Formal Analysis of Rule-Based Models in Systems Biology

Ph.D. Thesis Proposal

Matej Troják

Advisor: prof. RNDr. Luboš Brim, CSc.

Brno, Fall 2019
# Contents

1 Introduction 1

2 Rule-based modelling 3
   2.1 Formal definition .................................. 3
   2.2 Analysis methods .................................. 6

3 State of the Art 8
   3.1 Rule-based languages ............................... 8
      3.1.1 Kappa language ............................... 9
      3.1.2 BioNetGen Language ......................... 9
      3.1.3 PySB ........................................ 10
      3.1.4 Chromar ..................................... 11
      3.1.5 Language for Biochemical Systems .......... 12
      3.1.6 Biochemical Space Language ................ 12
      3.1.7 SBML-multi package ........................ 14
   3.2 Analysis methods .................................. 14
      3.2.1 Simulation ................................... 16
      3.2.2 Model checking ............................... 16
      3.2.3 Parameter synthesis .......................... 18
      3.2.4 Monitoring ................................... 19
      3.2.5 Robustness analysis ......................... 19
      3.2.6 Static analysis .............................. 19

4 Aim of the Thesis 21
   4.1 Objective I: Analysis methods .................... 21
      4.1.1 Model checking ............................... 22
      4.1.2 Parameter synthesis .......................... 22
      4.1.3 Robustness analysis .......................... 23
   4.2 Objective II: Software tool ........................ 23
   4.3 Progression schedule ................................ 24

5 Achieved results 25

6 Publications 29
Bibliography

A Summary of Author’s PhD Study
  A.1 Posters and talks ........................................ 39
  A.2 Teaching, supervising, and reviewing .................... 40

B Attached publications ........................................ 41
Modelling complex systems in systems biology has to be conducted at several levels of abstraction that reflect well the known information [56]. At every level, the system has to be described rigorously in a formal language that allows to avoid misunderstanding 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.

Traditional approaches used to describe biochemical systems are a chemistry approach employing “mechanical” descriptions by chemical reactions or a mathematical approach using ordinary differential equations or other mathematical formalisms. An advantage of chemical reactions over mathematical equations is the fact that chemical reactions are composable to some extent, human-readable and well-understood across disciplines while still having executable semantics. Moreover, there are methods automatising the generation of mathematical models from chemical equations. The problem of both approaches is combinatorial explosion on the level of behaviour of modelled system (so called state space explosion problem) and the level of model specification (the compactness of the specification).

The so-called computational approach [21, 39] offers abstraction of the information about individual model components and interactions, providing a way how to avoid the combinatorial explosion on the level of model specification. A promising computational approach is provided by rule-based modelling [30, 43] and process-algebraic frameworks [21, 24, 82]. Rule-based models form a natural extension of the mechanical reaction-based models used in biochemistry while avoiding the combinatorial explosion using compactness. Instead of operating with objects, rule-based frameworks operate with types that allow to avoid the combinatorial explosion that occurs when underlying objects are specified explicitly. The semantics of the model is given in terms of rules defined on given types. The rules express a high degree of modularity which helps to avoid the explicit enumeration of all
possible molecular species or all the states of a system. They also enable a more natural starting point for model development than making ad hoc assumptions to decide the model scope. An important advantage of the rule-based models 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 [20], PDEs [1], chemical master equation or continuous-time Markov chains [75, 85], reaction-diffusion systems [86], etc.).

The long-term aim of our research is the development of Comprehensive Modelling Platform [59, 60] – a general modelling framework that combines model building, model analysis, and annotation tasks in a single public web site related to a particular system. It respects the need for maintaining existing ODE models (which is still a typical scenario in systems biology) but allows to align them with a mechanistic rule-based description that is understandable by biologists, compact in size, and executable in terms of allowing basic analysis tasks ensuring consistency of the description. Such a comprehensive solution supports modellers in building mathematical models that have clear biochemical meaning and can be easily integrated. Moreover, mechanistic descriptions can be later used as standalone 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 a rule-based language called Biochemical Space Language (BCSL) [34]. It has the following key aspects: human-readability – easy to read, write, and maintain; executability – formal executable semantics is defined allowing efficient static analysis and consistency checking; universality – principally different cellular processes can be sufficiently combined in a single specification; and compactness – the combinatorial explosion of the description is avoided. However, similarly to any other rule-based language, the problem of the state space explosion is not covered efficiently.

The main objectives of the thesis are to develop methods for analysis of rule-based models operating on the level of the model specification, targeting the problem of state space explosion; and to develop a software tool supporting Biochemical Space Language. The main effort is to develop exact methods which provide global guarantees overcoming the limitations of approximative, sampling-based, and bounded methods while avoiding complete state space exploration. The emphasis is placed on model checking, parameter synthesis, and robustness analysis.
Chapter 2

Rule-based modelling

The conventional modelling of a biochemical system usually involves drawing a reaction-scheme diagram depicting the chemical species and reactions in the system and then manually translate it into a set of equations \[90\] (e.g. ordinary differential equations). The limit of such an approach is that the models derived in this way are usually based on assumptions and simplifications. Lifting the simplifications causes a combinatorial explosion in the number of possible reactions and species \[11\], which makes manual modelling impractical.

A new starting point for the model specification is necessary. One of the potential approaches is the idea to specify the biochemical interactions as rules that serve as generators of chemical reactions (or reaction events) and species. A rule can be viewed as a definition of a reaction class (a generalised reaction).

A rule-based system is based on rewriting of some structured objects called agents according to predefined rules. The structure defined on the agents is motivated by the fact that enzymatic and binding activities of proteins involved in signalling tend to be localised to modular domains \[76\]. Proteins may be composed of smaller parts, usually referenced as sites, used for modification or molecular binding. Such structure enables varying the level of context given in the rules, which is generally used to restrict the domain the rule can be applied to. The less context is given, the more general the rule becomes. It is the key feature which supports the conciseness of the rule-based approach.

2.1 Formal definition

In this chapter, we define a simplified version of the rule-based system. The defined rules are based on simpler objects called reactions, which are used for rewriting of some data structures composed by species. Then, the rule is defined as a set of reactions, encoding multiple possibilities how it can be
used, depending on a particular context. First, we start with the definition of a multiset as a form of a data structure used for rewriting.

**Definition 2.1 Multiset**

Multiset $\mathcal{M}$ is a pair $(\mathcal{A}, \mathbf{m})$ where $\mathcal{A}$ is a set and $\mathbf{m} : \mathcal{A} \rightarrow \mathbb{N}$ 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 $\mathbf{M}(a)$.

The elements of the multiset are often referenced as species. We denote with $\mathcal{M}$ the set of all multisets. With $\Gamma$ we denote the set of all arbitrary rational functions over a domain of species. The notation $\nu(\mathcal{M})$ with $\nu \in \Gamma$ and $\mathcal{M} \in \mathcal{M}$ denotes the evaluation of function $\nu$ to a rational number by substituting each variable $a$ by its corresponding value $\mathbf{M}(a)$.

**Definition 2.2 Reaction**

A reaction $\omega$ is a triple $(\text{lhs}, \text{rhs}, \nu) \in \mathcal{M} \times \mathcal{M} \times \Gamma$ where $\text{lhs}$ is left-hand side, $\text{rhs}$ is right-hand side, and $\nu$ is a rational rate function.

Informally, a reaction states how different species interact to change the number of copies of the corresponding species in the state of the system. The state of the system is an unstructured collection (multiset) of copies of each species. By $\Omega$ we denote the set of all reactions.

**Definition 2.3 Reaction application**

Let $\omega = (\text{lhs}, \text{rhs}, \nu)$ be a reaction and $\mathcal{M}$ be a multiset. The reaction application to the multiset $\mathcal{M}$, written $\omega(\mathcal{M})$, is a pair $(\mathcal{M} - \text{lhs} + \text{rhs}, \nu(\mathcal{M}))$ if $\mathcal{M} - \text{lhs} > 0$; $(\mathcal{M}, 0)$ otherwise.

Applying a reaction to a state (multiset) means removing from the state the number of copies of elements that appear in the left-hand side ($\text{lhs}$) of the reaction and adding the elements that appear on the right-hand side ($\text{rhs}$) of the reaction. Additionally, the rate function $\nu$ is evaluated in the process and provides reaction rate, i.e. speed at which the reaction happens. In the case when the number of occurrences of some species required by $\text{lhs}$ is not high enough, the application is not possible and the rate is zero. For example of reaction application, see Figure 2.1.

**Definition 2.4 Rule**

A rule $\rho \in 2^{\Omega}$ is defined as a set of reactions. By $\omega \in \rho$ we denote a reaction from the definition domain of rule $\rho$.

A rule is a generalisation of multiple reactions. It can be viewed as a high-level compact definition of a set of chemical reactions. Instead of a single possible way of application as in the case of reaction, there are
Figure 2.1: A graphical example of reaction application. In a multiset represented by a big circle we have three kinds of agents – □, ○, and △. In the reaction $\omega = (\{\square : 1, \bigcirc : 1\}, \{\triangle : 1\}, 2 \times \square \times \bigcirc)$, the agents □ and ○ are consumed and a new agent △ is produced. The rate of the reaction is given as a function of molecular counts of both reactants. After substituting the counts to $\square = 2$ and $\bigcirc = 4$, the evaluation of $2 \times 2 \times 4$ gives the rate 16.

multiple outcomes of the application to a given multiset. By $\Sigma$ we denote the set of all rules.

**Definition 2.5 Rule application**

Let $\rho$ be a rule and $M$ be a multiset. The rule application to the multiset $M$, written $\rho(M)$, is a set $\{\omega(M) \mid \omega \in \rho\}$ of all reaction applications from the definition domain of the rule.

Given a multiset, applying a rule to it produces a set of multiset–rate pairs. This is caused by the fact that all possible reactions represented by the rule can produce a different outcome (we do not consider reactions with zero rate due to a non-negative number of repetitions requirement in reaction application). Example of rule application is given in Figure 2.2.

**Definition 2.6 Rule-based model**

A rule-based model $\mu$ is a pair $(R, M_0) \in 2^\Sigma \times M$ where $R$ is a set of rules and $M_0$ is an initial state (multiset) of the model.

A rule-based model is composed of two parts – the initial state, describing the number of occurrences of individual species in the default situation, and the set of rules which define how the species can interact. The semantics of the model is given in terms of rule application, which is transitively applied to the initial state.

A natural extension of the model is parametrised rule-based model where rate functions can be enriched by unknown parameters from a known domain (usually integers or rational numbers). It is a typical case in systems biology since the actual values of reaction kinetics are often hard or even impossible to be measured. Then, to evaluate a rate function, a parametrisation $p$ from
the admissible parameter space has to be given, representing a particular biological scenario.

2.2 Analysis methods

An important advantage of the rule-based approach is that mathematical models can be automatically generated from it. In particular, instead of relying on a single mathematical formalism, different mathematical models can thus be obtained for a given model (e.g., ODEs [20], chemical master equation or continuous-time Markov chains [75, 85]). This fact results in multiple possible ways of how to analyse and explore the models.

Simulation is the most usual way how to explore the behaviour of a model. Depending on the interpretation of rule rates, we can use numerical simulation of the rule-based model for both its deterministic and stochastic dynamics. Analysing the models using simulation is very useful, but does not provide a global overview of its behaviour.

Model checking [27] is an automated way for checking whether the model meets a given specification. For that, we need a specification language which allows us to express a property dependent on time. A suitable tool to use is
a temporal logic. Examples of logics which are often used in systems biology are LTL [79], CTL [26], STL [70], or PCTL [44] and their usage depends on particular mathematical representation we use.

**Definition 2.1 Model checking**

Given a rule-based model $\mu = (R, M_0)$ and a property formula $\phi$, the problem of model checking is to decide whether the model $\mu$ satisfies the property $\phi$.

In the case of unknown parameter values present in the model, we can be interested in finding those parametrisations from the parameter space which satisfy the given property. For that, a method called parameter synthesis is used. The basic principle is to search parameter space and identify regions where satisfaction is (not) guaranteed. Regions, where no guarantees are given, can be produced due to some computational error of used searching method.

**Definition 2.2 Parameter synthesis**

Given a parametrised rule-based model $\mu = (R, M_0)$ and a formula $\phi$, the problem of parameter synthesis is to compute a partitioning of the given parameter space into three disjoint subsets: TRUE – the parameter values satisfying the property, FALSE – the parameter values violating the property, and UNKNOWN – the result is not known.

Additionally, instead of the binary answer to the satisfaction of a property, a measure of how much the property is preserved (resp. violated) in the given parametrisation can be useful. This measure is called local robustness $D_{\mu}^{\phi}(p)$ and states how much the property $\phi$ is preserved in parametrisation $p$ of model $\mu$.

**Definition 2.3 Robustness analysis**

The problem of global robustness of a model $\mu$ is defined as

$$R_{\phi,P}^{\mu} = \int_P \psi(p)D_{\phi}^{\mu}(p)dp$$

where $\phi$ is the property of the system under scrutiny, $P$ is the parameter space, $\psi(p)$ is the probability of the parametrisation $p$, and $D_{\phi}^{\mu}(p)$ is the local robustness.

The global robustness [58] defines a measure of how the given property is globally preserved in the model with respect to the parameter space.
Chapter 3
State of the Art

In this chapter, we provide an insight into the state-of-the-art of the rule-based modelling, including its foundations, existing available formalisms, and established analysis techniques which can help in the process of model-building tasks.

The formal foundation of rule-based modelling is based in process algebra that enables principled mathematical reasoning about system behaviour and its dependency on interaction capabilities of system components. One of the first used approaches is $\pi$-calculus [71], a general and minimalist model-specification language originally designed to capture essential features of concurrent and distributed systems in computer science. Use of $\pi$-calculus to model biological interactions was suggested in [83].

However, because $\pi$-calculus is a minimalist language, and some of its notational features are irrelevant and even inappropriate for biological applications (for example, communication has directionality, whereas physical association does not), researchers have sought to develop a more congruent higher-level language for modelling biochemical systems that retains the mathematical formality of $\pi$. In the next section, we provide a list of the most established rule-based formalisms.

3.1 Rule-based languages

There are several rule-based languages dedicated to modelling of the biological systems. Each of them uses different features and abstractions. In this chapter, we will highlight the key features of some of the representatives and describe analysis established for them. It is important to note that generally, they can always be formulated in terms defined in Chapter 2.

A specificity of individual languages resides in the level of abstraction they use and the way how they handle the encoding of a rule such that they avoid the explicit reaction network representation. The implicit representation they use includes both syntactic and semantic level, providing a direct
apparatus of rules application.

### 3.1.1 Kappa language

The Kappa language [30] was primarily developed for the modelling of protein interactions. The key structures used in the language are agents with binding sites, which allow formation of bonds between the agents. Each binding site must be unique with at most one bond. Each site can occur in one of several pre-defined states.

The Kappa rules are changing properties of the agents. Particularly, the rule might add, delete, or change a bond or a state of one or multiple agents at once. The rules are patterns with two sides where each of them is a sequence of agents delimited by a comma. The introduction of sites allows to group multiple reactions to a single rule by omitting the context which is not relevant. This is a typical way how rule-based languages encode reactions to rules and it ensures conciseness of description. Example of a rule is in Figure 3.1.

(a) \( A(\text{bsa} \sim \{u, p\}); B(\text{bsb} \sim \{a, i\}) \)

(b) \( A(\text{bsa}), B(\text{bsb} \sim a) \rightarrow A(\text{bsa}!1), B(\text{bsb} \sim a!1) \)

Figure 3.1: An example of a Kappa rule. (a) Definition of associated agents. There are allowed two agents, \( A \) and \( B \), both with a binding site which might occur in two possible states. (b) A Kappa rule which defines a creation of bond on the sites of agents \( A \) and \( B \) such that site of agent \( B \) must be in an active state. Note that the context with no importance for the interaction is omitted (in this case the state of binding site \( \text{bsa} \) of agent \( A \)), encoding multiple possible reactions.

The general problem present in the Kappa language is too detailed description. For biological systems, it might often be difficult to manage bonds between sites, especially in cases when we do not need such details.

### 3.1.2 BioNetGen Language

The BioNetGen Language (BNGL) [43] is similar to the Kappa language, but there are several differences. The basic idea of the language is to use molecules to describe the building blocks of a biological system such as proteins, genes, and metabolites. Each molecule has a number of sites with associated internal states that are used to represent the status of post-translational modifications or bindings with other molecules. The rules describe the interactions among molecules including associations, dissociations, modifications to the internal state of a site as well as the produc-
tion or consumption of molecular species. Rules also provide rate laws for transformations resulting from molecular interactions. Patterns are used to identify a set of molecules that share the same internal states. The rule can be specified using patterns such that it defines not a single reaction but a potentially large class of reactions, all involving a common transformation parametrised by the same rate law.

BNGL rule typically represent molecular interactions and the consequences of these interactions. Example of a rule is given in Figure 3.2. Software tools have been developed to construct, visualise, simulate, and analyse BNGL models [43, 85, 91, 92].

\[
\begin{align*}
(a) & \quad A(bsa \sim u \sim p) \\
& \quad B(bsb \sim a \sim i) \\
(b) & \quad A(bsa) + B(bsb \sim a) \to A(bsa!1).B(bsb \sim a!1)
\end{align*}
\]

Figure 3.2: An example of a BNGL rule. (a) Definition of associated agents. There are allowed two agents, A and B, both with a binding site which might occur in two possible states. (b) A BNGL rule which defines association of two agents. Note that the agents A and B must first create reaction complex denoted by '+' (i.e., they must be close to each other) and then they create complex of agents denoted by '.' (i.e., they are physically connected).

