisfy one of the conditions given in Definition 3.7.

Definition 3.7 (*Location conditions*)

  (i) $\mathsf{A} :: \mathsf{T} \Leftrightarrow$ there exists $\mathsf{A}' \in \gamma(\mathsf{T})$ such that $\mathsf{A} \lhd \mathsf{A}'$,

 (ii) $\mathsf{A} :: \mathsf{X} \Leftrightarrow$ there exists $\mathsf{A}' \in \mu(\mathsf{X})$ such that $\mathsf{A} \lhd \mathsf{A}'$,

(iii) $\mathsf{T} :: \mathsf{X} \Leftrightarrow$ there exists $\mathsf{T}' \in \mu(\mathsf{X})$ such that $\mathsf{T} \lhd \mathsf{T}'$.

For each pair of agents (α, β) with allowed '::' operator between them, we can construct just one agent β' without the operator by taking the most left agent α' from full (resp. partial) composition of the agent β such that it is *compatible with* the agent α. Then, agent α' is merged with agent α and agent β' is constructed.

Running example 3.8 (*The example model* $\mathcal{M}$)

  (i) $S\{u\} :: KaiC :: KaiC3 :: cyt \Rightarrow S\{p\} :: KaiC :: KaiC3 :: cyt$

 (ii) $S\{u\} :: KaiC :: KaiBC :: cyt \Rightarrow S\{p\} :: KaiC :: KaiBC :: cyt$

(iii) $3\ KaiC :: cyt \Leftrightarrow KaiC3 :: cyt$

Definition of rule expressions (i) and (ii) from Running example 3.6 was replaced using locations. The localisation operator allowed us to additionally use the complex signatures.

3.6 Variables

Rule expressions (i) and (ii) from Running example 3.8 are very similar except for the context of complex expression they take place in. We can substitute this context with a variable with a given domain.

In a rule expression, one agent expression might be referenced using a variable as a set of rule agent expressions it can be replaced with (Running example 3.9). Such an agent expression is referenced as ?X. Moreover, in the case when a ?X is used in a location, it must hold conditions from Definition 3.7.

Each rule expression associated with a variable can be easily written as several rule expressions where the variable is replaced with agent expression from the set of agent expressions attached to the variable. For simplicity, only one variable can be used per rule expression.

Running example 3.9 (*The example model* $\mathcal{M}$)

  (i) $S\{u\} :: KaiC :: ?X :: cyt \Rightarrow S\{p\} :: KaiC :: ?X :: cyt\ ;\ ?X = \{KaiC3, KaiBC\}$

 (ii) $3\ KaiC :: cyt \Leftrightarrow KaiC3 :: cyt$

Definition of rule expressions (i) and (ii) from Running example 3.8 was replaced as a single rule expression with a variable.

This is the final syntactic extension. Compared to the original model (Running example 3.1), the resulting model is more concise and readable.

4 Static analysis

The BCS language offers interesting capabilities to provide several static analysis techniques of given models. These techniques are based on defined *compatibility* operator ◁, which formulates suitable properties for each type of agent.

Definition 4.1 (*Ordering of agents*) Let x_1, x_2 be two arbitrary agents. The compatibility relation induces *partial ordering* of agents x_1 and x_2, written $x_1 \leq x_2$, *iff* $x_1 \triangleleft x_2$.

Notation 4.2 *The universe of complex agents $\mathbb{X}$ with partial order $\leq$ is a partially ordered set $\mathbb{X}_\leq$.*

The compatibility operator defines a partial order on $\mathbb{A}, \mathbb{T}$, and $\mathbb{X}$ sets. For our purposes, only partially ordered set $\mathbb{X}_\leq$ is relevant. The reason is that complex agents actually encapsulate all the other agent types. However, partial order of the entire universe of complex agents is not very useful, since most of the agents cannot be compared by compatibility operator. We are interested in particular subsets where every two complex agents can be either compared directly or there exists an agent compatible with both of them.

Definition 4.3 (*Compatible set*) A finite set $\mathcal{X} \subseteq \mathbb{X}$ is a *compatible set* if:

(i) $\forall X_1, X_2 \in \mathcal{X} \ \exists X' \in \mathcal{X} : X_1 \triangleleft X' \wedge X_2 \triangleleft X'$,

(ii) and for each finite set $\mathcal{X}' \subseteq \mathbb{X}$ such that $\mathcal{X} \cap \mathcal{X}' = \emptyset$ holds: $\forall X \in \mathcal{X} \ \forall X' \in \mathcal{X}' : \neg(X \triangleleft X' \vee X' \triangleleft X)$.

Remark 4.4 The compatible set $\mathcal{X}$ inherits partial order of $\mathbb{X}_\leq$ since it is its subset.

A compatible set $\mathcal{X}$ contains partially ordered complex agents such that they all have the same sequences in terms of agent names. Example of a compatible set is given in Figure 2.
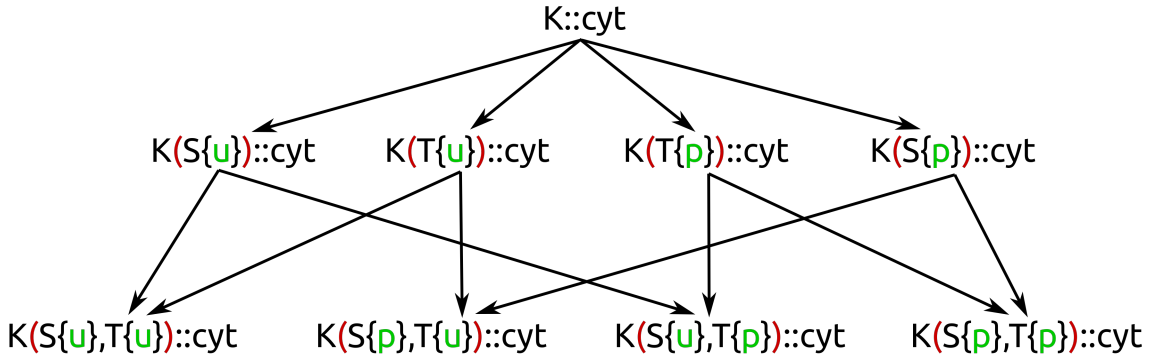

Fig. 2. An example of a compatible set $\mathcal{X}$. The set is formed by a complex in *cyt* compartment, which has only one structure agent K in its sequence. The structure agent K has allowed atomic agents T and S in its partial composition. These two atomic agents might occur in two states – u and p. The set is complete – there are all relevant agents bounded by compatibility operator.