### 3.1.3 PySB

The PySB language [68] is an open-source programming framework written in Python that allows concepts and methodologies from contemporary software engineering to be applied to the construction of transparent, extensible, and reusable biological models. It is implemented as a package in the Python programming language.

The definition of the models directly in the code allows the usage of the full syntax of Python, which significantly increases the expressive power of PySB. Its widespread use in the computational biology community, support for object-oriented and functional programming, and rich ecosystem of mathematical and scientific libraries make it an excellent choice of programming language for this purpose. On the other hand, the increased expressive power make the models harder to analyse.

The core of the language is defined by translating to BNGL. PySB is closely integrated with Python numerical tools for simulation and graphical tools that enable plotting of model trajectories and topologies. The main advantage of the language is that it can be used to decompose models into reusable macros that can be independently tested, and then used to generate composite models. Example of a rule is given in Figure 3.3.
# Declare the monomers
Monomer('A', ['bsa'])
Monomer('B', ['bsb'])

# Declare the binding rule
Rule(A(bsa=None) + B(bsb=None) <> A(bsa=1) % B(bsb=1), kf, kr)

Figure 3.3: An example of a PySB rule. The rule describes creation of a bond between agents A and B.

3.1.4 Chromar

The Chromar language [47] allows to define attributes for agents and range them over pre-defined domains. The qualitative semantics is given by rule match on multisets composed of these agents producing a reaction. It is followed by standard application of the reaction (in the manner of multiset operations, for details see Definition 2.3). The language is very useful when creating a model where we need to create new distinct objects and control the population of these objects.

Embedding this language into the functional programming language Haskell increases its expressive power while making the ability of some analysis more expensive. Moreover, a user needs to understand Haskell (at least its basics) in order to use Chromar.

It is important to highlight a feature which offers the stochastic semantics of this language. Compared to the other rule-based languages, it is capable of specifying the rates for individual reactions inherited from the rule (Figure 3.4). It is allowed by variable value bindings and type-determination between the left and right-hand side of the rule.

\[
A(a = x), A(a = y) \xrightarrow{f(x, y)} A(a = x + y), A(a = y - 1) \ [g(x, y)]
\]

Figure 3.4: An example of a Chromar rule. Arithmetic operations can be used in order to change the properties of individual agents. Moreover, a rate function f and conditional function g increase applicability and practical use of the rule.

However, when it comes to readability and presentation to the user, the language has a crucial disadvantage – all the biologically relevant terms such as states have to be encoded in natural numbers.
3.1.5 Language for Biochemical Systems

Language for Biochemical Systems (LBS) [78] combines rule-based approaches to modelling with modularity. The main features of the language are species expressions for manipulating large complexes in a concise manner, parameterised modules with a notion of subtyping for writing reusable modules, and nondeterminism for handling combinatorial explosion.

A simple example of an LBS program is shown in Figure 3.5. Species identifiers such as mRNA are here assigned to new species. The text new{} is formally a species expression which evaluates to a species value with no modification sites and with a globally unique name. Hence the name of the species identifier, mRNA, does not in itself hold any identity of a species, and one may bind the same identifier to an entirely different species in another part of the program. This allows different modules, possibly developed by different people, to use the same species identifier for mRNA molecules which are semantically and biologically different, and subsequently combine the modules into a single program without unintended cross-talk.

```
spec gene = new{}, rnap = new{}, mrna = new{}, prot = new{};
comp c new comp; comp n = new comp inside c;
  c |
    n [ gene + rnap -> gene + rnap + mrna ] |
    n [ mrna | -> mrna ]
    mrna -> prot
```

Figure 3.5: An LBS program for gene expression.

In [78], authors provide a demonstration of multiple concrete semantics defined for the rule-based description: namely basic Petri nets, coloured Petri nets, ordinary differential equations, and Continuous-time Markov chains. For that, they usually use a reaction-based (or ground) representation of the system (except coloured Petri nets which allow most of the rule-based abstract features to be encoded in colours).

Moreover, LBS-Kappa [77] is a modular extension of Kappa language based on LBS, providing a language for writing high-level and modular rule-based models. Since LBS is a general framework, it was only needed to define the necessary instantiation to Kappa and refer to the general semantics of LBS.

3.1.6 Biochemical Space Language

Biochemical Space Language (BCSL) [88] is a language that combines the advantages of different approaches and thus makes an effort to overcome several problems of existing solutions. BCSL is an integral part of a general framework developed for comprehensive modelling in systems biology [59,
BCSL relies on the formal basis of the rule-based methodology while preserving user-friendly syntax of plain chemical equations. The target group of users is a biologically-oriented community. Therefore the emphasis is placed on usability and positive user experience.

The language uses several types of agents, describing different levels of abstraction and detail. Particularly, there is an atomic agent describing the most basic type of biological objects which have assigned an internal state. The state typically represents a modification of the objects, e.g. oxidised, reduced, methylated etc. Structure agent represents a biochemical object that is composed of several known atomic agents while we know that composition is abstract and not necessarily complete. Since not all biochemical structures are known in detail, this feature is useful in expressing partial knowledge (for example, see Figure 3.6). Complex agent groups several structure or atomic agents, forming a non-trivial composite biochemical object. A typical feature of the language is that it does not use any form of binding when it comes to complex formation. Instead, the complex formation is considered as a form of coexistence; the particular interpretation depends on the context of the model.

\[
\text{ps2(oec\{2+\}, yz\{+\})::tlm} \Rightarrow \text{ps2(oec\{3+\}, yz\{n\})::tlm}
\]

Figure 3.6: An example of BCSL rule. Oxidation of S2-state of the oxygen-evolving complex \text{oec\{2+\}} by \text{yz\{+\}} in photosystem II (\text{ps2}). The photosystem is represented as a structure agent. There are many other subparts of photosystem II. (e.g. chlorophyl, phaeophytin), but their states are not important in this case. This makes the syntax of the language very concise and readable.

The agents interact via rules. In the definition of a rule, identifiers of substrates and products are used to make the notation of the rules compact. Every agent appearing in a rule equation has to be followed by the localisation operator associating it with a particular compartment. This is, for example, important for rules that act on both sides of a membrane. That way, a rule is always precisely localised in or between the compartments.

Moreover, it is possible to assign a name to complex, which increases the readability and provides additional annotation options. This is supported by the binary localisation operator ‘::’. The main idea is to allow zooming into individual parts of agents. The localisation operator naturally leads to an extension by variables, which can group several rules matching in the same modification of an agent (for example, see Figure 3.7).
\[ S\{u}\cdot KaiC::?X::cyt \leftrightarrow S\{p\}:KaiC::?X::cyt \; \]
\[ ?X = \{ KaiC6, KaiA2C6, KaiA4C6, \\
KaiB6C6, KaiA4B6C6, KaiA6B6C6 \} \]

Figure 3.7: An example of BCSL rule. The state of a serine residue \( S \) of \( KaiC \) protein can be changed from unphosphorylated to phosphorylated whenever the \( KaiC \) protein is included in a complex. The particular allowed complexes which enable this modification are given by the variable \(?X\). Each of them is provided in the form of a complex alias, which requires a proper complex definition in the model.

### 3.1.7 SBML-multi package

Systems Biology Markup Language (SBML) [48] is a standard established for systems biology based on XML. It is popular standardised format for the electronic storage, exchange, and reuse of mathematical models of biochemical systems. However, it is based on the assumption that a model can be specified adequately in terms of a reaction scheme, which is not suitable for a rule-based model.

The newest version SBML Level 3 introduced modular language extension capability, which allows different language features to be added to a common language core. An SBML-multi package [94] is able to describe all the necessary rule-based features, and therefore, it is possible to export each model in a rule-based language in this format. It is the most suitable format for the exchange and storage of the models but less for analysis and direct representation to the users. It serves as an intermediate format for model sharing. The presence of a feature tag informs a software tool reading the model that the model uses that particular feature and permits the tool to quit if it does not have the necessary interpretive capabilities. For example, see Figure 3.8.

### 3.2 Analysis methods

In the context of systems biology, the computer-aided verification methods are becoming important for analysing the models, validating new experimental results, automatically checking behaviours of interest, and identifying the inputs or parameters of the system enforcing the desired behaviour. The formal verification of a model consists of proving that its execution satisfies a given specification of the required behaviour.

Rule-based models can be used to generate other computational models using different semantics. Therefore, any developed analysis techniques for
Figure 3.8: An example of a simple SBML-multi model. The model requires specification of language level (version). In the definition, there are included SpeciesTypes, Species which belong to these types, and Reactions where those species are interacting. Additionally, all processes are encapsulated in the compartments.
the other computational models can be indirectly used to analyse the rule-based models. A disadvantage of such a procedure is that the computational models are often explicit, and thus the implicit representation of the rule-based approach is lost during the translation.

3.2.1 Simulation

The most usual way how to exploit the behaviour of a rule-based model is a simulation. While qualitative simulation is also possible (generate a possible successor of the current state while abstracting the notion of time), more usual is a quantitative simulation. Using the generated reaction network from the model rules, one can perform stochastic simulation, for example, using Gillespie algorithm [40], or deterministic simulation [80] by consequent translation to Ordinary differential equations (ODEs) [45]. However, the cost of simulating the kinetics of a reaction network depends on the size of the network. Thus, if a network is large, the simulation cost can be expensive in terms of computation time or computer memory. To address this problem, network-free simulators were developed that take advantage of rules. The method generalises an agent-based kinetic Monte Carlo method that has been shown to circumvent the combinatorial bottleneck in simulations [93].

An example of a network-free simulator is NFsim [85]. The advantage of NFsim compared to reaction-based approaches is that during simulation, rules operate directly on molecular objects to produce exact stochastic results with the performance that scales independently of the reaction network size. Reaction rates can be defined as arbitrary functions of molecular states to provide powerful coarse-graining capabilities. NFsim enables researchers to simulate many biological systems that were previously inaccessible to general-purpose software.

Analysing the models using simulation is very useful, but usually not sufficient. The models are often difficult to calibrate due to many unknown parameters and limited experimental training data. It is necessary to apply the powerful tools developed in the context of program verification to biological models, particularly when they can be cast in the form of rules that are executed stochastically on the fly. The simulation will only yield a particular trajectory at each run. Even when many runs are gathered to perform statistical analysis, observing a time series of concentrations (or molecule numbers) does not necessarily lead to an understanding of the model.

3.2.2 Model checking

A typical approach of dynamic analysis is to compute a data structure, describing the whole system behaviour. The easiest way to do this is by constructing the reachability graph (also called transition system). The nodes of
a reachability graph represent all possible states of the modelled system and
the arcs represent actions which caused the particular state change. How-
ever, reachability graphs tend to be extremely large because they comprise
all possible system states. The state space explosion motivates the static
analyses discussed below in this section.

If we succeed in constructing the complete reachability graph, we can
decide the behavioural properties of the modelled system. Here are some
typical examples of qualitative dynamic analysis:

- **Reachability.** The reachability problem is to check whether there exists
  a path from a given state to a particular state. The problem can be
  reformulated to finding a sub-state, meaning we are only interested
  in the molecular counts of particular species. It has been shown that
  the reachability problem is decidable [61] although it takes at least
  exponential space (and time) to verify in the general case [66].

- **Boundedness.** A system is bounded if the number of molecular counts
  is finite for any state reachable from the given initial state. This is
  very useful for the detection of undesirable inconsistency in the flows
  in the model. It is also generally required by model checking.

- **Persistence.** A system is persistent if, for any two reaction events,
  the firing of one reaction will not disable the other. This analysis
  is suitable for detecting concurrent reactions in the models, which
  are often responsible for some critical decision-making in biological
  systems.

Generally, checking a property of a model is called **model checking.**
Model checking is an automatic formal verification technique able to per-
form a clever exhaustive search of the state space of a model. It operates
over a discrete-time model with a finite number of states (usually a Kripke
structure [62]), determining the truth value of a temporal logic formula
which specifies desired property of the modelled system. Examples of logics
which are often used in systems biology are LTL [79], CTL [26], STL [70],
or PCTL [44], depending on the particular mathematical formalism.

A rule-based model can be translated to such a formalism and analysed
by an efficient model checker such as [23]. An important issue with this
technique is that the number of states of a model usually grows exponentially
in the number of its variables, giving rise to the state explosion problem.
Additionally, in the general case, a state space of the rule-based is infinite,
making the analysis even harder to handle. For this reason, there are often
some limitations in the application of the rules which bound possible state
space (e.g. global bound on allowed molecular counts in states), abstracting
some information and providing a way how to create a finite state space.
For these reasons, most of the model checkers usually represent the states symbolically [18], such as binary decision diagrams [17]. Relevant methods for the analysis of continuous semantics are based on the abstraction of the behaviour to a finite number of discrete states. For example, one can be interested in parallel semi-symbolic coloured model checking [8], bifurcation analysis [7], and attractor analysis [9].

Model checking has been extended to many other computational models such as continuous- and discrete-time Markov chains (CTMC and DTMC) by adding probabilities [63]. A tool used for analysing the Markov chains is PRISM [64], which calculates the probability that the model will satisfy the property of interest. It can either provide the exact solution [2] or an approximated solution [52] (using a set of samples, generated using a Monte Carlo simulation of the model). Moreover, it is also possible to compute a numerical value expressing the measure of satisfaction instead of a binary decision, often referred to as quantitative model checking [49].

On-the-fly algorithms employ a top-down approach to model checking, which does not require global knowledge of the complete state space [10, 28, 41, 46, 65]. Instead, the algorithms construct step by step local knowledge of the state space until it is possible to decide whether the given state satisfies the formula.

### 3.2.3 Parameter synthesis

Parameter synthesis is the inverse problem to model checking – its goal is to find the maximal subset of parameter values such that they meet the stated dynamical constraints. In practice, the unknown parameters are usually limited by a set of admissible values, providing a parameter space. In the context of systems biology, there are several relevant methods which solve parameter synthesis.

In [5], the authors present an algorithm for parameter synthesis based on parallel model checking, which is conceptually universal with respect to the modelling approach employed. In [6], an efficient method to analyse large and possibly incomplete parametrised piecewise-affine differential equation models is proposed. Usage of a symbolic encoding of the model structure enables the methods to take full advantage of symbolic model checkers for testing CTL dynamic properties. An algorithm for computing parameter synthesis in non-linear dynamical systems [36] extends formal verification techniques that were first introduced in the context of continuous and hybrid non-linear dynamical systems [38].

In [53], algorithms for studying the parameter space of stochastic biochemical models were presented. Methods for parameter synthesis of parametric Markov chains have been introduced with symbolic computation of reachability properties through state elimination [32, 42, 51], recently improved by parameter lifting [81] and fraction-free Gaussian elimination [50].
An alternative approach to the analysis of complex stochastic models under parameter uncertainty is based on statistical methods [3, 13, 14, 69]. There are only a few works that bridge the rule-based framework to such techniques. In [67], a statistical parameter sampling method is employed to analyse unknown parameters in BNGL models represented by means of CTMCs where the rate function is limited to mass action kinetics. The recent work [55] combines statistical model checking with machine learning techniques to calibration (estimation) of parameters in order to maximise the probability of satisfying a given specification. In [12], the authors adapt simulation-based and moment-based methods. In general, statistical techniques do not give an exact symbolic representation of satisfying parameter sets.

3.2.4 Monitoring

Another way to tackle the state explosion problem is the analysis of a single execution trace instead of performing an exhaustive verification. Monitoring aims to check whether the current execution of a program satisfies or violates a property of interest. For example, in BIOCHAM [19], authors implemented monitoring of numerical simulations of biological models with LTL used to specify properties of real-valued variables. Additionally, in [35] they implemented a tool where the evaluation of the STL [70] formula robustness is used for a particular trajectory through monitoring.

3.2.5 Robustness analysis

According to [57], robustness is a property that allows a system to maintain its functions against internal and external perturbations. The formal definition of robustness is given in Definition 2.3. The concept of robustness is well established for deterministic systems [37, 84]. The evolution of a stochastic system is given by a set of paths compared to a single trajectory of a deterministic system. The stochastic system at any given time is described by a probability distribution over states of the corresponding CTMC in contrast to the single state representation of a deterministic system. Therefore, the definition of robustness for stochastic systems requires a more sophisticated approach. In [22], authors developed an adaptation of the concept of robustness to stochastic systems, particularly robustness analysis in CTMCs. The method is based on a numerical approximation of the evaluation function [16], which provides accuracy guarantees in contrast to existing methods employing parameter sampling and statistical techniques.

3.2.6 Static analysis

Static analysis is performed on the specification of the model without the need to actually execute it [73]. While model checking generally needs to
explore all the states originated by executing the semantics of the model, static analysis operates on the syntactic level of the specification or by using abstract interpretation [29] over finite approximations of the possible model executions [74]. It is a powerful tool preferably used to detect potential issues in the model.

There are several well-defined static analysis methods for rule-based systems, which can be useful in finding inconsistency in the models [31]. For example: detection of dead rules – a rule is called dead, if there is no trace starting from the initial state in which this rule is applied; detection of dead agents – an agent is called dead, if there is no trace starting from the initial state with at least one state in which this agent occurs; detection of redundant rules – a rule is redundant if after removing it from the model, the particular semantics do not change. These and many more methods can be run statically, providing an efficient way of how to verify special types of properties before executing the model.