Lemma 4.5 *In every compatible set $\mathcal{X}$, there always exists a global supremum $\sup(\mathcal{X})$.*

Proof. *The lemma follows from Definition 4.3 condition (i) which claims that there is a supremum (in terms of compatibility) for every two complex agents in the compatible set $\mathcal{X}$. Since there exists a supremum for every two items in the set and the set is finite, there must exist a global supremum for the entire set.* □

Lemma 4.6 *For every complex agent X there exists exactly one compatible set $\mathcal{X} \subseteq \mathbb{X}$ such that $X \in \mathcal{X}$.*

Proof. *Let us assume a complex agent X belongs to two compatible sets, namely $X \in \mathcal{X}_1, \mathcal{X}_2$. From Definition 4.3 condition (i) follows that there exists a $X_1 \in \mathcal{X}_1$ such that $X \triangleleft X_1$.*

Next, the condition (ii) claims that no complex agent from $\mathcal{X}_1$ and no complex agent from $\mathcal{X}_2$ can be compatible. Namely, $X_1 \in \mathcal{X}_1$ cannot be compatible with $X \in \mathcal{X}_2$. However, X and X_1 are compatible ($X \triangleleft X_1$). It follows $X \notin \mathcal{X}_2$, which is a contradiction. □

In practise, compatible sets can be used for finding non-trivial relationships between the rules (Section 4.1) and for static analysis on the level of complexes (Section 4.2).

Definition 4.7 (*Compatible subset*) Let $\mathcal{X} \subseteq \mathbb{X}$ be a compatible set and $X \in \mathcal{X}$ a complex agent. A set $\overline{\mathcal{X}} \subseteq \mathcal{X}$ is called *compatible subset* of $\mathcal{X}$ w.r.t. X if the following conditions hold:

(i) $\forall X' \in \bar{\mathcal{X}} : X' \lhd X \wedge \triangle X'$,

(ii) $\nexists X'' \in \mathcal{X} \setminus \bar{\mathcal{X}} : X'' \lhd X \wedge \triangle X''$.

Compatible subset formally defines all fully specified agents from the compatible set which are compatible with a given member of the set (i.e. there are no compatible agents with them in the set). Note that for any complex agent X there exists just one compatible subset. The reason follows from Lemma 4.6 and Definition 4.7.

4.1 Rule redundancy elimination

There might be cases where there are redundant rules in a model (Definition 4.8). These rules do not cause any semantic difference, only increase the size of the model. We provide a static method how to detect such rules and eventually delete them from the model. Please note the redundancy is relevant only in the qualitative context. In the quantitative context, the same rules with different kinetics might have their relevance, yet it is still useful to detect potential redundancies.

Definition 4.8 (*Redundant rule*) Let $\mathcal{M}_1 = (\mathcal{R} \cup \{R\}, \Sigma_A, \Sigma_T, \mathcal{S})$ and $\mathcal{M}_2 = (\mathcal{R}, \Sigma_A, \Sigma_T, \mathcal{S})$ be BCSL models where R is a rule such that $R \notin \mathcal{R}$. The rule R is *redundant* if $LTS(\mathcal{M}_1) = LTS(\mathcal{M}_2)$.

The redundant rule R does not add any semantic information to the model. It generally means the LTSs produced from the models with and without the rule are equal.

Theorem 4.9 *Let* $R = (\chi, \omega, \iota, \varphi, \psi)$ *and* $R' = (\chi', \omega', \iota', \varphi', \psi')$ *be two rules such that* $|\chi| = |\chi'| = n$ *for some* $n \in \mathbb{N}$. *The rule* R' *is* redundant *if* $\forall i \in [1, n] : \chi'_i \lhd \chi_i$.

Proof. *The problem whether the elimination of a redundant rule preserves semantics can be reduced to a simple question – if it holds for a single pair of complex agents for a position* k *in the appropriate rules, then it generally holds for entire rule, because the condition of redundancy holds for each pair of complexes independently.*

Assume the complex agents X_k *and* X'_k *both belong to the same compatible set* $\mathcal{X}$ *since* $X_k \lhd X'_k$, *which follows from the condition of the theorem. We can create subsets* $\bar{\mathcal{X}}, \bar{\mathcal{X}'} \subseteq \mathcal{X}$ *for both complex agents respectively (Definition 4.7). Since the agents are compatible (*$X_k \lhd X'_k$*), the compatible subset* $\bar{\mathcal{X}}$ *w.r.t. agent* X_k *is subset of the compatible subset* $\bar{\mathcal{X}'}$ *w.r.t. agent* X'_k *(*$\bar{\mathcal{X}} \subseteq \bar{\mathcal{X}'}$*).*

Applied generally on the entire rule, the produced set of reactions (using matching relation – Definition 2.31) from the redundant rule is actually a subset of reactions produced from the non-redundant rule. □

In the proof, we used compatible sets of complex agents and the fact that we can generate reactions from the rules, while we are actually enumerating all agents from the compatible set which are *compatible with* original agent in the rule. This is demonstrated in Example 4.10.