To the best of our knowledge, the only tool dedicated to static analysis of rule-based models is KaSa [15]. This static analyser abstracts the set of reachable states of models and then uses this information to collect insightful properties. In particular, KaSa may warn about rules that may never be applied, about potential definitive transformations of proteins, and about the potential formation of unbounded molecular compounds.
Chapter 4

Aim of the Thesis

In general, there exist three approaches to the analysis of rule-based models: (i) direct approach operating on the level of the model (set of rules), (ii) indirect approach analysing the transition system explicitly generated from the model, and (iii) semi-direct approach analysing the transition system while avoiding its explicit generation (e.g., using heuristics or a symbolic representation).

There are multiple ways how to analyse rule-based models indirectly through existing computational methods established in computer science (Section 3.2). Nonetheless, the general drawback of indirect methods is that they use an explicit transition system of the model, leading to the state space explosion problem. On the other hand, direct methods (e.g., static analysis discussed in Section 3.2.6) or some semi-direct methods (e.g., statistical model checking discussed in Section 3.2.3), provide an approach how to avoid (to some extent) the explicit representation of the transition system. However, to the best of our knowledge, they do not solve crucial problems in model analysis, such as model parameter synthesis or robustness analysis.

The main objectives of the thesis are to develop automatic analysis methods for rule-based models using either direct or semi-direct approach, targeting the state explosion problem; and to develop a software tool for support of Biochemical Space Language, as a rule-based language suitable for modelling in systems biology.

4.1 Objective I: Analysis methods

The first objective of the thesis is to propose direct or semi-direct analysis methods for rule-based models, which should be general and applicable to any rule-based formalism. The focus is put on model checking, parameter synthesis, and robustness analysis.
4.1.1 Model checking

To the best of our knowledge, the existing model checking methods for rule-based languages use either the indirect or the semi-direct approach. Since there do not exist direct methods for model checking of rule-based models, the primary goal is to establish a direct method. Alternatively, an improvement of existing semi-direct approaches such that they provide exact results (no heuristics) is also possible. The emphasis is placed on the reachability problem, as one of the most important problems when it comes to the analysis of biological systems.

Computer science already offers several approaches for model checking of models, which can be potentially used in the case of rule-based models. One option is to use model checking methods developed for reaction-based systems which avoid the exploitation of entire state space. It is necessary to adapt these methods for rule-based formalism.

Another option would be to use an existing indirect approach which requires complete state space and encode it symbolically or to explore on-the-fly techniques. For example, there is a method for probabilistic reachability analysis of DTMC, which constructs a regular expression (RE) using state elimination. The evaluation of RE gives the probability of reaching targeted species. The algorithm for RE construction, however, does not necessarily need the explicit DTMC. We could use this fact for an on-the-fly construction of the RE, potentially avoiding the complete state space enumeration.

The ideal solution avoids the building of the state space completely. It is the most challenging task and requires the development of new symbolic and static methods. However, as it was shown in [88], some of the model checking subtasks can be solved statically due to unique abstraction employed by rule-based languages (e.g., some cases of non-reachability).

4.1.2 Parameter synthesis

Parameter synthesis is a crucial branch of model analysis in the case when the parameters of the modelled system are not completely known. However, in rule-based community, the direct approach to this problem is not yet well developed. The only known work uses the semi-direct approach and focuses on parameter estimation using statistical model checking (Section 3.2.3). Therefore another goal of this thesis is to develop parameter synthesis algorithm for rule-based models, which provides exact results avoiding the indirect approach.

One of the options on how to solve this problem is based on model checking approach from the previous subsection. Once we have an efficient model checking method, the parameter synthesis can be formulated using this method. For example, the mentioned method which constructs RE could be generalised to the parametrised case. Instead of a single probability
evaluation, we obtain a probability function of parameters, which can be then used for parameter space exploitation.

Other solutions could, for example, use some heuristic to obtain probability function, avoiding the usage of model checking, but providing only limited guarantees on the correctness of the results.

4.1.3 Robustness analysis

Robustness analysis characterises the mean validity of a formula over all parameter values in the given perturbation set. To the best of our knowledge, this problem was not yet formulated for rule-based systems. Another goal of this thesis is to define and solve this problem in the context of rule-based systems.

In order to provide an exact solution, it is necessary to compute the function of parameters. One of the possible solutions is using RE approach mentioned in previous sections. Having such a function, we can integrate over the perturbation set and compute the robustness.

Another possible way how to achieve this goal is by using heuristics to estimate the local robustness, which would provide an approximate estimation of global robustness.

4.2 Objective II: Software tool

The second objective of the thesis is to develop tool BCSgen to support modelling and analysis of Biochemical Space Language, which includes the implementation of the proposed analysis methods. In order to enable the proposed analysis of the models specified in the language, an extension of the language by quantitative properties is necessary, which are not supported in the present version yet.

BCSgen implements environment suitable for editing and maintenance of the BCSL models by providing the following functionality: an editor with syntax highlighting and error messaging in the process of model building; simulation engine offering both deterministic and stochastic simulations; reaction network generator; transition system generator; export and import of models in appropriate standard file format; visualisation of transition system including highlighting paths from the initial state to other states, which satisfy reachability condition; visualisation of time-series data from simulation; all the proposed techniques for model checking, parameter synthesis, and robustness analysis.

The tool will be integrated within CMP, improving the user’s model building experience. It is supported by the fact that BCSL is targeted to be so-called human-readable and simple enough such that it is understandable beyond the computer science community. The languages from Section 3.1
all offer interesting features when it comes to modelling. However, none of them except BCSL focuses on these properties.

4.3 Progression schedule

Plan for future study and research activities:

- Fall 2019: Doctoral exam and defence of this thesis proposal.
- Fall 2019 – Fall 2020: Development of analysis methods.
- Fall 2020 – Spring 2021: Implementation of developed methods.
- Spring 2021: Release of a stable version of the software tool.
- Fall 2021: Thesis defence.
Chapter 5

Achieved results

Our research has been primarily focusing on the development of new modelling approaches in systems biology. Our long-term ongoing research is focused on the development of Comprehensive Modelling Platform (CMP) – a general framework for public sharing, annotation, and visualisation of domain-specific biological models. For a selected system (e.g. an organism, particular process), the framework is instantiated as a web-based application which allows capturing several aspects of biological models. The platform is targeting biologically-oriented users and is trying to improve model-building tasks, including analysis support and automatic verification of models.

In [87], we presented e-cyanobacterium.org\(^1\) – an instance of CMP oriented on processes of cyanobacteria. The platform is unique in focusing purely on cyanobacteria, which allows integration of the knowledge and its presentation in a concise and understandable form. Several well-annotated and curated models of particular cyanobacteria processes developed by the consortium are provided within e-cyanobacterium, covering environmental processes, respiration and photosynthesis, carbon concentrating mechanism, circadian clock, and metabolism.

We used the models from e-cyanobacterium to automatically analyse attractors [9], which can characterise complex dynamics arising in biological systems. We demonstrated a novel method for detecting terminal strongly connected components in the ODE models, which can be abstracted in the form of parametrised graphs. The main advantage of the method is the ability to detect attractors without precise knowledge of systems behaviour or even its parameters. This type of analysis can be performed to discover unstable or oscillatory behaviour in a model where such behaviour is typically undesired. In Figure 5.1, there are some interesting results from this case study.

The core part of CMP is Biochemical Space (BCS) [59]. BCS mechanistically describes the modelled process with emphasis placed on precise

\(^1\)https://www.e-cyanobacterium.org
Figure 5.1: Dependence of stable state in $CO_2$ and $HCO_3^-$ residing in carboxysome on the value of parameter fast affecting the rate of carbon fixation reaction in the Clark model [25] studied in [9].

annotation by links to existing biological ontologies and by association with experimental data. This formal description then serves as a knowledge base for all mathematical models enabled by relating model components to BCS entities (for example of a BCS entity, see Figure 5.2). The mechanism of linking individual objects assigns biological meaning to mathematical constructs. This step is useful in the interpretation process of analysis results.

<table>
<thead>
<tr>
<th>ID:</th>
<th>KaiC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>circadian clock protein kinase KaiC</td>
</tr>
<tr>
<td>Compartments:</td>
<td>cyt</td>
</tr>
<tr>
<td>Composition:</td>
<td>S, T</td>
</tr>
<tr>
<td>Classification:</td>
<td>protein</td>
</tr>
<tr>
<td>Description:</td>
<td>Monomer component representing a core component of the circadian clock system.</td>
</tr>
<tr>
<td>Links:</td>
<td>UniProt::Q79PF4, CyanoBase::Synpcc7942_1216</td>
</tr>
<tr>
<td>Organism:</td>
<td>Synechococcus elongatus PCC 7942</td>
</tr>
</tbody>
</table>

Figure 5.2: An example of complete information given for an agent representing circadian clock protein kinase KaiC of Synechococcus elongatus PCC 7942 strain. The emphasis is placed on the annotation – relevant data entries are Q79PF4 in UniProt database [89] and Synpcc7942_1216 in CyanoBase [72]. Links to other databases such as KEGG [54] and ChEBI [33] are also supported.

An important feature of BCS is specification language it uses for the description of the processes. The key aspects necessary for such language are: human-readability – easy to read, write, and maintain the description; ex-
ectability – defined semantics are needed for efficient consistency checking and analysis; universality – multiple cellular processes with different level of detail can be sufficiently combined in a single specification; and compactness – the combinatorial explosion of the specification is avoided.

To that end, we have developed the first version of Biochemical Space Language [34]. The language was presented in its basic form with emphasis on the annotation. The semantics of the language was defined indirectly by embedding to Kappa [30]. However, 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 general modelling framework [60]. Additionally, Kappa does not provide a hierarchical description, which is one of the key aspects of BCSL.

In [88], we introduced a significantly improved version of BCSL with respect to the primary prototype. The new key aspects are: 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; 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. Together with the formal definition, we provided several static analysis techniques. Particularly, we showed how redundant rules in a model could be detected and automatically eliminated without loss of information, we provided a context-based reduction technique which produces an over-approximation of the modelled system, and we showed a static non-reachability technique which can check some reachability properties without the execution of model behaviour. On a case study of fibroblast growth factor signalling pathway, we demonstrated these techniques and showed how useful they could be in model building tasks.

The language is supported by BCSgen tool. The general goal of the tool is to provide the environment suitable for editing and maintenance of the BCSL models and provide the functionality to analyse the models. The tool was developed as a part of Dean’s Program of the Faculty of Informatics MU for support of student research and development projects MUNI33/092015, where the only functionality provided by the tool was the import of the model from a text file (in a predefined format) and enumeration of the entire transition system of the model. The implementation was done by translating to Kappa and there was a simple graphical user interface.

The project MUNI33/062017 in the same program focused on the improvement of models maintenance, additional analysis services, and visualisation. The transition system generator was improved, using a reaction-based approach instead of translation to another language. Additionally, an interactive editor with automatic syntax highlighting and error messag-

---

2https://github.com/sybila/BCSgen
ing, import and export of the models, and reachability analysis were implemented. The graphical exploration of the transition system with highlighting the chosen node and announcing the content of the node and highlighting a path from the initial state to a state satisfying reachability condition were also implemented.

Currently, a web-based version eBCSgen$^3$ is under development using Galaxy platform, which a first step towards integrating it within CMP.

$^3$https://github.com/sybila/eBCSgen
Chapter 6

Publications


  (85%) A journal paper presenting BCSL on an informal level. I formulated the language description and prepared case study demonstrating static analysis and usability of the language on different biological processes.


  (14%) A case study paper presenting the technique from [4] applied to models from the e-cyanobacterium.org repository. I helped with the experimental evaluation and contributed to the text describing the models and the evaluation results. The paper was awarded the best paper award in its category.


  (80%) A conference paper presenting a new version of BCSL. I formulated the formal definition of the language and static analysis, including their application examples.


29
A tool paper presenting e-cyanobacterium.org as an instance of CMP. I helped with writing most of the text and brought several ideas to the web platform development.


A conference paper describing the first version of BCSL defined by embedding to another language. I helped to formulate the formal definition and algorithms for the embedding.
Bibliography


Appendix A

Summary of Author’s PhD Study

A.1 Posters and talks

As a member of Systems Biology Laboratory and a PhD student, I had several presentations on conferences:


and several poster presentations:


A.2 Teaching, supervising, and reviewing

Appendix B

Attached publications
Formal Biochemical Space with Semantics in Kappa and BNGL

T. Děd, D. Šafránek, M. Troják, M. Klement, J. Šalagović, L. Brim

Faculty of Informatics, Masaryk University
Brno, Czech Republic

Abstract

Biochemical Space (BCS) has been introduced as a semi-formal notation for reaction networks of biological processes. It provides a concise mapping of mathematical models to their biological description established at a desired level of abstraction. In this paper, we first turn BCS into a completely formal language with rigorously defined semantics by means of a simplified Kappa calculus. On the practical end, we support BCS with translation to BNGL, a well-known practically used rule-based language. Finally, we show the current status of BCS defined for cyanobacteria processes.

Keywords: Kappa, Biochemical space, CMP, cyanobacteria processes

1 Introduction

To provide a rigorous representation of complex biological processes without congesting the users with overcomplicated syntax, we have enriched our online platform for modelling of cyanobacteria processes, e-cyanobacterium, with a semi-formal textual notation called Biochemical Space (BCS) [10]. BCS represents reaction networks of the studied processes and provides a concise mapping of mathematical models to a precise biological description that is established at a consortium-agreed level of abstraction.

The concept of BCS makes a crucial methodological part of Comprehensive Modelling Platform (CMP), a general platform for computational modelling and analysis of biological processes, first introduced in [15] as a concept for unambiguous representation of internally consistent reduced mathematical models of oxygenic photosynthesis [17] and further refined to a general online modelling platform as described in [11]. In general, the main goal of BCS as a part of CMP is to simplify

1 This work has been supported by the Czech Science Foundation grant No. GA15-11089S.

http://dx.doi.org/10.1016/j.entcs.2016.09.017
1571-0661/© 2016 Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
systems level model-building tasks by providing simple and clear way of notation easily understandable by *in silico* modellers on the one end, and experimental biologists on the other end.

![Graphical representation of Comprehensive Modelling Platform (CMP)](image)

In [1] we have shown that rule-based methods can be directly used for rewriting existing kinetic models of oxygenic photosynthesis into a compact non-redundant form obtained by applying a set of automatised syntactic reductions defined in Kappa [4]. That achievement lead us further to employ rule-based definition of biological processes as the framework for qualitative description of the consortium-agreed understanding of chemical reactions behind the processes. Existing quantitative models can be then mapped onto the qualitative rule-based BCS.

BCS borrows concepts from two worlds – the formal rule-based languages and semi-formal reaction network annotation bases such as KEGG [9]. The BCS language is defined with a clear relation to BNGL [6], a practical tool-supported rule-based language compatible with Kappa. Since the most important requirement of the consortium-driven modelling platform is a simple-to-use format well-adjusted to the suitable level of abstraction employed for biological process description, we were not able to directly employ any of the well-established rule-based languages and rather defined a new language with a clear relation to the existing formats.

In particular, for our purpose BNGL and Kappa consider too many details. The most important fact is that BNGL requires to specify bindings inside the complex structures. This demands binding sites specification for each molecule and unique labelling for each interaction. In BCS, these structural details are abstracted out. It is enough just to know that molecules interact and form a complex while abstracting from the details. Another issue is the fact that existing formalisms consider biological entities as agents all defined at the same level of abstraction. In BCS we allow hierarchical construction of agents from simple molecules to composite structures and complexes. Finally, the algebraic representation of Kappa and BNGL goes quite far from common chemical notation and is not human readable. BCS attempts to avoid this.

In [10] we have presented general ideas behind BCS. The language has been defined as a semi-formal notation. In this paper, we turn BCS into a completely formal language with clearly defined syntax and semantics (by embedding to Kappa). We
define the relation of BCS and BNGL which allows us to translate specification between both languages. In Section 7, we show the current status of BCS description implemented for cyanobacteria.

1.1 Related Work

On the bioinformatics side, the closest format to BCS is KEGG [9]. In contrast to BCS, KEGG does not support rule-based description allowing compact representation of combinatorial states. Moreover, it does not support logical organization of entities and reactions into an organism-specific hierarchy that may significantly simplify understanding of the complex processes driving the organisms physiology and its interaction with the environment. Since the notation relies on a simple textual base and focuses on a simple but still reasonably precise and compact description maintainable by biologists, the format of BCS specification is compliant with KEGG.

BCS should be also compared to the well-acclaimed standard provided by SBML [8, 13] that might be also used for representation of a biochemical space. BCS completely avoids issues related with dynamical models. As an annotation platform purely focused on process-level description, BCS goes beyond SBML level 2 in generalization of entities to hierarchical agents, in introducing entity states, and in dealing with related combinatorial explosion. These issues are solved in detail by rule-based approaches [4,7] and there is a draft of a package for SBML level 3 in preparation [18] (multi).

In comparison with process algebraic languages treating chemical reactions mechanistically as communicating concurrent processes [2,5], BCS keeps a purely qualitative level of description closed to chemical reactions and remains as simple as possible to cover the consortium-agreed level of abstraction. The language defined in [16] targets a similar level of abstraction as BCS. However, it is intended more as a programming language for biological systems than an annotation format.

2 Background

We define simplified Kappa (kappa_s) using a process-like notation as is presented in [3], syntax and the notions of structural equivalence and matching are entirely taken from [3]:

expression: \( E ::= \emptyset | a, E \)
agent: \( a ::= N(\sigma) \)
agent name: \( N ::= A \in A \)
interface: \( \sigma ::= \emptyset | s, \sigma \)
site: \( s ::= n^\lambda \)
site name: \( n ::= x \in S \)
interface: \( n ::= x \in S \)
internal state: \( \iota ::= \epsilon | m \in \mathbb{V} \)
binding state: \( \lambda ::= \epsilon | i \in \mathbb{N} \)

where \( A \) is a finite set of agent names, \( S \) is a finite set of site names, \( \mathbb{V} \) is a finite set of values representing modified states of the sites. We use notation \( \sigma(a) \) for a signature associated to an agent \( a \).