Example 4.10 Redundant rule. Let us consider two rules:

(i) $K(S\{u\}).K :: cell \Rightarrow K(S\{p\}).K :: cell$

(ii) $K(S\{u\}, T\{i\}).K :: cell \Rightarrow K(S\{p\}, T\{i\}).K :: cell$

Considering structure signature $\Sigma_T(K) = \{S, T\}$ and atomic signatures $\Sigma_A(S) = \{u, p\}$ and $\Sigma_A(T) = \{a, i\}$, the rule (i) produces following set of eight reactions:

$$\left\{ \begin{array}{l} K(S\{u\}, T\{a\}).K(S\{u\}, T\{a\}) :: cell \Rightarrow K(S\{p\}, T\{a\}).K(S\{u\}, T\{a\}) :: cell, \\ K(S\{u\}, T\{a\}).K(S\{u\}, T\{i\}) :: cell \Rightarrow K(S\{p\}, T\{a\}).K(S\{u\}, T\{i\}) :: cell, \\ K(S\{u\}, T\{a\}).K(S\{p\}, T\{a\}) :: cell \Rightarrow K(S\{p\}, T\{a\}).K(S\{p\}, T\{a\}) :: cell, \\ K(S\{u\}, T\{a\}).K(S\{p\}, T\{i\}) :: cell \Rightarrow K(S\{p\}, T\{a\}).K(S\{p\}, T\{i\}) :: cell, \\ K(S\{u\}, T\{i\}).K(S\{u\}, T\{a\}) :: cell \Rightarrow K(S\{p\}, T\{i\}).K(S\{u\}, T\{a\}) :: cell, \\ K(S\{u\}, T\{i\}).K(S\{u\}, T\{i\}) :: cell \Rightarrow K(S\{p\}, T\{i\}).K(S\{u\}, T\{i\}) :: cell, \\ K(S\{u\}, T\{i\}).K(S\{p\}, T\{a\}) :: cell \Rightarrow K(S\{p\}, T\{i\}).K(S\{p\}, T\{a\}) :: cell, \\ K(S\{u\}, T\{i\}).K(S\{p\}, T\{i\}) :: cell \Rightarrow K(S\{p\}, T\{i\}).K(S\{p\}, T\{i\}) :: cell \end{array} \right\}$$

while the rule (ii) produces set of four reactions:

$$\left\{\begin{array}{l} K(S\{u\}, T\{i\}).K(S\{u\}, T\{a\}) :: cell \Rightarrow K(S\{p\}, T\{i\}).K(S\{u\}, T\{a\}) :: cell, \\ K(S\{u\}, T\{i\}).K(S\{u\}, T\{i\}) :: cell \Rightarrow K(S\{p\}, T\{i\}).K(S\{u\}, T\{i\}) :: cell, \\ K(S\{u\}, T\{i\}).K(S\{p\}, T\{a\}) :: cell \Rightarrow K(S\{p\}, T\{i\}).K(S\{p\}, T\{a\}) :: cell, \\ K(S\{u\}, T\{i\}).K(S\{p\}, T\{i\}) :: cell \Rightarrow K(S\{p\}, T\{i\}).K(S\{p\}, T\{i\}) :: cell \end{array}\right\}$$

which is a subset of the previous one. It follows the rule (ii) is redundant.

4.2 Context-based reduction

There might be cases when simplifying some details of the given BCSL model preserves some properties while making the analysis of the model simpler. This is particularly the case of dynamic analysis, where a minor change in the model specification can dramatically affect the behaviour. To address the model simplification, we first define a function that simplifies rules and then define the notion of a reduced model and show what kind of information does it preserve.

Definition 4.11 (*Rule reduction*) Let $R = (\chi, \omega, \iota, \varphi, \psi)$ be a rule. We define a reduced rule $R' = (\chi', \omega', \iota', \varphi', \psi')$ as a function $\theta(R)$ such that $\forall i \in [1, k] : \chi_i' = \sup(\mathcal{X})$ where $\mathcal{X}$ is a compatible set such that $\chi_i \in \mathcal{X}$, length $k = |\chi'| = |\chi|$ (i.e. the number of complex agents in both rules is the same), and $\iota = \iota'$.

Definition 4.12 (*Reduced model*) Let $\mathcal{M} = (\mathcal{R}, \Sigma_A, \Sigma_T, \mathcal{S})$ be an initial BCSL model. We define *reduced model* $\widetilde{\mathcal{M}} = (\widetilde{\mathcal{R}}, \Sigma_A, \Sigma_T, I)$ such that the following conditions hold:

(i) for every rule $R \in \mathcal{R}$, $\theta(R) \in \widetilde{\mathcal{R}}$ and every rule in the reduced model is the image by θ of a rule of the initial model;

(ii) for every complex agent $X \in \mathcal{S}$, $\sup(\mathcal{X}) \in I$ where $\mathcal{X}$ is a compatible set such that $X \in \mathcal{X}$ and every complex agent in the reduced model is the image by $\sup(\mathcal{X})$ of a complex agent of the initial model.

Reduced model $\widetilde{\mathcal{M}}$ is created from the given BCSL model by reducing the context of complexes in the rules to the maximum level. This is achieved by taking supremum from compatible set $\mathcal{X}$. This procedure can produce some not well-formed rules – such rules are omitted (Figure 3). Consequently, only rules creating/destroying agents and complex formation/dissociation should remain. Since we are reducing context, the number of rules in the resulting model is equal to or smaller than the number of rules in the initial model.