An agent is denoted by its name and its interface. Interface consists of a sequence of sites. \( x^\lambda_i \) denotes a site \( x \) with internal state \( \iota \) and binding state \( \lambda \). If the binding state is \( \epsilon \) then the site is free, otherwise it is bound. By convention, when a binding...
or internal site is not specified, $\epsilon$ is considered.

Note that full Kappa is richer. It allows a binding state meaning a free or bound site, denoted by a question mark. We also omit rates from the rules.

**Definition 2.1** An expression is well-formed if a site name occurs only once in an interface and if each binding state ($\neq \epsilon$) present in the expression occurs exactly twice. The set of all well-formed expressions is denoted as $\mathcal{E}$.

We assume a standard structural equivalence on well-formed expressions that treats as equivalent all expressions differing in order of sites in interfaces, order of agents in expression, and naming of binding sites.

A **rule** is a pair of expressions $E_l$, $E_r$ (usually written as $E_l \rightarrow E_r$). The set of all rules is denoted as $\mathcal{R}$. The left hand side $E_l$ of the rule describes the solution taking part in the reaction and the right hand side $E_r$ describes the effects of the rule. The rule can be either a binding rule or a modification rule. A binding (unbinding) rule binds two free sites together (or unbinds two bound sites). A modification rule modifies some internal state [3].

**Matching** is a relation denoted as $|$ $\subseteq \mathcal{E} \times \mathcal{E}$ and defined inductively in the left column below. **Replacement** is a function $\mathcal{E} \times \mathcal{E} \rightarrow \mathcal{E}$ defined in the right column below:

$$
n_i^\lambda \models n_i^\lambda \quad n_i^\lambda[n_i^\lambda] = n_i^\lambda
$$

$$
\sigma \models \emptyset \quad \sigma[\emptyset] = \sigma
$$

$$
s \models s_l \quad \sigma \models \sigma_l \quad s, \sigma[s_r, \sigma_r] = s[s_r], \sigma[\sigma_r]
$$

$$
\sigma \models \sigma_l \quad N(\sigma)[N(\sigma_r)] = N(\sigma[\sigma_r])
$$

$$
E \models \emptyset \quad E[\emptyset] = E
$$

$$
a \models a_l \quad E \models E_l \quad (a, E)[a_r, E_r] = a[a_r], E[E_r]
$$

A replacement can be applied only if the corresponding matching is satisfied.

In order to apply a rule $E_l \rightarrow E_r$ to a solution $[E]$ the expression $E$ representing the solution must first be reordered to an equivalent expression $E'$ that matches $E_l$ (according to the definition of matching stated above). $E'$ is then replaced with $E'[E_r]$ (also defined above).

**Rule application** is a mapping $t : \mathcal{E} \times \mathcal{R} \rightarrow \mathcal{E}$ such that $t([E], (E_l, E_r)) = [E'[E_r]]$ whenever $\exists E' \in [E] : E' \models E_l$. Rules yield a transition system between solutions containing an edge $[E] \rightarrow E_l, E_r \ [E'[E_r]]$ whenever $\exists E' \in [E], E' \models E_l$.

An **agent signature** $(\Sigma, I)$ is a pair of mappings $\Sigma : \mathcal{A} \rightarrow 2^S$ and $I : \mathcal{A} \times S \rightarrow 2^\mathcal{V}$. Informally, $\Sigma$ restricts for each agent name $A \in \mathcal{A}$ the set of site names that can occur in an agent with name $A$ and $I$ restricts the set of internal states a particular site can attain. Additionally, expressions are treated as complete if their agents employ all sites and states of the signature. For formal definitions see [1] or the original paper [3].

A **rule-based model** $\mathcal{M}$ is a tuple $(\Sigma, I, \mathcal{R})$ such that $\mathcal{R}$ satisfies the signature $(\Sigma, I)$. An **initialised model** $\mathcal{M}_0$ is a pair $(\mathcal{M}, E_i)$ where $\mathcal{M} = (\Sigma, I, \mathcal{R})$ is a rule-
based model and $E_i$ is an expression representing the initial solution such that $E_i$ is complete for the signature $(\Sigma, I)$.

**Definition 2.2** A state space of an initialised model $M_0 = (M, E_i)$ is a pair $(\text{Solutions}(M_0) \subseteq \mathcal{E}, \text{Reactions}(M_0) \subseteq \mathcal{E} \times \mathcal{E})$ defined inductively as follows:

1. $[E_i] \in \text{Solutions}(M_0)$
2. $[E] \in \text{Solutions}(M_0)$ and $\exists r \in \text{Rules}(M).t([E], r) = [E']$ if and only if $[E'] \in \text{Solutions}(M_0)$ and $([E], [E']) \in \text{Reactions}(M_0)$

In BNGL, agents are called molecules and they are specified in a similar manner as in kappa. An example of a molecule is $A(x \sim n!1)$ where the site $x$ has an internal state $n$ (separated from the site by a tilde) and a binding state is $1$ (separated by the exclamation mark). The BNGL alternatives to agent signatures are called molecule types and they are defined using the notation demonstrated in the following example: $A(x \sim n \sim b, y \sim n \sim a)$. Here, the allowed internal states of the individual sites are separated by tildes (site $x$ can have an internal state $n$ or $b$). Rules are described by the $\text{lhs} \rightarrow \text{rhs}$ notation (or $\text{lhs} \leftrightarrow \text{rhs}$ in the case of reversible rules). The individual model components (molecule types, reaction rules, seed species, observables) are in BNGL separated by the `begin` keyword and `end` keyword.

### 3 Biochemical Space

BCS provides well described biological background for mathematical models of processes taking place in specific organism. Complete BCS model provides a connection between existing ontologies and partial mathematical models. A BCS model is represented in a form of a textual file. This file offers a human readable format of BCS which can be easily edited in a dedicated editor and visualised on the website. First part of a BCS model is represented by a set of entities (to be compliant with process-algebraic frameworks we call entities agents), while the second part contains rules (abstractly represented chemical reactions defined over the set of entities). In our case study, a consortium of scientists is involved in modelling several cyanobacterial processes and in establishing of the respective BCS model.

When building the BCS model, emphasis is put on well-defined and complete annotations. Therefore, links to relevant ontologies must be specified for each entity and rule. Unique IDs provided by ontologies can help to automatically detect duplicities. IDs are also used to create hypertext links to related ontologies on the web, thus providing a one part of the already mentioned connection between ontologies and models. At this moment, links to KEGG, ChEBI, CyanoBase [14] and other databases are supported. A single entity or a rule can have multiple links to several external databases. An example is presence of a particular entity in ChEBI as well as in KEGG. In the case of annotating enzymatic rules, an EC number (here acting as a descriptor of the rule mechanism behind the catalytic reaction) is associated to the enzyme via a respective KEGG ID. For an entity that represents a protein, annotation can be enriched with a sequence of genes that encode the protein. A single link (in our case to genome browser in CyanoBase) is created for every gene
separately. If more than one gene sequence is present, additional information about every particular sequence is specified in terms of notes. In general, NOTES records carry internal information about an entity or a rule. Finally, a comma is used as a separator between records within links and notes fields. In most cases, ontologies contain general information about entities and about rule mechanisms. If this is not available, verbal description of the role of an entity or a rule can be specified directly within the particular record.

Example 3.1 Description, links, and notes information for an entity.

| DESCRIPTION: | Protein involved in hydrolysis of N-acetylated amino acids |
| LINKS: | KEGG::ec3.5.1.14, CBS::slr1653, CBS::sll0100 |
| NOTES: | ChEBI link is missing |

The fact that most fields in entity and rule definitions are tightly coupled with information from linked ontologies is the reason why we have started with describing annotation attributes. In the first place, one of these attributes is ENTITY NAME, which is taken from ontologies or follows the standard naming conventions. ENTITY ID of every entity is fixed by the consortium. KEGG ID, ChEBI ID or internal ID is used if no reasonable ID is available. IDs of rules are internal and assigned automatically.

Example 3.2 Complete information given for an atomic entity.

| ENTITY ID: | HCO3 |
| STATES: | {−, +} |
| LOCATIONS: | cyt, liq |
| COMPOSITION: |  |
| ENTITY NAME: | hydrogen carbonate |
| CLASSIFICATION: | small molecule |
| DESCRIPTION: | Plays major role in carbon concentrating mechanism (CCM). |
| LINKS: | CHEBI::17544 |
| ORGANISM: | Synechococcus elongatus PCC 7942 |

An entity in our interpretation is a bounded space (a so-called compartment) or a structural part of a specific organism. BCS covers a hierarchy of entities ranging from small molecules (atomic entities (agents)) through composite structures (structure entities (agents)) to large complex molecules (complex entities (agents)). Our goal is to make BCS as simple as possible. In existing ontologies, entities residing in several different states (oxidised, reduced, etc.) are usually treated as separate entities, thus causing the total number of entities to be enormous. To reduce this complexity, the concept of STATES is defined in BCS. All states are enclosed in curly brackets and they are comma-separated. The relationship entity–state is of the form parent–child. All information about an entity is inherited by its states unless it is specified explicitly for a particular state. The ID of an entity and its state in curly brackets form together a unique entity identifier. If no state is specified, the default value is the ‘neutral’ (ground) state.

BCS extends the traditional concept of compartmentalisation with a hierarchy at
the level of entities. A particular entity can reside in several different compartments as specified in the LOCATIONS field. Additionally, the CLASSIFICATION field specifies the type of an entity in a sense of functional or structural characterisation.

An entity can be a part of a structurally more complex entity. We consider two kinds of composite entities: structure and complex entities. Structure entity represents partially specified composite species (we employ the partial composition operator ‘|’, e.g., \( ps2(chl|yz|oec) \)), a photosystem complex partially specified with prosthetic groups of interest \( ps2(chl|yz|oec) \). Complex entity represents fully specified composite species (we employ the full composition operator ‘.’, e.g., a homodimer \( KaiC.KaiC \)). The composition of a composite entity is given in the field COMPOSITION. We employ a so-called localisation operator ‘::’ to express the fact that an entity plays a role of a location for the structurally simpler entity (e.g., chlorophyl \( chl \) located in a photosystem \( ps2 \) is written \( chl :: ps2 \)). In Example 3.3 there is a protein KaiC specified as a partial composition of two amino acids of interest – serine (S) and threonin (T). In such a configuration, serine-phosphorylated state of KaiC can be written as \( S\{p\} :: KaiC \).

**Example 3.3** Complete information given for a structure entity.

<table>
<thead>
<tr>
<th>ENTITY ID:</th>
<th>KaiC</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATES:</td>
<td>cyt</td>
</tr>
<tr>
<td>LOCATIONS:</td>
<td>S</td>
</tr>
<tr>
<td>COMPOSITION:</td>
<td>circadian clock protein kinase KaiC</td>
</tr>
<tr>
<td>ENTITY NAME:</td>
<td>enzyme</td>
</tr>
<tr>
<td>CLASSIFICATION:</td>
<td>Monomer component representing a core component of the circadian clock system.</td>
</tr>
<tr>
<td>DESCRIPTION:</td>
<td>uniprot::Q79PF4, cyanobase::Synpcc7942_1216</td>
</tr>
<tr>
<td>LINKS:</td>
<td>Synechococcus elongatus PCC 7942</td>
</tr>
</tbody>
</table>

Rules are specified by rule equations enriched with additional annotation information. When defining a rule equation, identifiers of substrates and products are used to make the notation of rules compact. Every entity appearing in a RULE EQUATION has to be followed by the localisation operator associating it with a particular compartment. This is important especially for rules that act on both sides of a membrane. That way, a rule is always precisely localised in/inbetween compartments. A natural stoichiometric coefficient can be placed before any entity in a rule equation. Irreversible and reversible rules are distinguished by the operators ‘\( \Rightarrow \)’, ‘\( \Leftrightarrow \)’. The ‘+’ symbol is used as a separator between individual substrates and individual products. A rule can also have an assigned classification. Rule classification assigns a list of higher level biophysical processes in which the rule is involved.

**Example 3.4** Complete information for a rule.
In some cases, emphasis on a detailed description leads to very complex BCS models. Abstraction of some processes is therefore needed to keep BCS models as simple as possible. To this end, rules expressing enzymatic reactions are considered in a simplified form. In fact, there should be at least two different rules describing an enzymatic reaction (one for a substrate binding and another for a catalytic step). Instead, since an enzyme is not affected during the reaction, it is affiliated to the rule as a MODIFIER. However, it is difficult to define precise meaning of a modifier in this case. We rather treat the modifier field informally as an entity which has to be present for the rule to be enabled. The exact reaction mechanism of an enzyme is not always clear and therefore it is abstracted out (see Example 3.4).

**Example 3.5** A rule employing structure entity state change.

Higher abstraction comes into account when several electrons play ‘musical chairs’ inside protein complexes. The issue is that parts of processing protein complex can have different unstable states during a short period of time. When one tries to define all rules among these proteins, combinatorial explosion of the number of states of the complex arises. Not all of these combinations are biologically correct. Even when excluding biologically inadmissible cases, the number of states is still enormous. For the purpose of BCS, we introduce a solution inspired by the enzymatic rule mentioned above. We treat a protein complex as a structure entity on which structurally simpler entities change its state (not necessarily proteins) and we abstract from background processes. We can see a particular rule as a change of a state of a structure entity (see Example 3.5).
entities and reaction rules. The annotation part has been described in [10]. Here we define the formal core of BCS and associate it with a formal semantics by means of translating BCS rules into $kappa_s$.

Model in BCS is defined in similar way as a $kappa_s$ model. First, we define syntax of expressions describing agents formally in BCS. Next, the notion of agent signature is defined that allows to specify restrictions on the general expressions. Finally, agents are used as elementary constructs in definition of BCS rules.

4.1 BCS Agents

Let $N_a$, $N_T$, $N_x$, $N_c$, $N_s$ be mutually exclusive finite sets of atomic, structure, complex, compartment, and state names, respectively.

Agents are defined hierarchically starting from atomic agents that are of two kinds: class atoms representing (abstract) class agents and object atoms representing (concrete) object agents. Class atomic agents allow us to represent compactly objects that can reside in several selected (or even all possible) states whereas object atomic agents represent concrete objects specified with the particular state. Every atomic agent must be accompanied with a physical compartment within which it is considered.

Atomic agent expressions have the following syntax:

- atomic agent $a ::= a_\bigcirc \mid a_\circ$  
- class atom $a_\bigcirc ::= \alpha^\delta ::= c$  
- object atom $a_\circ ::= \alpha\{s\} ::= c$  
- atom name $\alpha ::= n \in N_a$

From now on, we restrict ourselves to atomic agents where the state signature can be treated as a set (a state cannot occur more than once in a state signature). This restriction is motivated by the aim to keep the language as simple as possible. Treating the state signatures as multisets would lead to confusions and is actually not needed to clearly represent biological objects.

**Definition 4.1** Let $a$, $a'$ be atomic agents. We define the structural equivalence of atomic agents by claiming $a \equiv a'$ whenever $a$, $a'$ are (i) two identical object atoms or (ii) two identical class atoms that differ only in the order of states in the state signature.

**Notation 4.2**

- We denote $s \in \delta$ the fact that $s$ is included in the state signature $\delta$.
- For better readability of class atomic agents, we enclose non-trivial state signatures into curly brackets. I.e., we write $\alpha^{\{\delta\}}$ instead of $\alpha^\delta$ whenever $\delta$ contains more than one state.

Since our notion of atomic agents considers concrete objects as well as general classes of objects, we need to formally relate a class with concrete objects that instantiate it. To this end, we define compatibility relation $\ll$ that is stronger than structural equivalence.

**Definition 4.3** Let $a$, $a'$ be atomic agents. We say $a$ is compatible with $a'$, written
a ≪ a′, iff a ≡ a′ or iff there exist α, α′, s, δ, c such that a = α{s} :: c, a′ = α′δ :: c, s ∈ δ, and α = α′.

An example of a class and object atomic agents is given in Table 1. In particular, the class atom $S^{\{u,p\}}$ represents a serine amino acid that can be considered in two different states. An object atom $S\{u\}$ represents the unphosphorylated form of serine.

Next we proceed with defining structure agents. A structure agent represents a biochemical object that is composed from several known atomic agents provided that we know that such a composition is abstract and not necessarily complete. To incorporate such an abstraction of biological structures into our language, a structure agent is defined to be labelled with a unique name and it is constructed only from atomic agents considered in the same physical compartment.

The key construct of a structure agent is partial composition defined as a list of atomic agents which are considered to be relevant parts of the structure agent. We allow this list to be empty, in that case the meaning is a biological structure for which we do not know its composition.

A typical example of a structure agent is a protein where the atomic agents are individual amino acids that are of interest in the particular setting. In Table 1 there is an example of a cyanobacteria clock protein $KaiC$ specified with an interest put to the serine amino acid (here denoted by the class atomic agent $S$).

structure agent $T ::= \tau(\gamma_p) :: c$
structure name $\tau ::= n \in \mathcal{N}_T$
partial composition $\gamma_p ::= \emptyset | a | \gamma_p$

We restrict the language to structure agents where the partial composition does not contain replicated agents (stoichiometry is not considered at this level). More precisely, in every partial composition there is always at most one occurrence of an atomic agent with a name $n \in \mathcal{N}_a$. The main motivation for such a simplification is again the practical purpose of our language. The concept of partial composition is primarily considered as a rigorous identification of relevant parts of the non-trivial biochemical entity (most typically a protein). These parts are possibly subject to state changes.