E{a}::cyt + S{u}::cyt => E{a}.S{u}::cyt   E{a}.S{u}::cyt => E{i}.S{p}::cyt

E::cyt + S::cyt => E.S::cyt   ~~E.S::cyt => E.S::cyt~~
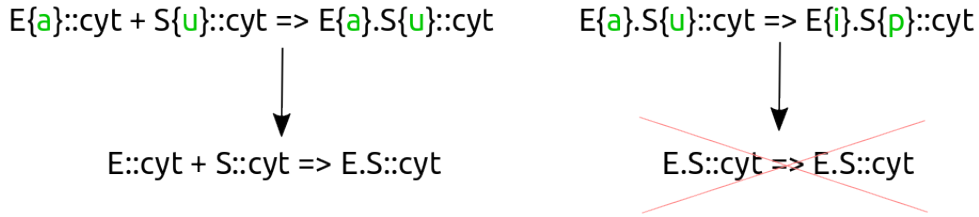
Fig. 3. Examples of rule reductions. (*left*) A rule of complex formation is reduced to a version where none of the states is specified. (*right*) A rule of state change inside of a complex is reduced to a rule which is not well-formed. It violates the condition (i) of Definition 2.26 – an agent has to change during the rule application. Therefore it is removed from the reduced model.

Definition 4.13 (*Compatibility of states*) Let $\mathcal{M}$ be a BCSL model and s_1, s_2 two states from its LTS. The state s_1 is *compatible with* state s_2, written $s_1 \lhd s_2$, *iff* there exists a bijective function $f : s_1 \rightarrow s_2$ such that $\forall X \in s_1 : \sup(\mathcal{X}) = f(X)$ where $\mathcal{X} \subseteq \mathbb{X}$ is a compatible set w.r.t. X.

Definition 4.14 (*Over-approximation of LTS*) Let $LTS(\mathcal{M}), LTS(\mathcal{M}')$ be labelled transition systems of some BCSL models $\mathcal{M}, \mathcal{M}'$. The $LTS(\mathcal{M}')$ is an *over-approximation* of $LTS(\mathcal{M})$ if for every path $\ldots s_1' s_2' s_3' \ldots s_n' \ldots$ in $LTS(\mathcal{M}')$ there exists a path $\ldots s_1 s_2 s_3 \ldots s_m \ldots$ in $LTS(\mathcal{M})$ such that $\forall s_i', s_{i+1}' \exists s_k, s_l : (l > k \wedge s_k \lhd s_i' \wedge s_l \lhd s_{i+1}')$.

A reduced model $\widetilde{\mathcal{M}}$ is actually an over-approximation of a BCSL model $\mathcal{M}$ in the context of their LTSs (Definition 4.14). It can be used for some types of analyses which avoid combinatorial explosion of the initial model $\mathcal{M}$.

Theorem 4.15 *Let* X *be a complex agent,* $\mathcal{X}$ *be a compatible set w.r.t.* X, $\mathcal{M}$ *be a given BCSL model, and* $\widetilde{\mathcal{M}}$ *be an appropriate reduced model of model* $\mathcal{M}$. *If supremum* $\sup(\mathcal{X})$ *is non-reachable in* $LTS(\widetilde{\mathcal{M}})$, *then agent* X *is also non-reachable in the* $LTS(\mathcal{M})$.

13

Proof. *Let us assume a complex agent* $\sup(\mathcal{X})$ *is non-reachable in* $LTS(\widetilde{\mathcal{M}})$, *but* $\mathsf{X} \in \mathcal{X}$ *is reachable in* $LTS(\mathcal{M})$. *Generally, there is a path formed from rules in the* $LTS(\mathcal{M})$ *such that we transform complex agents from initial agents to desired complex agent* X. *When we move to context of* $LTS(\widetilde{\mathcal{M}})$, *there is no such path for* $\sup(\mathcal{X})$.

According to Definition 4.12, for every such rule there exists a reduced rule, such that all interacting complexes are reduced to their suprema. Therefore, if we could apply an initial rule on a complex agent, we can do the same with reduced rule and its supremum. It follows there must exist such path also in $LTS(\widetilde{\mathcal{M}})$ *and the complex agent* $\sup(\mathcal{X})$ *is reachable, which is a contradiction.* □

When we are checking whether an agent is reachable in $LTS(\mathcal{M})$ for given model $\mathcal{M}$, we might first check whether the respective abstract agent (the supremum) is reachable in $LTS(\widetilde{\mathcal{M}})$ of the reduced model $\widetilde{\mathcal{M}}$. If this holds then we are still not certain about reachability of the agent in its initial form. This has to be checked in $LTS(\mathcal{M})$. However, Theorem 4.15 states that agent which is not reachable in $LTS(\widetilde{\mathcal{M}})$ is also not reachable in $LTS(\mathcal{M})$. The usage of the theorem is demonstrated in Section 5.

4.3 Static non-reachability analysis

Since we have defined the compatibility operator for agents, we can apply static non-reachability analysis before enumerating the entire transition system of the model $\mathcal{M}$. We can use the fact that there has to exist a compatible agent on the right-hand side of a rule with the desired agent in order to construct it eventually. This analysis is independent of the initial state of the model. However, it is worth noting that we do not consider the trivial case when the desired agent is already in the initial state.

Theorem 4.16 *Let* $\mathcal{M}$ *be a BCSL model and* $\mathcal{R}$ *its set of rules. Let* X *be a complex agent. The complex agent* X *is non-reachable w.r.t. set of rules* $\mathcal{R}$ *if the following holds:* $\forall\, \mathsf{R} \in \mathcal{R}\ \forall i \in RHS(\mathsf{R}) : \neg(\chi_i \lhd \mathsf{X})$, *where* $\mathsf{R} = (\chi, \omega, \iota, \varphi, \psi)$.

Proof. *Let us assume we have a path of states constructed by applying corresponding rules from* $\mathcal{R}$ *where* X *is reachable. At some point on the path, we inevitably have to create a complex agent* $\mathsf{X}_2 \lhd \mathsf{X}$ *from a complex agent* X_1 *applying a rule* R.
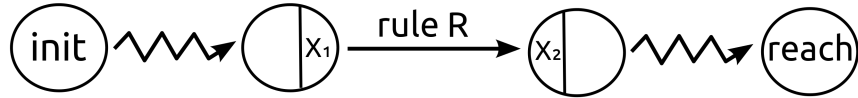
It requires there has to be a complex agent X_2' *in the rule which is compatible with the complex agent* X_2. *If there is no such agent, the agent* X *is non-reachable.* □

Compared to dynamic non-reachability analysis, Theorem 4.16 completely avoids any combinatorial explosion and gives an answer only by checking structural properties of rules. The usage of the theorem is demonstrated in Section 5.

5 Case study

We want to demonstrate practical purposes of static analysis defined in this paper. Yamada et al. model [28] is a model of *fibroblast growth factor* (FGF) signalling pathway. The model represents a signalling pathway, which is typically a cascade of signal transduction. It means that incorrect behaviour on a particular point in the cascade will influence the rest of the pathway. The entire model written in BCSL syntax consists of 20 types of agents interacting in 57 rules. Most of proteins can undergo phosphorylation (state change from u to p on some amino acid residues). We consider initial conditions such that there are all required agents in one or two repetitions (in cases when there are required multiple agents to create complexes, e.g. *FGF*). In such case, the number of reachable states can grow up to 2^{72}, which is too high to be effectively enumerated. In Figure 4, there is a fragment of the model required for our purposes, the whole model is available in Appendix A.

For example, we want to check whether given agent *FRS(Thr{u}, Tyr{u}).FGF(Thr{u}).R.FGF(Thr{u}).R::cyt* is reachable for the given model. The agent is formed from *FGF* proteins which are unphosphorylated (u) on threonine residues (*Thr*). With the traditional approach, we have to enumerate entire transition system of the model and then use model checking method to check it. In our case, we can check if it is non-reachable using *static reachability analysis* (Theorem 4.16). The conclusion is that there is no compatible agent on any right-hand side of the rules. It follows that the given complex agent is non-reachable.

Demonstration of *context-based reduction* (Theorem 4.15) is provided on the same model as in the previous case. We can compute with the entire model since we will reduce its context to the minimum. Applying the

14

reduction, there are created 16 bidirectional rules (Figure 5). The size of transition system has significantly decreased – it has approximately six hundreds of states and two thousands of edges.
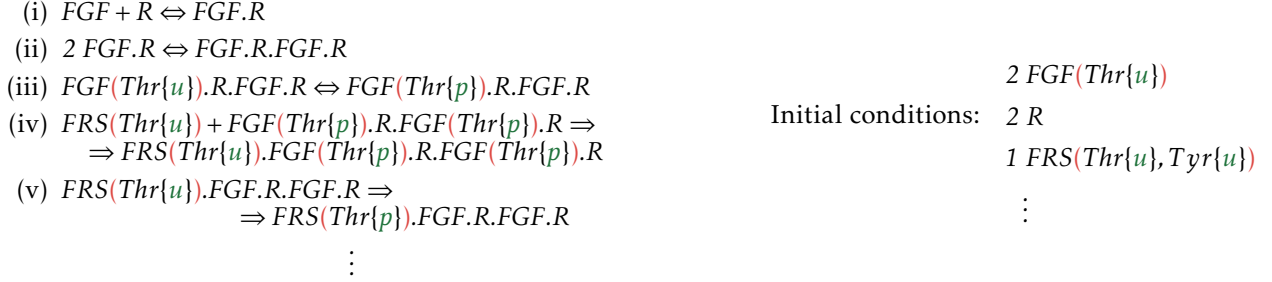
(i) $FGF + R \Leftrightarrow FGF.R$

(ii) $2\ FGF.R \Leftrightarrow FGF.R.FGF.R$

(iii) $FGF(Thr\{u\}).R.FGF.R \Leftrightarrow FGF(Thr\{p\}).R.FGF.R$

(iv) $FRS(Thr\{u\}) + FGF(Thr\{p\}).R.FGF(Thr\{p\}).R \Rightarrow$
$\quad \Rightarrow FRS(Thr\{u\}).FGF(Thr\{p\}).R.FGF(Thr\{p\}).R$

(v) $FRS(Thr\{u\}).FGF.R.FGF.R \Rightarrow$
$\quad \Rightarrow FRS(Thr\{p\}).FGF.R.FGF.R$
$\qquad \vdots$

Initial conditions:

$2\ FGF(Thr\{u\})$

$2\ R$

$1\ FRS(Thr\{u\}, Tyr\{u\})$

$\vdots$

Fig. 4. A fragment of Yamada et al. model [28] of *FGF* signalling pathway written in BCSL. All agents are residing in a cytosol *cyt* compartment, which are omitted for simplicity. The rule (iv) requires both threonine residues (*Thr*) on *FGF* proteins to be phosphorylated (*p*). Basically, it is not possible to create a complex from *FRS* and unphosphorylated (*u*) *FGF* proteins. Full model is available in Appendix A.

For instance, we want check reachability of a complex agent $Raf(Thr\{p\}).ERK(Tyr\{p\}, Thr\{p\})::cyt$ in the initial model. We can first check whether its corresponding least specified agent $Raf.ERK::cyt$ is non-reachable in the reduced model. Since the transition system of the model is relatively small, it can be quite easily checked using dynamical model checking. The answer in this case is non-reachable, which means the original agent in non-reachable too.

FGF	$+$	R		$\Leftrightarrow$	$FGF.R$		GAP	$+$	Ras	$\Leftrightarrow$	$GAP.Ras$
$FGF.R$	$+$	$FGF.R$		$\Leftrightarrow$	$FGF.R.FGF.R$		Ras	$+$	Raf	$\Leftrightarrow$	$Ras.Raf$
$FGF.R.FGF.R$	$+$	FRS		$\Leftrightarrow$	$FGF.R.FGF.R.FRS$		PP	$+$	Raf	$\Leftrightarrow$	$PP.Raf$
FRS	$+$	SHP		$\Leftrightarrow$	$FRS.SHP$		Raf	$+$	MEK	$\Leftrightarrow$	$Raf.MEK$
GS	$+$	GPP		$\Leftrightarrow$	$GS.GPP$		XPP	$+$	MEK	$\Leftrightarrow$	$XPP.MEK$
GS	$+$	ERK		$\Leftrightarrow$	$GS.ERK$		MEK	$+$	ERK	$\Leftrightarrow$	$MEK.ERK$
FRS	$+$	GS		$\Leftrightarrow$	$FRS.GS$		MKP	$+$	ERK	$\Leftrightarrow$	$MKP.ERK$
$FRS.GS$	$+$	Ras		$\Leftrightarrow$	$FRS.GS.Ras$		ERK	$+$	FRS	$\Leftrightarrow$	$ERK.FRS$

Fig. 5. Yamada et al. model [28] after context-based reduction was applied. All agents are residing in a cytosol *cyt* compartment, which are omitted for simplicity. Original model is available in Appendix A.