Note that a compartment of a structure agent is uniquely given by the compartment specified in its parts. We restrict ourselves to structure agents where all atomic agents in the partial composition have the same compartment. Assuming this restriction, we can shorten the notation by omitting compartments in the atomic agents of a partial composition.

Notation 4.4
- We denote $\tau(\ldots | a | \ldots) :: c$ a structure agent of the name $\tau$ such that an atomic agent $a$ makes its part.
- We denote $a \in \gamma_p$ the fact that $\gamma_p$ includes the atomic agent $a$.
- The agent of the form $\tau(\emptyset) :: c$ is usually written as $\tau :: c$.
- A structure agent $\tau(\gamma_p) :: c$ is usually written $\tau(\alpha_1|\alpha_2|\ldots|\alpha_n) :: c$ where $\alpha_1,\ldots,\alpha_n$
are names of all agents in $\gamma_p$ such that $\gamma_p = a_1|...|a_n$ where each agent $a_i$ is either of the form $a_i = \alpha_i^{\delta_i} :: c$ or $a_i = \alpha_i\{s_i\} :: c$ for some $\delta_i$ a state signature, $s_i$ a state, and $c$ a compartment shared among all agents in $\gamma_p$.

**Definition 4.5** Let $T, T'$ be structure agents. We define the structural equivalence of structure agents by claiming $T \equiv T'$ iff there exist $\tau, \tau', \gamma_p, \gamma'_p, c$ such that $T = \tau(\gamma_p) :: c, T' = \tau'(\gamma'_p) :: c, \tau = \tau'$ and $\gamma_p, \gamma'_p$ are equal or differ only in the order of its components (the operator ‘|’ is considered associative and commutative).

As a representative of a class of structurally equivalent structure agents we consider the agent $\tau(\gamma_p)$ where the agents in $\gamma_p$ are lexicographically ordered by names. Since atomic agents cannot be repeated in a structure agent, such an order is total.

**Definition 4.6** Let $T, T'$ be structure agents. We say $T$ is compatible with $T'$, written $T \triangleleft T'$, iff either $T \equiv T'$ or for each atomic agent $a$ such that $T = \tau(...|a|...) :: c$ there exists an atomic agent $a'$ such that $T' = \tau'(...|a'|...) :: c, \tau = \tau'$ and $a \triangleleft a'$.

In the following we define the last step in the hierarchy of agents. In particular, we define complex agents. A complex agent represents a non-trivial composite biochemical object that is (inductively) constructed from already known biological objects. In common rule-based languages this is typically defined by introducing some kind of bonds between individual biochemical objects. In BCS we abstract from detailed specification of bonds and we rather assume a complex as a coexistence of certain objects in a particular group. Such a group can be optionally referred to by a unique name. A complex agent is constructed from structure agents where all are required to reside in the same compartment $c$.

A complex agent is given either directly as an expression inductively built by applying coexistence operator ‘|’ to structure agents or indirectly as a name referring to a separate set of definitions of complex agents (incorporated in the notion of agent signature). We use that approach because we do not want to overcomplicate complex agent expressions.

The key element of a complex agent is full composition describing inductively constructed coexistence expressions from existing agents. We restrict ourselves to full compositions where all agents reside in the same compartment.

$$\begin{align*}
\text{complex agent} & : x ::= \gamma_f :: c | n \in \mathbb{N} : c \\
\text{full composition} & : \gamma_f ::= T.T | T.\gamma_f
\end{align*}$$

In contrast to partial compositions, we allow replication at the level of full compositions (an agent of a certain name can appear more than once in a full composition). Moreover, names of complex agents are not associated with particular full compositions at the level of agent expressions. This is done at the level of agent signatures (see Section 4.2).

Note that in similar way as in the case of structure agents, we restrict the formalism to complex agents where the compartment is the same for all agents inside the respective full composition.

**Notation 4.7**
<table>
<thead>
<tr>
<th><strong>symbol</strong></th>
<th><strong>description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>$S^{(u,p)} :: cyt$</td>
<td>Serine (S) in two possible states phosphorylated (p) and unphosphorylated (u) existing in compartment cytosol (cyt).</td>
</tr>
<tr>
<td>$S{u} :: cyt$</td>
<td>Serine (S) in state unphosphorylated (u) existing in compartment cytosol (cyt).</td>
</tr>
<tr>
<td>Protein KaiC containing Serine (S) in its partial composition $\gamma_p$.</td>
<td>It is possible to obtain two different derivations $KAiC(S{u}) :: cyt$ and $KAiC(S{p}) :: cyt$.</td>
</tr>
<tr>
<td>$KAiC(S^{(u,p)}) :: cyt$</td>
<td>Complex of six KaiC structure agents (order does not matter).</td>
</tr>
</tbody>
</table>

### Examples of different forms of an agent.

1. Let $X = \gamma_f :: c$ for some full composition $\gamma_f$. We denote $T \in X$ the fact that $T$ is a structure agent included in $\gamma_f$. Moreover, we denote $\#T[X]$ the number of occurrences of $T$ in $\gamma_f$.

2. For a complex agent $X = \gamma_f :: c$ where each item $x \in X$ is an agent assigned to a compartment $c$, we can use simplified notation that omits the compartment suffix ‘:: $c$’ in individual agents of $\gamma_f$.

Next we define structural equivalence of complex agents. We employ set-based approach to aggregate complex agents into equivalence classes. In particular, at that level we achieve commutativity and associativity of the operator ‘.’.

**Definition 4.8** Let $X, X'$ be complex agents. We define structural equivalence of complex agents by claiming $X \equiv X'$ iff either of the following conditions holds:

(i) There exist a compartment $c$ and $n, n' \in \mathbb{N}$ such that $X = n :: c, X' = n' :: c$ and $n = n'$.

(ii) If both $X, X'$ are specified as full compositions then the following two conditions must be satisfied:

- for each $T \in X$ there exists $T' \in X'$ such that $T \equiv T'$ and $\#T[X] = \#T'[X']$,
- for each $T' \in X'$ there exists $T \in X$ such that $T' \equiv T$ and $\#T'[X'] = \#T[X]$.

An example of a complex agent is given in Table 1 where the given complex agent expression represents a large set of hexamers composed from $KAiC$ molecules each considered in arbitrary state.

**Remark 4.9** From now on, we always consider a lexicographically ordered agent as a representative of a class of structurally equivalent agents. Since agents are defined hierarchically, lexicographical order is applied recursively to all nested agents. This allows us to always have a clearly defined unique representative.

### 4.2 BCS Agent Signatures

The language of agents defined in the previous section gives us a formal way how to encode biochemical objects at several levels of hierarchy and abstraction. The notion of structure agents allows to generate arbitrary partial compositions. Practically, we need to restrict the construction of composite biochemical objects by giving a set of constraints reflecting our understanding of biological objects and the desired level of abstraction. This can be achieved by assigning every structure agent name
with a maximal partial composition that gives the restriction on structure agents that can be considered.

Similarly, the set of complex agents also needs to be restricted by specifying the catalogue of complex biochemical objects that can appear in the considered biochemical space. This can be achieved by assigning every complex agent name with a full composition that provides its definition. This allows us to name biological compounds, e.g., H$_2$O, and specify their clear definition under the coexistence abstraction, e.g., H.$H$.O.

**Definition 4.10** We say a pair $(\Sigma_\tau, \Sigma_x)$ is agent signature where $\Sigma_\tau$ and $\Sigma_x$ are relations representing constraints on the construction of structure agents and complex agents, respectively, defined in the following way:

- Every $\tau \in \mathcal{N}_T$ is assigned some partial composition $\gamma_p$, $(\tau, \gamma_p) \in \Sigma_\tau$.
- Every $n \in \mathcal{N}_x$ is assigned some full composition $\gamma_f$, $(n, \gamma_f) \in \Sigma_x$.

**Definition 4.11** We say a structure agent $T = \tau(\gamma_p)$ satisfies an agent signature $(\Sigma_\tau, \Sigma_x)$, written $T \models (\Sigma_\tau, \Sigma_x)$, iff for every atomic agent $a \in \gamma_p$ there exists $a' \in \gamma'_p$ such that $a \prec a'$ and $(\tau, \gamma'_p) \in \Sigma_\tau$.

**Remark 4.12** Note that for every $\tau \in \mathcal{N}_T$ the pair $(\tau, \gamma_p) \in \Sigma_\tau$ specifies the most general structure agent of the name $\tau$ with respect to the relation $\prec$. The meaning of the agent $\tau(\emptyset) :: c$ (simply written $\tau :: c$) is a short form for the most general structure agent of the name $\tau$ specified in the signature. This is just the agent given by the partial composition $\gamma_p$. Any agent $\tau(\gamma'_p)$ where $\gamma'_p$ is constructed from $\gamma_p$ by omitting some atomic agents makes a short form for $\tau(\gamma_p)$.

Note that with respect to Remark 4.12 we can write structure agents very compactly. E.g., assume $(\tau, a_1\{s\}a_2^{s_1,s_2}) \in \Sigma_\tau$ then $\tau(\emptyset)$ is interpreted as $\tau(a_1\{s\}a_2^{s_1,s_2})$, $\tau(a_2\{s\}a_2\{s_1\})$ as $\tau(a_1\{s\}a_2\{s_1\})$, etc.

Next we define expansion of named complex agent with respect to the given signature. In particular, we treat each pair $(n, \gamma_f) \in \Sigma_x$ as a specification of a full composition that is named $n$. Expansion then means replacing every complex agent name with the respective full composition. Finally, agents expanded with respect to a given signature are treated as agents satisfying the signature.

**Definition 4.13** Let $(\Sigma_\tau, \Sigma_x)$ be a signature. Every complex agent $x = n \in \mathcal{N}_x$ is expanded with respect to an agent signature $(\Sigma_\tau, \Sigma_x)$, written $x[(\Sigma_\tau, \Sigma_x)]$ and defined $x[(\Sigma_\tau, \Sigma_x)] = \gamma_f$ where $(n, \gamma_f) \in \Sigma_x$.

**Definition 4.14** We say a complex agent $x = \gamma_f$ satisfies an agent signature $(\Sigma_\tau, \Sigma_x)$, written $x \models (\Sigma_\tau, \Sigma_x)$, iff every structure agent $T \in \gamma_f$ satisfies $T \models (\Sigma_\tau, \Sigma_x)$. A complex agent $x = n \in \mathcal{N}_x$ satisfies a signature $(\Sigma_\tau, \Sigma_x)$ iff $x[(\Sigma_\tau, \Sigma_x)] \models (\Sigma_\tau, \Sigma_x)$.

### 4.3 BCS Rules

At this point, we proceed to define the set of BCS rules. In contrast to $\kappaappa_s$, a BCS rule has more complicated structure. This is due to the fact that BCS
goes closer to traditional formalism of chemical reactions, in particular, BCS rules consider stoichiometry and compartmentalisation of reacting species. Moreover, to a certain extent we introduce variables in rule expressions allowing us to compact specification of repeating objects.

The list of rules $\mathcal{R}$ is defined by the following syntax:

$$
\begin{align*}
\text{rules} & \quad \mathcal{R} ::= \emptyset \mid r, \mathcal{R} \\
\text{rule equation} & \quad r ::= \Gamma \circ \Gamma \\
\text{direction} & \quad \circ ::= \Rightarrow \mid \Leftrightarrow \\
\text{rule expression} & \quad \Gamma ::= \emptyset \mid \rho \epsilon :: c \mid \rho \epsilon :: \Gamma + \Gamma \\
\text{stoichiometry} & \quad \rho ::= n \in \mathbb{N}^+ \\
\text{rule expression item} & \quad \epsilon ::= \epsilon_1 \mid \epsilon_2 \mid \epsilon_3 \\
\text{basic rule agent} & \quad \epsilon_1 ::= a \mid T \mid X \\
\text{shallow rule agent} & \quad \epsilon_2 ::= a :: T \mid T :: X \\
\text{deep rule agent} & \quad \epsilon_3 ::= a :: T :: X
\end{align*}
$$

We assume that a single rule cannot appear more than once in the list $\mathcal{R}$ (every rule must be unique). In relation to that, we can use the notation $r \in \mathcal{R}$ to refer to rules in $\mathcal{R}$. See Section 7 for examples of several rules.

Rule expressions allow more extensive syntax in terms of the localisation operator ‘::’. The localisation operator is intended for allowing an alternative way of expressing the hierarchically constructed agents. The main idea is to allow zooming into individual parts of a complex or a structure agent. E.g., for a structure agent $\tau(\alpha_1{s})\alpha_2^{(s,t)} :: c$ residing in compartment $c$ we can use the notation $\alpha_2{t} :: \tau(\alpha_1{s})\alpha_2^{(s,t)} :: c$ to refer explicitly to a concretisation of its subagent $\alpha_2$. This notation is fully equivalent with the original form $\tau(\alpha_1{s})\alpha_2{t}$ and can be therefore considered as an alternative way to concretise a structure agent.

Similarly, the concept of localisation is applied also to complex agents. E.g., for a complex agent $A(\alpha_1{s}).B(\alpha_2^{(s,t)}) :: c$ we can zoom to some of its components and express its concretisation such as $B(\alpha_2{t}) :: A(\alpha_1{s}).B(\alpha_2^{(s,t)}) :: c$. In this case, the notation $B(\alpha_2{t}) :: A(\alpha_1{s}).B(\alpha_2^{(s,t)})$ is equivalent to the complex agent $A(\alpha_1{s}).B(\alpha_2{t})$.

In every rule subexpression $\rho \epsilon :: c$ the compartment $c$ makes the scope for every agent appearing in $\epsilon$. In particular, every agent inside $\epsilon$ is assumed to be assigned the compartment $c$.

To simplify the resulting language to construct reasonable expressions only, we restrict ourselves to rules where the operator ‘::’ respects constraints given in Definition 4.15.

**Definition 4.15** Let $\epsilon$ be a rule expression item that appears in a rule $r \in \mathcal{R}$. The rule expression $\epsilon$ is well-defined iff the following constrains are satisfied:

(i) If $a :: \tau(\gamma_p)$ is a subexpression of $\epsilon$ for some $a, \tau, \gamma_p$ then there must exist $a' \in \gamma_p$ such that $a \triangleleft a'$.

(ii) If $T :: X$ is a subexpression of $\epsilon$ for some $T, X$ then there must exist $T' \in X$ such that $T \triangleleft T'$.

Every rule agent in a shallow or deep form can be translated to an equivalent basic form. Formally, this is given in Lemma 4.16.
Lemma 4.16 (Rule Flattening) Let $(\Sigma_T, \Sigma_x)$ be a signature and $R$ a set of rules. Every rule $r \in R$ that includes some rule agents in shallow or deep form can be reduced to a rule $r' \in R$ where every rule agent is in basic form. For every rule agent $\epsilon$ in $r$, the reduction is done by replacing $\epsilon$ with $\epsilon'$ in the following way:

1. If $\epsilon = a :: T$ where $T = \tau(\gamma_p)$ for some $\tau, \gamma_p$ then there must exist $a' \in \gamma_p$ such that $a \triangleleft a'$. Then we set $\epsilon' = \tau(\gamma'_p)$ where $\gamma'_p$ is constructed from $\gamma_p$ by replacing $a' \in \gamma_p$ with $a$.
2. If $\epsilon = T :: X$ where $X = \gamma_f$ then there must exist $T' \in \gamma_f$ such that $T \triangleleft T'$. Then we set $\epsilon' = \gamma'_f$ where $\gamma'_f$ is constructed from $\gamma_f$ by replacing $T' \in \gamma_f$ with $T$.
3. If $\epsilon = a :: T :: X$ then the steps (i,ii) above are applied successively.

Definition 4.17 We say that a rule $r \in R$ satisfies agent signature $(\Sigma_T, \Sigma_x)$, written $r \models (\Sigma_T, \Sigma_x)$, iff every structure or complex agent that appears as a rule agent in $r$ satisfies agent signature $(\Sigma_T, \Sigma_x)$. To increase succinctness, we extend the language with a variable $?\nu$. A variable can be assigned to any rule in place of an agent. Evaluation of a variable within a rule expression. The domain of a variable is assumed to be considered as a set (values are not repeated). An example is given in Example 7.2. The extended syntax is the following:

- **Rule Equation:** $r' ::= r | \Gamma \odot \Gamma ; \text{var}$
- **Variable:** $\text{var} ::= \emptyset | ?\nu = \{\phi\} | ?\nu_1 = \{\phi_1\} | ?\nu_2 = \{\phi_2\} | ?\nu_3 = \{\phi_3\}$
- **Variable Value:** $\phi ::= \phi_1 | \phi_2 | \phi_3$
- **Atomic Variable Value:** $\phi_1 ::= a, \phi_1 | a$
- **Structure Variable Value:** $\phi_2 ::= T, \phi_2 | T$
- **Complex Variable Value:** $\phi_3 ::= X, \phi_3 | X$
- **Extended Basic Rule Agent:** $\epsilon'_1 ::= \epsilon_1 | ?\nu$
- **Extended Shallow Rule Agent:** $\epsilon'_2 ::= \epsilon_2 | ?\nu_1 :: T | a :: ?\nu_2 | ?\nu_2 :: X | T :: ?\nu_3$
- **Extended Deep Rule Agent:** $\epsilon'_3 ::= \epsilon_3 | ?\nu_1 :: T :: X | a :: ?\nu_2 :: X | a :: T :: ?\nu_3$

Finally, we define the notion of a BCS model that is given by a signature and a set of rules.

Definition 4.18 A **BCS model** $M$ is a tuple $((\Sigma_T, \Sigma_x), R)$ such that every $r \in R$ it holds that $r \models (\Sigma_T, \Sigma_x)$.

# 5 Translation to $kappa_s$

To define semantics for BCS language, we give an algorithm that translates a given BCS model $M$ to a $kappa_s$ model $M$. We assume the model $M$ is normalised using the following procedures:

1. all rules are flattened by employing Lemma 4.16,
2. every bidirectional rule is replaced by the two respective unidirectional rules,
(iii) every rule with variables is replaced by the set of rules generated by expanding all acceptable values for every variable.

Algorithm 1. Transform a BCS model $M$ to a kappa$_s$ model $M$.

1: function toKappa($M = (\Sigma_r, \Sigma_x, \mathcal{R})$)  
2: $\Sigma_\kappa, I, \mathcal{R}, \Lambda := \emptyset$  
3: for all $r \in \mathcal{R}$ do  
4: for all $\Gamma \in \{\Gamma_I, \Gamma_r\}$ such that $r = \Gamma_I \Rightarrow \Gamma_r$ do  
5: $E := \emptyset$  
6: for all $\rho \in c \in \Gamma$ do  
7: if $c$ has the form $a\{s\}$ then  
8: $E \leftarrow \text{translateAtom}(c :: c)$  
9: if $c$ has the form $\tau(\gamma_p)$ then  
10: $E \leftarrow \text{translateStructure}(c :: c)$  
11: if $c$ has the form $X$ then  
12: $E := \text{translateComplex}(c :: c, E)$  
13: construct a kappa$_s$ rule $r_\kappa$ from the two resulting sets $E$ obtained for $\Gamma_I, \Gamma_r$  
14: $\mathcal{R} \leftarrow r_\kappa$  
15: return $M = (\Sigma_\kappa, I, \mathcal{R})$  

Algorithm 1 takes a BCS model $M$ and returns a kappa$_s$ model $M$. It uses three subroutines that modify respective types of BCS agents. Algorithm 2 translates an atomic agent directly by extending an agent name with a compartment name and adding a site $p$. Algorithm 3 translates a structure agent where each atomic agent in its partial composition is encoded as a unique site. Finally, algorithm 4 translates a complex agent where each structure agent in the respective full composition is treated as a kappa$_s$ agent. Since BCS does not provide binding sites, we fix linear binding (see Section 6 for further discussion).

Algorithm 2. Transforms an atomic agent to a kappa$_s$ agent.

1: function translateAtom($a\{s\} :: c$)  
2: $\Lambda \leftarrow a.c$  
3: $\Sigma_\kappa \leftarrow \langle a.c, \{p\} \rangle$  
4: $\sigma(a.c) \leftarrow p$s  
5: return $a.c(\sigma)$  

Algorithm 3. Function transforms structure agent to kappa$_s$ agent.

1: function translateStructure($\tau(\gamma_p) :: c$)  
2: $\Lambda \leftarrow \tau.c$  
3: for all $a\{s\} :: c \in \gamma_p$ do  
4: $\Sigma_\kappa \leftarrow \langle \tau.c, \{a\} \rangle$  
5: $\sigma(\tau.c) \leftarrow a$s  
6: return $\tau.c(\sigma)$  

Algorithm 4. Transforms a complex agent to a kappa$_s$ agent.

1: function translateComplex($X :: c, E$)  
2: $i := 0$  
3: for all $T \in X$ do  
4: agent $a := \text{translateStructure}(T)$  
5: $\Sigma_\kappa \leftarrow \langle a, \{l, r\} \rangle$  
6: if $i \neq 0$ then  
7: $l \leftarrow \langle (a, l), \{i\} \rangle$  
8: $\sigma(a) \leftarrow l$s  
9: $i := i + 1$  
10: if $i \neq \#T[X]$ then  
11: $l \leftarrow \langle (a, r), \{i\} \rangle$  
12: $\sigma(a) \leftarrow r$s  
13: $E \leftarrow a(\sigma)$  
14: return $E$
6 Comparison of BCS and BNGL

It is necessary to note that BCS currently does not provide quantitative semantics. It just describes the system structure and the relationships between entities.

Another issue is if there is a rule containing a modifier, there are two options how to express it in BNGL. The first option is to add the modifier to both sides of the rule, the second is to include it (quantitatively) in the reaction rate function. BCS has to employ the first option. There is also an alternative solution in BCS – the field \textit{MODIFIER} in the BCS rule annotation record.

BCS does not provide binding sites. This is caused by the fact it also does not provide specification of a bond. In BNGL, binding makes a binary operation between two components and the bond must be always specified by using operators ‘.’ and ‘!’ where each bond has a unique ID inside a complex. This detailed notion is not present in BCS since we want to abstract such details.

However, this kind of abstraction introduces the inability to distinguish two complexes composed from the same subparts (e.g. proteins). For example, consider a protein \(P\) with a single binding site. A complex formed from \(n\) proteins \(P\) can be created from \(n-1\) bonds (linear conformation) to \(NPI(n) = \frac{n(n-1)}{2}\), which is the maximal number of possible interactions inside the complex (assuming only one bond between two proteins is possible). It follows that a complex \(C_{BCS}\) formed from \(n\) proteins \(P\) in BCS is the set of all possible structural conformations \(C_i\) of the complex in BNGL where all proteins are considered:

\[
C_{BCS} = \bigcup_{i=0}^{m} C_i, \text{ where } m \text{ is the number of connected graphs on } n \text{ nodes.}
\]

Stringency of a rule makes a relevant difference. Stringency stands for degree of universality or specificity of the rule, i.e. the width of the applicability. In both languages, this can be solved by context of the rule. However, it is not always suitable to list the whole context. An example can be phosphorylation in circadian clock (Example 7.2). It can occur on each KaiC protein which is included in a complex. For this purpose there is ‘site!’ notation in BNGL which requires the protein to be in a bound state. Since BCS does not provide binding sites, this cannot be used.

To this end, we employ the localisation operator ‘::’ in rule agents. It allows to nest rule agents to strengthen the stringency. Moreover, we have introduced \textit{variables} in BCS. A variable in a reactant is denoted \(?\nu\) and can be specified as a set of atomic, structure or complex agents to which the rule can be applied.

The last fact that is worth noting is construction of complex structures. In BNGL, each complex is identified with an exact structural notation which does not allow hierarchical construction. BCS provides the notion of structure and complex agents, this allows to form a hierarchy of the agents. Additionally, when defining a rule with quantities of interacting entities, in BNGL it is necessary to enumerate all of them whereas in BCS the stoichiometry is allowed in standard way.
6.1 BCS and BNGL translation

It is possible to translate from BCS to BNGL. This can be achieved by the application of finite set of transformation steps. The procedure is analogous to translation to kappa\textsubscript{s} (Section 5).

Translation BNGL to BCS is also possible, but the bond information is discarded in the process. In particular, all binding operations have to be removed. The only problem is the ‘!+’ notation in BNGL which requires a bond for a entity. This kind of bond has a high level of abstraction. For this reason we cannot translate such a rule. However, every rule in BNGL with ‘!+’ can be expanded to finite number of rules where this operator is omitted. Instead of an unknown bond, there are enumerated rules each accompanied with a known binding partner. In that case, the variable $?\nu$ is added to the BCS rule containing all the enumerated binding partners.

7 Case Study

BCS makes a part of CMP and is implemented at e-cyanobacterium.org and currently covers several functional modules of cyanobacteria. To support translation between BCS and BNGL, we have implemented a set of scripts\footnote{http://www.e-cyanobacterium.org/downloads/} allowing to translate a BCS model to BNGL and vice versa.

7.1 Metabolism

Metabolism forms the backbone of cyanobacteria cellular processes and in BCS covers the largest part of cyanobacteria network. We distinguish two groups of entities in metabolism – enzymes and metabolites. Enzymes drive metabolic reactions and therefore are assigned to rules as modifiers. On the other hand, metabolites are small molecules playing a role of substrates or products of metabolic rules with no enzymatic function. Both groups are involved in rules which occur mostly in the cell cytoplasm, therefore the majority of their entities uses cytoplasm as a compartment.

Example 7.1 A rule from metabolism of cyanobacteria. It is visualised in Figure 2 in the upper left part.

\begin{center}
\begin{tabular}{|l|l|}
\hline
RULE ID: & (S)-malate:NAD\{\textasteriskcentered\} oxidoreductase \\
RULE EQUATION: & malate :: cyt + NAD\{\textasteriskcentered\} :: cyt ⇔ oxaloacetate :: cyt + \\
& + NADH :: cyt + H\{\textasteriskcentered\} :: cyt \\
MODIFIER: & malate oxidation \\
RULE NAME: & \\
CLASSIFICATION: & oxidation, reduction \\
DESCRIPTION: & Process is involved in citric acid cycle. Malate is oxidised to oxaloacetate producing NADH from NAD\{\textasteriskcentered\}. \\
\hline
\end{tabular}
\end{center}

In metabolism, there are approximately 770 rules. Despite the fact that there are plenty of molecules, the rules are very specific. In our proposed rule-based language
it means the mapping of reactions to rules is almost one-to-one (reaction-like rules). The stringency of rules is high which is what allows them to be applied only to a narrow group of molecules. It causes that compaction of metabolism in rules brings almost no benefits.

7.2 Circadian clock

Circadian clock is one of the most complex processes in cyanobacteria BCS. Its core is formed by three proteins KaiA, KaiB and KaiC. Moreover, KaiC contains two phosphorylation sites serine S431 and threonine T432. These sites can be phosphorylated independently, but only if KaiC is in a complex. All these proteins can interact with each other in predetermined ways and form specific complexes. All processes inside the cell are then controlled by periodical formation/dissociation and (de)phosphorylation of these complexes.

Example 7.2 Serine (de)phosphorylation on KaiC protein. In Figure 3 it is (also with threonine phosphorylation) responsible for all short cycles.

```
RULE ID: serine (de)phosph.
RULE EQUATION:
S\{u\} :: KaiC :: ?X :: cyt ⇔ S\{p\} :: KaiC :: ?X :: cyt ;
?X = \{KaiC6, KaiA2C6, KaiB6C6, KaiA4C6, KaiA6B6C6\}

MODIFIER:
RULE NAME: Serine phosphorylation and dephosphorylation
CLASSIFICATION: phosphorylation, dephosphorylation
DESCRIPTION:
KaiC molecule is phosphorylated/dephosphorylated on serine amino acid. This process can appear whenever KaiC is in one of the complexes enumerated in variable X.
```

Owing to the fact the proteins can form homohexamers or smaller complexes, and each of these complexes can interact with others, it causes combinatorial ex-

Fig. 2. Part of the reaction scheme of metabolism in cyanobacteria [12].
plosion. Together there is possible formation of six different complexes containing $\text{KaiC}$: $\text{KaiC}_6$, $\text{KaiB}_6\text{C}_6$, $\text{KaiA}_2\text{C}_6$, $\text{KaiA}_4\text{C}_6$, $\text{KaiA}_4\text{B}_6\text{C}_6$ and $\text{KaiA}_6\text{B}_6\text{C}_6$. Each protein $\text{KaiC}$ can occur in four different states because of the two phosphorylation sites. Considering all six complexes and also other rules in circadian clock, we obtain combinatorial explosion of different species in the system. To achieve representation of the whole system it is inefficient to enumerate each single conformation. To this end, we employ the capability of BCS rules.

**Example 7.3** Formation of $\text{KaiB}_6\text{C}_6$ complex is important for circadian clock. It can be seen in the upper left part of Figure 3, where it forms the bigger cycle (with all other complex formation rules).

| RULE ID: | KaiB6C6 form./diss. |
| RULE EQUATION: | 6 $\text{KaiB} :: \text{cgt} + \text{KaiC}_6 :: \text{cgt} \Leftrightarrow \text{KaiB}_6\text{C}_6 :: \text{cgt}$ |
| MODIFIER: | |
| RULE NAME: | KaiB6C6 complex formation and dissociation |
| CLASSIFICATION: | complex formation, dissociation |
| DESCRIPTION: | Formation of complex from six $\text{KaiB}$ molecules and $\text{KaiC}$ hexamer and its dissociation. $\text{KaiC}_6$ represents specification of complex composed from six $\text{KaiC}$ proteins, $\text{KaiB}_6\text{C}_6$ complex of six $\text{KaiC}$ and six $\text{KaiB}$ respectively. |
| LINKS: | doi::10.1093/emboj/18.5.1137, doi::10.1016/j.febslet.2009.11.021 |

In BCS we have achieved complete, human readable representation of circadian clock using only 17 rules (examples are rules in Example 7.2 and Example 7.3). Regarding the defined agents, it gives us over 500 different distinguishable entities, while in BNGL similar number of rules describing the same system gives us almost 25000 entities.
7.3 Photosynthesis

Photosynthesis represents part of BCS of cyanobacteria. The process occurs in a specific folds of the cell membrane called thylakoid membrane. Photosynthesis serves as the source of energy taken from light and transferred into production of ATP and NADPH molecules with oxygen resulting as a by-product.

![Fig. 4. Reaction scheme of photosynthesis in cyanobacteria. The lumen processes are displayed under thylakoid membrane while stroma processes are above.](image)

**Example 7.4** A rule from photosynthesis. Oxidation reaction on PSII.

| RULE ID: | PSII oxidation |
| RULE EQUATION: | $\text{ps2(oec}\{3+\} \ | yz\{+\}) : : \text{tlm} \Leftrightarrow \text{ps2(oec}\{4+\} \ | yz\{n\}) : : \text{tlm}$ |
| MODIFIER: | |
| RULE NAME: | oxidation from S3 to S4 of oxygen evolving complex oxidation |
| CLASSIFICATION: | Oxidation occurring on photosystem II. Electron is transferred from oxygen evolving complex oec to active tyrosine yz. |
| DESCRIPTION: | |

Entities of photosynthesis BCS are represented by several complex proteins (enzymes) residing on the thylakoid membrane (tlm) in the cell. Since the thylakoid membrane encloses the inner-membrane space called lumen (lum) where $H_2O$ molecules are processed, there are basically three locations defined for this set of entities. Rules occurring in the lumen, cytosol and in-between the thylakoid membrane and these locations have classical form. However, electron transfer reactions occurring in the structure of complex processes lead to combinatorial explosion of all possible conformations.

Photosynthesis is constructed from approximately 30 agent definitions which are interacting in over 60 rules. From the rule-based point of view, this representation is somewhere between circadian clock (Section 7.2) and metabolism (Section 7.1). It means the number of generated distinguishable entities arises compared to defined agents, but not as dramatically as in circadian clock. However, photosynthesis is a good example of rule-based process.
8 Conclusions

We have lifted the annotation format BCS to a formal language compatible with well-established rule-based languages. We have given an automated support for translating between BCS and BNGL. Currently, BCS is used on the portal e-cyanobacterium.org for description of cyanobacteria processes. In case study section we have shown the language is suitable for rule-based systems as well as reaction-based systems. For future work we plan to define an operational semantics directly without an intermediate format. This will enable implicit description of the model states space and allow to gain from the compact representation and take the advantages of on-the-fly model checking.

References


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; BioSi [23] and SPIM [22] use very elemental asymmetric binary synchronisation primitives; BioPEPA [5] adapts process-algebraic formalism 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.

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

Definition 2.2

- Let $S$ be a set. By $\Omega^S$ we denote the set of all possible finite multisets $(A, m)$ such that $A \subseteq S$.
- 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) $L$ 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 $L = (S, A, T, s_0)$ 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 $X \cdot Y$, is defined as: $X \cdot Y = (x_1, \ldots, x_n, y_1, \ldots, y_m)$.

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 $\sum_{i=1}^n T_i$ is defined as: $\sum_{i=1}^n T_i = T_1 \cdot T_2 \cdot \ldots \cdot T_n$.

2.2 Objects definition

Let $N_A, N_T, N_S, N_c$ be mutually exclusive finite sets of atomic names, structure names, states, and compartments respectively. Moreover, $\varepsilon$ 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 $\Sigma_A : N_A \to 2^{N_s}$ that associates each atomic name to a set of state names. Similarly, structure signature is a function $\Sigma_T : N_T \to 2^{N_A}$ 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, \( \{ S \rightarrow [u,p], Q \rightarrow [a,i] \} \) is an atomic signature and \( \{ KaiC \rightarrow \{S,Q\}, KaiB \rightarrow \emptyset \} \) is a structure signature.

### 2.2.2 Atomic agent

**Definition 2.8 (Atomic agent)** An atomic agent \( \mathbf{A} \) is a pair \((\eta, \delta)\) where \( \eta \in \mathcal{N}_\mathcal{A} \) is a name and \( \delta \in \mathcal{N}_\delta \cup \{\varepsilon\} \) is a state. The name and the state of the agent \( \mathbf{A} \) is usually denoted by \( \eta(\mathbf{A}) \) and \( \delta(\mathbf{A}) \), 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 \( \varepsilon \) state) is given by signature \( \Sigma_\mathcal{A}(\eta) \).

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

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

**Notation 2.10** We use the symbol \( \mathcal{A} \) 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 \( \mathbf{A}_1 = \{S,u\}, \) written as \( S[u] \), and \( \mathbf{A}_2 = \{Q,\varepsilon\}, \) written as \( Q[\varepsilon] \). Note the meaning of \( \varepsilon \) is the state is unknown or not important to be considered in a given context.

**Definition 2.11 (Compatibility of atomic agents)** Let \( \mathbf{A}_1, \mathbf{A}_2 \) be atomic agents. The agent \( \mathbf{A}_1 \) is compatible with agent \( \mathbf{A}_2 \), written \( \mathbf{A}_1 \triangleleft \mathbf{A}_2 \) if either \( \mathbf{A}_1 = \mathbf{A}_2 \) or \( \eta(\mathbf{A}_1) = \eta(\mathbf{A}_2) \land \delta(\mathbf{A}_1) = \varepsilon \).