6 Conclusions

We have presented the second generation of Biochemical Space Language, a novel high-level language for the hierarchical description of biological structures and mechanistic description of chemical reactions by means of compact rules. With respect to the previous prototype [8] the language fully utilises the specific view on the biochemical structures and reactions and the level of abstraction is not lost by translating the language into a low-level formalism not capable of maintaining a hierarchy of object types at the adequate level of abstraction.

We have defined and consequently demonstrated on several case studies static analysis techniques that are unique for the level of abstraction the language uses. We have shown it is possible to detect redundant rules and answer some reachability queries statically. The potential of the language provides the basis for further static analysis that is enabled by the specific abstraction and rule-based approach. Compared to low-level languages, we can take advantage of the hierarchy and relationships built among agents, as demonstrated in provided analysis techniques.

We are aware of necessity to deeply compare these defined relations with the concepts of other formalisms. Our notion of compatible sets has a relation to orthogonal fragments in Kappa [11]. Despite the fact that on our level of abstraction we do not have binding sites, the compatible sets can be seen as a simplified version of orthogonal fragments operating only on the level of states. Similarly, the reduction of models (and consequently reachability analysis) can be related to decontextualisation in Kappa [10]. The formulation of exact relationships is left for the future work.

We are planning to extend the language by quantitative aspects such that we enable simulations of the models. However, this is quite a challenging task since writing a rate of the rule requires to express how particular agents from the rule participate in the rate while keeping the syntax readable and concise. We are also developing the tool BCSgen that is able to maintain and analyse BCSL specifications with its online version eBCSgen.

References

[1] Andrews, S. S., *Smoldyn: Particle-based Simulation with Rule-based Modeling, Improved Molecular Interaction and a Library Interface*, Bioinformatics **33** (2017), pp. 710 – 717.

[2] Calzone, L., F. Fages and S. Soliman, *BIOCHAM: An Environment for Modelling Biological Systems and Formalizing Experimental Knowledge*, Bioinformatics **22** (2006), pp. 1805–1807.

[3] Camporesi, F., J. Feret and K. Q. Lỳ, *KaDE: A Tool to Compile Kappa Rules into (Reduced) ODE Models*, in: *International Conference on Computational Methods in Systems Biology*, Springer, 2017, pp. 291–299.

[4] Cardelli, L., *From Processes to ODEs by Chemistry*, in: *The 5th Ifip International Conference On Theoretical Computer Science (TCS 2008)* (2008), pp. 261–281.

[5] Ciocchetta, F. and J. Hillston, *Bio-PEPA: A Framework for the Modelling and Analysis of Biological Systems*, Theoretical Computer Science **410** (2009), pp. 3065–3084.

[6] Danos, V., J. Feret, W. Fontana, R. Harmer and J. Krivine, *Rule-Based Modelling and Model Perturbation*, in: *Transactions on Computational Systems Biology XI* (2009), pp. 116–137.

[7] Danos, V. and J. Krivine, *Formal Molecular Biology Done in CCS-R*, Electronic Notes in Theoretical Computer Science **180** (2007), pp. 31–49.

[8] Děd, T., D. Šafránek, M. Troják, M. Klement, J. Šalagovič and L. Brim, *Formal Biochemical Space with Semantics in Kappa and BNGL*, Electronic Notes in Theoretical Computer Science **326** (2016), pp. 27–49, the 6th International Workshop on Static Analysis and Systems Biology, SASB 2015.

[9] Faeder, J. R., M. L. Blinov, W. S. Hlavacek et al., *Rule-based Modeling of Biochemical Systems With BioNetGen*, Methods Mol Biol **500** (2009), pp. 113–167.

[10] Feret, J., *Reachability Analysis of Biological Signalling Pathways by Abstract Interpretation*, in: *Proceedings of the International Conference of Computational Methods in Sciences and Engineering*, ICCMSE '2007, Corfu, Greece, number 963.(2) in American Institute of Physics Conference Proceedings (2007), pp. 619–622.

[11] Feret, J. and K. Q. L, *Reachability Analysis via Orthogonal Sets of Patterns*, Electronic Notes in Theoretical Computer Science **335** (2018), pp. 27–48, 7th International Workshop on Static Analysis and Systems Biology (SASB 2016).

[12] Fisher, J. and T. A. Henzinger, *Executable Cell Biology*, Nature biotechnology **25** (2007), p. 1239.

[13] Honorato-Zimmer, R., A. J. Millar, G. D. Plotkin and A. Zardilis, *Chromar, a Rule-based Language of Parameterised Objects*, Theoretical Computer Science (2017).

[14] Kitano, H., *Computational Systems Biology*, Nature **420** (2002), pp. 206 – 210.

[15] Klement, M., T. Děd, D. Šafránek, J. Červený, S. Müller and R. Steuer, *Biochemical Space: A Framework for Systemic Annotation of Biological Models*, Electronic Notes in Theoretical Computer Science **306** (2014), pp. 31–44.

[16] Klement, M., D. Šafránek, T. Děd, A. Pejznoch, L. Nedbal, R. Steuer, J. Červený and S. Müller, *A Comprehensive Web-based Platform for Domain-specific Biological Models* (2013), pp. 61–67.