Compatibility of atomic agents is a key property defined between agents. An agent is compatible with another 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 \( \mathbf{A} \in \mathcal{A} \) be an atomic agent. We say the agent \( \mathbf{A} \) is fully specified, written \( \triangleleft \mathbf{A} \), iff \( \forall a \in \mathcal{A} \) such that \( \mathbf{A}' \neq \mathbf{A} : \lnot(\mathbf{A}' \triangleleft \mathbf{A}) \).

### 2.2.3 Structure agent

**Definition 2.13 (Structure agent)** We define a structure agent \( \mathbf{T} \) as a pair \((\eta, \gamma)\) where \( \eta \in \mathcal{N}_\mathcal{T} \) is a name and \( \gamma \subseteq \mathcal{A} \) is a set of atomic agents called partial composition such that \( \forall \mathbf{A}, \mathbf{A}' \in \gamma : \eta(\mathbf{A}) \neq \eta(\mathbf{A}') \). The name and the partial composition of the agent \( \mathbf{T} \) is usually denoted by \( \eta(\mathbf{T}) \) and \( \gamma(\mathbf{T}) \), 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 \( \mathbf{T}, \mathbf{T}' \) be structure agents. \( \mathbf{T} \) is equal to \( \mathbf{T}' \), written \( \mathbf{T} = \mathbf{T}' \), iff \( \eta(\mathbf{T}) = \eta(\mathbf{T}') \land \gamma(\mathbf{T}) = \gamma(\mathbf{T}') \).

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 \( \mathcal{T} \) 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 \( \mathbf{T}_1 = (K,[\{S,p\},\{Q,i\}]) \), written as \( K(S[p],Q[i]) \), and \( \mathbf{T}_2 = (K,[\{Q,u\}]) \), written as \( K(Q[u]) \).

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 \( \gamma, \gamma' \) be partial compositions. We define difference of partial compositions \( \gamma \ominus \gamma' = \{ A | A \in \gamma \land A \notin \gamma \cap \gamma' \} \) where \( \gamma \cap \gamma' = \{ A | A \in \gamma \land \exists A' \in \gamma' : \eta(A') = \eta(A) \} \).
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 $T_1 \triangleleft T_2$, iff either $T_1 = T_2$ or $\eta(T_1) = \eta(T_2) \wedge \forall A_1 \in \eta(T_1) \exists A_2 \in \eta(T_2) : A_1 \triangleleft A_2$.

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 $T \in T$ be a complex agent. We say the agent $T$ is fully specified, written $\Delta T$, iff $\forall T' \in T$ such that $T' \neq T : -(T' \triangleleft T)$.

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 $(\mu, \text{com})$ where $\mu \in (A \cup T)^\eta$ is a sequence of agents, $\text{com} \in \mathcal{X}$ 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 kind 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 $\mu(X) = \mu(X') \wedge \Omega(X) = \Omega(X')$.

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

Notation 2.21 We use the symbol $\mathcal{X}$ 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 $X_1 \triangleleft X_2$, 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 $\forall i \in [1, n] : \mu_i(X) \triangleleft \mu_i(X')$, 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 $X \in \mathcal{X}$ be a complex agent. We say the agent $X$ is fully specified, written $\Delta X$, iff $\forall X' \in \mathcal{X}$ such that $X' \neq X : -(X' \triangleleft X)$.

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:

$$K(S[u]), B(\emptyset) :: cyt \Rightarrow K(S[p]) :: cyt + B(\emptyset) :: cyt.$$  

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,
- $\iota \in \{0, \ldots, n\}$ is an index determining the end of the left-hand side (LHS) of $\chi$,
- $\varphi \in \mathbb{N}^m$ is an index map from $\omega$ to $\chi$,
- $\psi \in ((\{-\} \cup \mathbb{N})^2)^n$ is an index map from LHS to RHS

where $n, m \in \mathbb{N}$, LHS $= (\chi_1, \ldots, \chi_i)$ is the left-hand side, and RHS $= (\chi_{i+1}, \ldots, \chi_n)$ 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 $\omega$ from sequence of complex agents $\chi$. The index map $\psi$ between the agents in $\omega$ 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 $\varphi$ serves for relating agents from $\omega$ back to the original sequence of complexes $\chi$. Finally, by index $\iota$ 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 $R$ to denote the universe of all possible rules.

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

\[
\chi = \begin{cases}
(K, ((S, u)), (B, \emptyset)), & (\text{cyt}), \\
((C, \emptyset), (D, i)), & (\text{cyt}), \\
((A, \emptyset)), & (\text{cyt}), \\
((K, ((S, p)), (B, \emptyset)), & (\text{cyt}), \\
((D, a), (A, \emptyset)), & (\text{cyt}) \\
((H, u)), & (\text{cyt})
\end{cases}
\]

\[
\omega = \begin{cases}
(K, ((S, u)), (B, \emptyset)), (C, \emptyset), & (\text{cyt}), \\
(D, i), (A, \emptyset), (K, ((S, p))), & \text{iff} \\
(B, \emptyset), (C, \emptyset), (D, a), (A, \emptyset), (H, u))
\end{cases}
\]

\[
\iota = 3 \\
\varphi = (2, 4, 5, 8, 10, 11) \\
\psi = [(1, 6); (2, 7); (3, 8); (4, 9); (5, 10); (-, 11)]
\]

written as:

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

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) $|\text{LHS}(R)| \neq |\text{RHS}(R)|$, 
(c) $\exists i \in \{1, i\} : \text{com}(\chi_i) \neq \text{com}(\chi_{i+1})$;

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

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

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 $\omega$ of the rule have the same name. Please note the conditions (i) and (ii) do not apply to those agents in $\omega$ 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 $\psi$ (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).
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 readable. It corresponds to the examples given while defining agents and rules above.

**Definition 2.27 (Grammar)**

<table>
<thead>
<tr>
<th>Atomic expression</th>
<th>Structure expression</th>
<th>Complex expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a ::= \eta(s) \mid \eta(c) )</td>
<td>( \tau ::= \eta(y) \mid \eta(\emptyset) )</td>
<td>( \Gamma ::= \beta_1 \ldots \beta_k :: c )</td>
</tr>
<tr>
<td>( \eta ::= n \in N_A )</td>
<td>( \gamma ::= a_1, \ldots, a_k )</td>
<td>( \beta_i ::= a \mid \tau )</td>
</tr>
<tr>
<td>( s ::= n \in N_S )</td>
<td>( \eta ::= n \in N_T )</td>
<td>( c ::= n \in N_c )</td>
</tr>
</tbody>
</table>

\[\left[\ldots\right]\]

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

where \( m, n \in \mathbb{N}_0 \land m > n \) 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 \( \left[ \ldots \right] \) as follows:

\[
F[\eta[c]] = (\eta, c) \in A \\
F[\eta[s]] = (\eta, s) \in A \\
F[\eta(\emptyset)] = (\eta, \emptyset) \in T \\
F[\eta(a_1, \ldots, a_k)] = (\eta, \{ F[a_1], \ldots, F[a_k] \}) \in T \\
F[\alpha_1 \ldots \alpha_k :: \epsilon] = (F[\alpha_1], \ldots, F[\alpha_k], \epsilon) \in X \\
F[\Gamma_1 + \Gamma_n \Rightarrow \Gamma_{n+1} + \Gamma_m] = (\chi, \omega, \psi, \psi) \in R \text{ such that:}
\]

- \( \chi = \{ F[\Gamma_1], \ldots, F[\Gamma_n], F[\Gamma_{n+1}], \ldots, F[\Gamma_m] \} \)
- \( \omega = \#_{i=1}^{|\chi|} \mu(\chi_i) \)
- \( \iota = n \)
- \( \varphi = (J_1, \ldots, J_m) \) where \( J_k = \sum_{i=1}^k \mu(\chi_i) \)

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 \( \chi \) and sequence of atomic and structure agents \( \omega \).

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 \( S \in \Omega^X \) such that \( A \) is the reference set of \( S \) and \( \forall X \in A: \Delta X \).

**Definition 2.30 (BCSL model)** We define BCSL model \( M \) as a quadruple \((R, \Sigma_A, \Sigma_T, S)\) where \( R \) is a set of rules, \( \Sigma_A \) is an atomic signature, \( \Sigma_T \) is a structure signature, and \( S \) is an initial solution.
A BCSL model is formed by a set of rules $R$, which define the behaviour of the model. The initial solution $S$ defines the state of the model in the beginning. Atomic signature $\Sigma_A$ defines allowed states for all atomic agents used in the rules. Finally, structure signature $\Sigma_T$ 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 $M$.

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

(i) $i = i'$ \land \varphi = \varphi' \land \psi = \psi'$,

(ii) $|x| = |x'| \land |\omega| = |\omega'|$,

(iii) $\forall i \in [1,|x|] : x'_i \vartriangleleft x_i$,

(iv) $\Omega(LHS(r)) = S$,

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

(a) $\omega_i \in A \Rightarrow \begin{cases} \omega'_i = \omega'_j & \text{if } \omega_i = \omega_j \\ \omega'_i \land \omega_j = \omega'_j & \text{if } \omega_i \neq \omega_j \end{cases}$,

(b) $\omega_i \in T \Rightarrow \gamma(\omega'_i) \lor \gamma(\omega_i) = \gamma(\omega'_i) \lor \gamma(\omega_i)$.

**Remark 2.32** Note the rule $r$ from the tuple $(R, S, r) \models$ 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 $\rightarrow \subseteq \Omega^X \times \mathbb{R} \times \Omega^X$ be the replacement relation such that a tuple $(S, R, S') \in \rightarrow$, written $S \rightarrow_R S'$, iff $\exists r \in \mathbb{R} \exists x \in S$ such that $R \models_r S \setminus \{x\} = \Omega(RHS(r))$.

Replacement relation defines how a solution is transformed according to a given rule. For a BCSL model $M$, rules yield a labelled transition system $LTS(M)$ between solutions containing an edge $S \rightarrow_R S'$. 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 $M$)**

(i) $\text{KaiC}(S|a, T|c), \text{KaiC}(S|c, T|e), \text{KaiC}(S|e, T|e) : \text{cyt} \Rightarrow \text{KaiC}(S|p, T|e), \text{KaiC}(S|c, T|e), \text{KaiC}(S|e, T|e) : \text{cyt}$

(ii) $\text{KaiC}(S|a, T|e), \text{KaiB}(\emptyset) : \text{cyt} \Rightarrow \text{KaiC}(S|p, T|e), \text{KaiB}(\emptyset) : \text{cyt}$

(iii) $\text{KaiC}(S|e, T|e) : \text{cyt} \Rightarrow \text{KaiC}(S|e, T|e) : \text{cyt} + \text{KaiC}(S|e, T|e) : \text{cyt} \Rightarrow \text{KaiC}(S|e, T|e) : \text{cyt} + \text{KaiC}(S|e, T|e) : \text{cyt}$

(iv) $\text{KaiC}(S|e, T|e), \text{KaiC}(S|e, T|e), \text{KaiC}(S|e, T|e) : \text{cyt} \Rightarrow \text{KaiC}(S|f|s, T|e) : \text{cyt} + \text{KaiC}(S|e, T|e) : \text{cyt} + \text{KaiC}(S|e, T|e) : \text{cyt}$

$$\Sigma_A = \{ S \rightarrow (a,p), T \rightarrow [a,i] \}$$

$$\Sigma_T = \{ \text{KaiC} \rightarrow [S,T], \text{KaiB} \rightarrow \emptyset \}$$

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 $e$ 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.
Running example 3.2 (The example model M)
(i) $\text{KaiC}(S[u]).\text{KaiC}(\emptyset).\text{KaiC}(\emptyset) \Rightarrow \text{KaiC}(S[p]).\text{KaiC}(\emptyset).\text{KaiC}(\emptyset) \Rightarrow \text{cyt}$
(ii) $\text{KaiC}(S[u]).\text{KaiB}(\emptyset) \Rightarrow \text{KaiC}(S[p]).\text{KaiB}(\emptyset) \Rightarrow \text{cyt}$
(iii) $\text{KaiC}(\emptyset) \Rightarrow \text{cyt} \Rightarrow \text{KaiC}(\emptyset) \Rightarrow \text{cyt} \Rightarrow \text{KaiC}(\emptyset) \Rightarrow \text{cyt}$
(iv) $\text{KaiC}(\emptyset).\text{KaiC}(\emptyset).\text{KaiC}(\emptyset) \Rightarrow \text{KaiC}(\emptyset) \Rightarrow \text{cyt} \Rightarrow \text{KaiC}(\emptyset) \Rightarrow \text{cyt} \Rightarrow \text{KaiC}(\emptyset) \Rightarrow \text{cyt}$

Additionally, this extension can go even further by omitting the $\emptyset$ part from structure agents completely (Running example 3.3). Since we have the structure signature $\Sigma_{cl}$ 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 M)
(i) $\text{KaiC}(S[u]).\text{KaiC}(\emptyset).\text{KaiC}(\emptyset) \Rightarrow \text{KaiC}(S[p]).\text{KaiC}(\emptyset).\text{KaiC}(\emptyset) \Rightarrow \text{cyt}$
(ii) $\text{KaiC}(S[u]).\text{KaiB}(\emptyset) \Rightarrow \text{KaiC}(S[p]).\text{KaiB}(\emptyset) \Rightarrow \text{cyt}$
(iii) $\text{KaiC} \Rightarrow \text{cyt} \Rightarrow \text{KaiC} \Rightarrow \text{cyt} \Rightarrow \text{KaiC} \Rightarrow \text{cyt}$
(iv) $\text{KaiC}.\text{KaiC}.\text{KaiC} \Rightarrow \text{cyt} \Rightarrow \text{KaiC} \Rightarrow \text{cyt} \Rightarrow \text{KaiC} \Rightarrow \text{cyt}$

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

3.2 Complex signature
We extend the model definition by complex signature $\Sigma_{X}$ (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 M)
Definition of complex signature $\Sigma_{X} = \{ \text{KaiC} \Rightarrow \text{KaiC}.\text{KaiC}.\text{KaiC} \Rightarrow \text{cyt}, \text{KaiBC} \Rightarrow \text{cyt} \Rightarrow \text{KaiC}.\text{KaiB} \Rightarrow \text{cyt} \}$
(i) $\text{KaiC}(S[u]).\text{KaiC}(\emptyset).\text{KaiC}(\emptyset) \Rightarrow \text{KaiC}(S[p]).\text{KaiC}(\emptyset).\text{KaiC}(\emptyset) \Rightarrow \text{cyt}$
(ii) $\text{KaiC}(S[u]).\text{KaiB}(\emptyset) \Rightarrow \text{KaiC}(S[p]).\text{KaiB}(\emptyset) \Rightarrow \text{cyt}$
(iii) $\text{KaiC} \Rightarrow \text{cyt} \Rightarrow \text{KaiC} \Rightarrow \text{cyt} \Rightarrow \text{KaiC} \Rightarrow \text{cyt}$
(iv) $\text{KaiC} \Rightarrow \text{cyt} \Rightarrow \text{KaiC} \Rightarrow \text{cyt} \Rightarrow \text{KaiC} \Rightarrow \text{cyt}$

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 $\rho : l \Rightarrow r$ can be written as two rule expressions $\rho_{1} : l \Rightarrow r \text{ and } \rho_{2} : r \Rightarrow l$ (Running example 3.5).

Running example 3.5 (The example model M)
(i) $\text{KaiC}(S[u]).\text{KaiC}(\emptyset).\text{KaiC}(\emptyset) \Rightarrow \text{KaiC}(S[p]).\text{KaiC}(\emptyset).\text{KaiC}(\emptyset) \Rightarrow \text{cyt}$
(ii) $\text{KaiC}(S[u]).\text{KaiB}(\emptyset) \Rightarrow \text{KaiC}(S[p]).\text{KaiB}(\emptyset) \Rightarrow \text{cyt}$
(iii) $\text{KaiC} \Rightarrow \text{cyt} \Rightarrow \text{KaiC} \Rightarrow \text{cyt} \Rightarrow \text{KaiC} \Rightarrow \text{cyt}$
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:
$$\beta_{1} \Rightarrow c \Rightarrow \beta_{2} \Rightarrow c \Rightarrow \ldots \Rightarrow \beta_{n} \Rightarrow c \Rightarrow \beta_{1}.\beta_{2}. \ldots \beta_{n} \Rightarrow c$$
we can reorder both sides such that we get non-crossing partition $P = B_{1}/B_{2}/\ldots/B_{k}$ with $k \leq n$ from its indices $[1, \ldots, n]$ such that: $\forall B \in P \forall \beta, \beta' \in B : \beta = \beta'$ and $\forall B, B' \in P \forall \beta \in B \forall \beta' \in B' : \beta \neq \beta'$ such that $B \neq B'$.

For the left-hand side $\beta_1 \leftrightarrow c + \beta_2 \leftrightarrow c + ... + \beta_n \leftrightarrow c$ of the reordered rule expression we can replace all rule expressions $[\beta_1, ..., \beta_n]$ which belong to the same non-crossing partition $B$ by notation $'k\beta'$, where $\beta$ is a representative from $\beta_1, ..., \beta_j$ (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 $M$)**

Definition of rule expressions:

(i) $KaiC(S[u]),KaiC.KaiC \leftrightarrow cyt \Rightarrow KaiC(S[p]),KaiC.KaiC \leftrightarrow cyt$

(ii) $KaiC(S[u]),KaiB \leftrightarrow cyt \Rightarrow KaiC(S[p]),KaiB \leftrightarrow cyt$

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

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 $a \leftrightarrow b$ 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) $A : T \Leftarrow \exists x \in \gamma(T)$ such that $A \Leftrightarrow A'$,

(ii) $A : X \Leftarrow \exists x \in \mu(X)$ such that $A \Leftrightarrow A'$,

(iii) $T : X \Leftarrow \exists x \in \mu(X)$ such that $T \Leftrightarrow T'$.