[17] Lopez, C. F., J. L. Muhlich, J. A. Bachman and P. K. Sorger, *Programming Biological Models in Python Using PySB*, Molecular Systems Biology **9** (2013).

[18] Misirli, G., M. Cavaliere, W. Waites, M. Pocock, C. Madsen, O. Gilfellon, R. Honorato-Zimmer, P. Zuliani, V. Danos and A. Wipat, *Annotation of Rule-based Models with Formal Semantics to Enable Creation, Analysis, Reuse and Visualization*, Bioinformatics **32** (2015), pp. 908–917.

[19] Paulevé, L., S. Youssef, M. R. Lakin and A. Phillips, *A Generic Abstract Machine for Stochastic Process Calculi*, in: *Proceedings of the 8th International Conference on Computational Methods in Systems Biology*, ACM, 2010, pp. 43–54.

[20] Pedersen, M., A. Phillips and G. D. Plotkin, *A High-level Language for Rule-based Modelling*, Plos One **10** (2015), pp. 1–26.

[21] Pedersen, M. and G. Plotkin, *A Language for Biochemical Systems: Design and Formal Specification*, in: *Transactions on Computational Systems Biology XII: Special Issue on Modeling Methodologies*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010 pp. 77–145.

[22] Phillips, A. and L. Cardelli, *Efficient, Correct Simulation of Biological Processes in the Stochastic Pi-calculus*, in: *International Conference on Computational Methods in Systems Biology* (2007), pp. 184–199.

[23] Regev, A. and E. Shapiro, *Cells as Computation*, in: *International Conference on Computational Methods in Systems Biology*, Springer, 2003, pp. 1–3.

[24] Romers, J. C. and M. Krantz, *rxncon 2.0: A Language for Executable Molecular Systems Biology*, bioRxiv (2017).

[25] Sneddon, M. W., J. R. Faeder and T. Emonet, *Efficient Modeling, Simulation and Coarse-graining of Biological Complexity with NFsim*, Nature Methods **8** (2011), pp. 177–183.

[26] Sorokina, O., A. Sorokin, J. D. Armstrong and V. Danos, *A Simulator for Spatially Extended Kappa Models*, Bioinformatics **29** (2013), p. 3105.

[27] Troják, M., D. Šafránek, J. Hrabec, J. Šalagovič, F. Romanovská and J. Červený, *E-cyanobacterium.org: A Web-based Platform for Systems Biology of Cyanobacteria*, in: *Computational Methods in Systems Biology*, LNBI **9859** (2016), pp. 316–322.

[28] Yamada, S., T. Taketomi and A. Yoshimura, *Model Analysis of Difference Between EGF Pathway and FGF Pathway*, Biochemical and Biophysical Research Communications **314** (2004), pp. 1113–1120.

[29] Zhang, F. and M. Meier-Schellersheim, *SBML Level 3 Package: Multistate, Multicomponent and Multicompartment Species*, *Version 1, Release 1*, Journal of Integrative Bioinformatics **15** (2018).

A Model Yamada et al. 2004

$$FGF :: cyt + R :: cyt \iff FGF.R :: cyt$$

$$2\ FGF.R :: cyt \iff FGF.R.FGF.R :: cyt$$

$$FGF(Thr\{u\}).R.FGF.R :: cyt \iff FGF(Thr\{p\}).R.FGF.R :: cyt$$

$$FRS(Thr\{u\}).FGF.R.FGF.R :: cyt \implies FRS(Thr\{p\}).FGF.R.FGF.R :: cyt$$

$$FRS(Thr\{p\}).FGF.R.FGF.R :: cyt \implies FRS(Thr\{p\}) :: cyt + FGF.R.FGF.R :: cyt$$

$$SHP :: cyt + FRS(Thr\{p\}) :: cyt \implies SHP.FRS(Thr\{p\}) :: cyt$$

$$FRS(Thr\{p\}).SHP :: cyt \implies FRS(Thr\{u\}).SHP :: cyt$$

$$FRS(Thr\{u\}).SHP :: cyt \implies FRS(Thr\{u\}) :: cyt + SHP :: cyt$$

$$GPP :: cyt + GS(Thr\{p\}) :: cyt \implies GPP.GS(Thr\{p\}) :: cyt$$

$$GS(Thr\{p\}).GPP :: cyt \implies GS(Thr\{u\}).GPP :: cyt$$

$$GS(Thr\{u\}).GPP :: cyt \implies GS(Thr\{u\}) :: cyt + GPP :: cyt$$

$$ERK(Tyr\{p\}, Thr\{p\}) :: cyt + GS(Thr\{u\}) :: cyt \implies ERK(Tyr\{p\}, Thr\{p\}).GS(Thr\{u\}) :: cyt$$

$$GS(Thr\{u\}).ERK :: cyt \implies GS(Thr\{p\}).ERK :: cyt$$

$$GS(Thr\{p\}).ERK :: cyt \implies GS(Thr\{p\}) :: cyt + ERK :: cyt$$

$$FRS(Thr\{p\}, Tyr\{u\}) :: cyt + GS(Thr\{u\}) :: cyt \iff FRS(Thr\{p\}, Tyr\{u\}).GS(Thr\{u\}) :: cyt$$

$$Ras(Thr\{u\}).FRS.GS :: cyt \implies Ras(Thr\{p\}).FRS.GS :: cyt$$

$$Ras(Thr\{p\}).FRS.GS :: cyt \implies Ras(Thr\{p\}) :: cyt + FRS.GS :: cyt$$

$$GAP :: cyt + Ras(Thr\{p\}) :: cyt \implies GAP.Ras(Thr\{p\}) :: cyt$$

$$Ras(Thr\{p\}).GAP :: cyt \implies Ras(Thr\{u\}).GAP :: cyt$$

$$Ras(Thr\{u\}).GAP :: cyt \implies Ras(Thr\{u\}) :: cyt + GAP :: cyt$$

$$Ras(Thr\{p\}) :: cyt + Raf(Thr\{u\}) :: cyt \implies Ras(Thr\{p\}).Raf(Thr\{u\}) :: cyt$$

$$Raf(Thr\{u\}).Ras :: cyt \implies Raf(Thr\{p\}).Ras :: cyt$$

$$Raf(Thr\{p\}).Ras :: cyt \implies Raf(Thr\{p\}) :: cyt + Ras :: cyt$$

$$PP :: cyt + Raf(Thr\{p\}) :: cyt \implies PP.Raf(Thr\{p\}) :: cyt$$

$$Raf(Thr\{p\}).PP :: cyt \implies Raf(Thr\{u\}).PP :: cyt$$

$$Raf(Thr\{u\}).PP :: cyt \implies Raf(Thr\{u\}) :: cyt + PP :: cyt$$

$$Raf(Thr\{p\}) :: cyt + MEK(Ser212\{u\}) :: cyt \implies Raf(Thr\{p\}).MEK(Ser212\{u\}) :: cyt$$

$$MEK(Ser212\{u\}).Raf :: cyt \implies MEK(Ser212\{p\}).Raf :: cyt$$

$$MEK(Ser212\{p\}).Raf :: cyt \implies MEK(Ser212\{p\}) :: cyt + Raf :: cyt$$

$$Raf(Thr\{p\}) :: cyt + MEK(Ser298\{u\}) :: cyt \implies Raf(Thr\{p\}).MEK(Ser298\{u\}) :: cyt$$

$$MEK(Ser298\{u\}).Raf :: cyt \implies MEK(Ser298\{p\}).Raf :: cyt$$

$$MEK(Ser298\{p\}).Raf :: cyt \implies MEK(Ser298\{p\}) :: cyt + Raf :: cyt$$

$$XPP :: cyt + MEK(Ser212\{p\}) :: cyt \implies XPP.MEK(Ser212\{p\}) :: cyt$$

$$MEK(Ser212\{p\}).XPP :: cyt \implies MEK(Ser212\{u\}).XPP :: cyt$$

$$MEK(Ser212\{u\}).XPP :: cyt \implies MEK(Ser212\{u\}) :: cyt + XPP :: cyt$$

$$XPP :: cyt + MEK(Ser298\{p\}) :: cyt \implies XPP.MEK(Ser298\{p\}) :: cyt$$

$$MEK(Ser298\{p\}).XPP :: cyt \implies MEK(Ser298\{u\}).XPP :: cyt$$

$$MEK(Ser298\{u\}).XPP :: cyt \implies MEK(Ser298\{u\}) :: cyt + XPP :: cyt$$

$$ERK(Thr\{u\}).MEK :: cyt \implies ERK(Thr\{p\}).MEK :: cyt$$

$$ERK(Thr\{p\}).MEK :: cyt \implies ERK(Thr\{p\}) :: cyt + MEK :: cyt$$

$$ERK(Tyr\{u\}).MEK :: cyt \implies ERK(Tyr\{p\}).MEK :: cyt$$

$$ERK(Tyr\{p\}).MEK :: cyt \implies ERK(Tyr\{p\}) :: cyt + MEK :: cyt$$

$$MKP :: cyt + ERK(Thr\{p\}) :: cyt \implies MKP.ERK(Thr\{p\}) :: cyt$$

$$ERK(Thr\{p\}).MKP :: cyt \implies ERK(Thr\{u\}).MKP :: cyt$$

$$ERK(Thr\{u\}).MKP :: cyt \implies ERK(Thr\{u\}) :: cyt + MKP :: cyt$$

$$MKP :: cyt + ERK(Tyr\{p\}) :: cyt \implies MKP.ERK(Tyr\{p\}) :: cyt$$

$$ERK(Tyr\{p\}).MKP :: cyt \implies ERK(Tyr\{u\}).MKP :: cyt$$

$$ERK(Tyr\{u\}).MKP :: cyt \implies ERK(Tyr\{u\}) :: cyt + MKP :: cyt$$

$$FRS(Tyr\{u\}).ERK :: cyt \implies FRS(Thr\{u\}, Tyr\{p\}).ERK :: cyt$$

$$FRS(Thr\{u\}, Tyr\{p\}).ERK :: cyt \implies FRS(Thr\{u\}, Tyr\{p\}) :: cyt + ERK :: cyt$$

$$FRS(Tyr\{p\}) :: cyt \implies FRS(Tyr\{u\}) :: cyt$$

$$ERK(Thr\{u\}) :: cyt + MEK(Ser212\{p\}, Ser298\{p\}) :: cyt \implies$$
$$\implies ERK(Thr\{u\}).MEK(Ser212\{p\}, Ser298\{p\}) :: cyt$$

$$Ras(Thr\{u\}) :: cyt + FRS(Thr\{p\}, Tyr\{u\}).GS(Thr\{u\}) :: cyt \implies$$
$$\implies Ras(Thr\{u\}).FRS(Thr\{p\}, Tyr\{u\}).GS(Thr\{u\}) :: cyt$$

$$FRS(Thr\{u\}) :: cyt + FGF(Thr\{p\}).R.FGF(Thr\{p\}).R :: cyt \implies$$
$$\implies FRS(Thr\{u\}).FGF(Thr\{p\}).R.FGF(Thr\{p\}).R :: cyt$$

$$ERK(Tyr\{u\}) :: cyt + MEK(Ser212\{p\}, Ser298\{p\}) :: cyt \implies$$
$$\implies ERK(Tyr\{u\}).MEK(Ser212\{p\}, Ser298\{p\}) :: cyt$$

$$FRS(Tyr\{u\}) :: cyt + ERK(Tyr\{p\}, Thr\{p\}) :: cyt \implies$$
$$\implies FRS(Tyr\{u\}).ERK(Tyr\{p\}, Thr\{p\}) :: cyt$$