For each pair of agents $(\alpha, \beta)$ with allowed ‘$\leftrightarrow$’ operator between them, we can construct just one agent $\beta'$ without the operator by taking the most left agent $\alpha'$ from full (resp. partial) composition of the agent $\beta$ such that it is compatible with the agent $\alpha$. Then, agent $\alpha'$ is merged with agent $\alpha$ and agent $\beta'$ is constructed.

**Running example 3.8 (The example model $M$)**

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

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 $M$)**

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

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

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 \(x_1 < x_2\).

**Notation 4.2** The universe of complex agents \(X\) with partial order \(\leq\) is a partially ordered set \(X\leq\).

The compatibility operator defines a partial order on \(A\), \(T\), and \(X\) sets. For our purposes, only partially ordered set \(X\leq\) 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 \(X \subseteq X\) is a compatible set if:

(i) \(\forall X_1, X_2 \in X \exists X' \in X : X_1 < X' \land X_2 < X'\),

(ii) and for each finite set \(X' \subseteq X\) such that \(X \cap X' = \emptyset\) holds: \(\forall X \in X \forall X' \in X' : \neg(X < X' \lor X' < X)\).

**Remark 4.4** The compatible set \(X\) inherits partial order of \(X\leq\) since it is its subset.

A compatible set \(X\) 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.

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

**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 \(X\). 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 \(X \subseteq X\) such that \(X \in X\).

**Proof.** Let us assume a complex agent \(X\) belongs to two compatible sets, namely \(X \in X_1, X_2\). From Definition 4.3 condition (i) follows that there exists a \(X_1 \in X_1\) such that \(X < X_1\).

Next, the condition (ii) claims that no complex agent from \(X_1\) and no complex agent from \(X_2\) can be compatible. Namely, \(X_1 \in X_1\) cannot be compatible with \(X \in X_2\). However, \(X\) and \(X_1\) are compatible \((X < X_1)\). It follows \(X \not\in X_2\), 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 \(X \subseteq X\) be a compatible set and \(X \in X\) a complex agent. A set \(\bar{X} \subseteq X\) is called compatible subset of \(X\) w.r.t. \(X\) if the following conditions hold:

![Fig. 2. An example of a compatible set \(X\). 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.](image-url)
(i) $\forall X' \in \overline{X}: X' \not\prec X \land \Delta X'$,
(ii) $\nexists X'' \in X: X'' \not\prec X \land \Delta X''$.

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 $M_1 = (\mathcal{R} \cup \{R\}, \Sigma_M, \Sigma_T, S)$ and $M_2 = (\mathcal{R}, \Sigma_M, \Sigma_T, S)$ be BCSL models where $R$ is a rule such that $R \notin \mathcal{R}$. The rule $R$ is redundant if $\text{LTS}(M_1) = \text{LTS}(M_2)$.

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

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 $X$ since $X_k \leftarrow X'_k$, which follows from the condition of Theorem. We can create subsets $\overline{X}$, $\overline{X'} \subseteq X$ for both complex agents respectively (Definition 4.7). Since the agents are compatible ($X_k \leftarrow X'_k$), the compatible subset $\overline{X}$ w.r.t. agent $X_k$ is subset of the compatible subset $\overline{X'}$ w.r.t. agent $X'_k$ ($\overline{X} \subseteq \overline{X'}$).

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

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:

\[
\begin{align*}
K(S[u], T[a]), K(S[u], T[a]) :: \text{cell} & \Rightarrow K(S[p], T[a]), K(S[u], T[a]) :: \text{cell}, \\
K(S[u], T[a]), K(S[u], T[i]) :: \text{cell} & \Rightarrow K(S[p], T[a]), K(S[u], T[i]) :: \text{cell}, \\
K(S[u], T[a]), K(S[p], T[a]) :: \text{cell} & \Rightarrow K(S[p], T[a]), K(S[u], T[a]) :: \text{cell}, \\
K(S[u], T[i]), K(S[u], T[a]) :: \text{cell} & \Rightarrow K(S[p], T[i]), K(S[u], T[a]) :: \text{cell}, \\
K(S[u], T[i]), K(S[u], T[i]) :: \text{cell} & \Rightarrow K(S[p], T[i]), K(S[u], T[i]) :: \text{cell}, \\
K(S[u], T[i]), K(S[p], T[a]) :: \text{cell} & \Rightarrow K(S[p], T[i]), K(S[p], T[a]) :: \text{cell}, \\
K(S[u], T[i]), K(S[p], T[i]) :: \text{cell} & \Rightarrow K(S[p], T[i]), K(S[p], T[i]) :: \text{cell},
\end{align*}
\]

while the rule (ii) produces set of four reactions:
K(S[u], T[i]).K(S[u], T[a]) :: cell ⇒ K(S[p], T[i]).K(S[u], T[a]) :: cell,
K(S[u], T[i]).K(S[u], T[i]) :: cell ⇒ K(S[p], T[i]).K(S[u], T[i]) :: cell,
K(S[u], T[i]).K(S[p], T[a]) :: cell ⇒ K(S[p], T[i]).K(S[p], T[a]) :: cell,
K(S[u], T[i]).K(S[p], T[i]) :: cell ⇒ K(S[p], T[i]).K(S[p], T[i]) :: cell

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

Definition 4.12 (Reduced model) Let \( M = (R, \Sigma_R, \Sigma_T, S) \) be an initial BCSL model. We define reduced model \( \tilde{M} = (\tilde{R}, \Sigma_R, \Sigma_T, \tilde{S}) \) such that the following conditions hold:

(i) for every rule \( R \in \tilde{R} \), \( \theta(R) \in \tilde{R} \) and every rule in the reduced model is the image by \( \theta \) of a rule of the initial model;
(ii) for every complex agent \( X \in \tilde{S} \), \( \text{sup}(\chi) \in I \) where \( \chi \) is a compatible set such that \( X \in \chi \) and every complex agent in the reduced model is the image by \( \text{sup}(\chi) \) of a complex agent of the initial model.

Reduced model \( \tilde{M} \) 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 \( \chi \). 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.

\[
E[a]::\text{cyt} + S[u]::\text{cyt} \Rightarrow E[a].S[u]::\text{cyt} \\
E[a].S[u]::\text{cyt} \Rightarrow E[i].S[p]::\text{cyt} \\
E::\text{cyt} + S::\text{cyt} \Rightarrow E.S::\text{cyt} \\
E.S::\text{cyt} \Rightarrow E.S::\text{cyt}
\]

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 \( M \) 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 \( s_1 \triangleleft s_2 \), if there exists a bijective function \( f : s_1 \rightarrow s_2 \) such that \( \forall X \in s_1 : \text{sup}(\chi) = f(X) \) where \( X \subseteq X \) is a compatible set w.r.t. \( X \).

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

A reduced model \( \tilde{M} \) is actually an over-approximation of a BCSL model \( M \) 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 \( M \).

Theorem 4.15 Let \( X \) be a complex agent, \( \chi \) be a compatible set w.r.t. \( X \), \( M \) be a given BCSL model, and \( \tilde{M} \) be an appropriate reduced model of model \( M \). If supremum \( \text{sup}(\chi) \) is non-reachable in \( \text{LTS}(M) \), then agent \( X \) is also non-reachable in the \( \text{LTS}(\tilde{M}) \).
**Proof.** Let us assume a complex agent \( \text{sup}(X) \) is non-reachable in \( \text{LTS} (\tilde{M}) \), but \( X \in \mathcal{X} \) is reachable in \( \text{LTS}(M) \). Generally, there is a path formed from rules in the \( \text{LTS}(M) \) such that we transform complex agents from initial agents to desired complex agent \( X \). When we move to context of \( \text{LTS}(\tilde{M}) \), there is no such path for \( \text{sup}(X) \).

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 \( \text{LTS}(\tilde{M}) \) and the complex agent \( \text{sup}(X) \) is reachable, which is a contradiction. \( \square \)

When we are checking whether an agent is reachable in \( \text{LTS}(M) \) for given model \( M \), we might first check whether the respective abstract agent (the supremum) is reachable in \( \text{LTS}(\tilde{M}) \) of the reduced model \( \tilde{M} \). If this holds then we are still not certain about reachability of the agent in its initial form. This has to be checked in \( \text{LTS}(M) \). However, Theorem 4.15 states that agent which is not reachable in \( \text{LTS}(M) \) is also not reachable in \( \text{LTS}(\tilde{M}) \). 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 \( M \). 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 \( M \) 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: \( \forall R \in \mathcal{R} \forall i \in \text{RHS}(R) : \neg (\chi_i < X) \), where \( R = (\chi, \omega, \iota, \phi, \psi) \).

**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 \( X_2 \triangleleft X \) from a complex agent \( X_1 \) applying a rule \( R \).

\[
\begin{array}{c}
\text{init} \quad \text{X}_1 \quad \text{rule R} \quad \text{X}_2 \quad \text{reach}
\end{array}
\]

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. \( \square \)

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 unphosphorylated to phosphorylated) 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 \( \text{FRS[Thr[\{u\}], Tyr[\{u\}]], FGF[Thr[\{u\}]], R:FGF[Thr[\{u\}]], R:cyt} \) 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
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) \( FRS(Thr[u]) + FGF(Thr[p]).R.FGF(Thr[p]).R \Rightarrow \)
\( \Rightarrow FRS(Thr[u]).FGF(Thr[p]).R.FGF(Thr[p]).R \)
(v) \( FRS(Thr[u]).FGF.R.FGF.R \Rightarrow \)
\( \Rightarrow FRS(Thr[p]).FGF.R.FGF.R \)

Initial conditions: \( 2 R \)
\( 1 FRS(Thr[u], Tyr[u]) \)

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 \( Raf.ERK.cytcyt \) 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.

\[
\begin{align*}
FGF & + R \Leftrightarrow FGF.R & GAP & + Ras \Leftrightarrow GAP.Ras \\
FGF.R & + FGF.R \Leftrightarrow FGF.R.FGF.R & Ras & + Raf \Leftrightarrow Ras.Raf \\
FGF.R.FGF.R & + FRS \Leftrightarrow FGF.R.FGF.R.FRS & PP & + Raf \Leftrightarrow PP.Raf \\
FRS & + SHP \Leftrightarrow FRS.SHP & Raf & + MEK \Leftrightarrow Raf.MEK \\
GS & + GPP \Leftrightarrow GS.GPP & XPP & + MEK \Leftrightarrow XPP.MEK \\
GS & + ERK \Leftrightarrow GS.ERK & MEK & + ERK \Leftrightarrow MEK.ERK \\
FRS & + GS \Leftrightarrow FRS.GS & MKP & + ERK \Leftrightarrow MKP.ERK \\
FRS.GS & + Ras \Leftrightarrow FRS.GS.Ras & ERK & + FRS \Leftrightarrow ERK.FRS
\end{align*}
\]

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


A Model Yamada et al. 2004

\[
\begin{align*}
\text{FGF} : \text{cyt} + R : \text{cyt} & \iff \text{FGF}.R : \text{cyt} \\
2 \text{FGF}.R : \text{cyt} & \iff \text{FGF}.R.FGF.R : \text{cyt} \\
\text{FGF}(\text{Thr}[^u]),R.RGF.R : \text{cyt} & \iff \text{FGF}(\text{Thr}[^p]).R.FGF.R : \text{cyt} \\
\text{FRS}(\text{Thr}[^u]).R.GFG.R : \text{cyt} & \iff \text{FRS}(\text{Thr}[^p]).R.FGF.R : \text{cyt} \\
\text{FRS}(\text{Thr}[^p]).R.GFG.R : \text{cyt} & \iff \text{FRS}(\text{Thr}[^p]).R.FGF.R : \text{cyt} \\
\text{HSP} : \text{cyt} + \text{FRS}(\text{Thr}[^p]) & \iff \text{SHP}.FRS(\text{Thr}[^p]) : \text{cyt} \\
\text{FRS}(\text{Thr}[^p]).SHP : \text{cyt} & \iff \text{FRS}(\text{Thr}[^u]).SHP : \text{cyt} \\
\text{FRS}(\text{Thr}[^u]).SHP : \text{cyt} & \iff \text{FRS}(\text{Thr}[^u]).R.FGF.R : \text{cyt} \\
\text{GPP} : \text{cyt} + \text{GS}(\text{Thr}[^p]) & \iff \text{GPP}.GS(\text{Thr}[^p]) : \text{cyt} \\
\text{GS}(\text{Thr}[^u]).GPP : \text{cyt} & \iff \text{GS}(\text{Thr}[^u]).GPP : \text{cyt} \\
\text{GS}(\text{Thr}[^u]).GPP : \text{cyt} & \iff \text{GS}(\text{Thr}[^u]).GPP : \text{cyt} \\
\text{ERK}(\text{Tyr}[^p], \text{Thr}[^p]) : \text{cyt} + \text{GS}(\text{Thr}[^u]) & \iff \text{ERK}(\text{Tyr}[^p], \text{Thr}[^p]).GS(\text{Thr}[^u]) : \text{cyt} \\
\text{GS}(\text{Thr}[^u]).\text{ERK} : \text{cyt} & \iff \text{GS}(\text{Thr}[^p]).\text{R.GPP} : \text{cyt} \\
\text{GS}(\text{Thr}[^u]).\text{R.GPP} : \text{cyt} & \iff \text{GS}(\text{Thr}[^u]).\text{R.GPP} : \text{cyt} \\
\text{FRS}(\text{Thr}[^p], \text{Tyr}[^u]) : \text{cyt} + \text{GS}(\text{Thr}[^u]) & \iff \text{FRS}(\text{Thr}[^p], \text{Tyr}[^u]).GS(\text{Thr}[^u]) : \text{cyt} \\
\text{Ras}(\text{Thr}[^u]).\text{FRS}.GS : \text{cyt} & \iff \text{Ras}(\text{Thr}[^p]).\text{FRS}.GS : \text{cyt} \\
\text{Ras}(\text{Thr}[^p]).\text{FRS}.GS : \text{cyt} & \iff \text{Ras}(\text{Thr}[^p]).\text{FRS}.GS : \text{cyt} \\
\text{GAP} : \text{cyt} + \text{Ras}(\text{Thr}[^p]) & \iff \text{GAP}.\text{Ras}(\text{Thr}[^p]) : \text{cyt} \\
\text{Ras}(\text{Thr}[^p]).\text{GAP} : \text{cyt} & \iff \text{Ras}(\text{Thr}[^p]).\text{GAP} : \text{cyt} \\
\text{Ras}(\text{Thr}[^u]).\text{GAP} : \text{cyt} & \iff \text{Ras}(\text{Thr}[^u]).\text{GAP} : \text{cyt} \\
\text{Ras}(\text{Thr}[^u]).\text{GAP} : \text{cyt} & \iff \text{Ras}(\text{Thr}[^u]).\text{GAP} : \text{cyt} \\
\text{Ras}(\text{Thr}[^u]).\text{GAP} : \text{cyt} & \iff \text{Ras}(\text{Thr}[^u]).\text{GAP} : \text{cyt} \\
\text{Raf}(\text{Thr}[^p]) : \text{cyt} + \text{Raf}(\text{Thr}[^u]) & \iff \text{Raf}(\text{Thr}[^p]).\text{Raf}(\text{Thr}[^u]) : \text{cyt} \\
\text{Raf}(\text{Thr}[^u]).\text{Raf} : \text{cyt} & \iff \text{Raf}(\text{Thr}[^p]).\text{Raf} : \text{cyt} \\
\text{PP} : \text{cyt} + \text{Raf}(\text{Thr}[^p]) & \iff \text{PP}.\text{Raf}(\text{Thr}[^p]) : \text{cyt} \\
\text{Raf}(\text{Thr}[^p]).\text{PP} : \text{cyt} & \iff \text{Raf}(\text{Thr}[^u]).\text{PP} : \text{cyt} \\
\text{Raf}(\text{Thr}[^u]).\text{PP} : \text{cyt} & \iff \text{Raf}(\text{Thr}[^u]).\text{PP} : \text{cyt} \\
\text{Raf}(\text{Thr}[^p]) : \text{cyt} + \text{MEK}(\text{Ser}212[^u]) & \iff \text{Raf}(\text{Thr}[^p]).\text{MEK}(\text{Ser}212[^u]) : \text{cyt} \\
\text{MEK}(\text{Ser}212[^u]).\text{Raf} : \text{cyt} & \iff \text{MEK}(\text{Ser}212[^p]).\text{Raf} : \text{cyt} \\
\text{MEK}(\text{Ser}212[^p]).\text{Raf} : \text{cyt} & \iff \text{MEK}(\text{Ser}212[^p]).\text{Raf} : \text{cyt}
\end{align*}
\]
\[
\begin{align*}
Raf(Thr[p]) &:: cyt + MEK(Ser298[u]):: cyt \quad \Rightarrow \quad Raf(Thr[p]).MEK(Ser298[u]):: cyt \\
MEK(Ser298[u]).Raf &:: cyt \quad \Rightarrow \quad MEK(Ser298[p]).Raf :: cyt \\
MEK(Ser298[p]).Raf &:: cyt \quad \Rightarrow \quad MEK(Ser298[p]):: cyt + Raf :: cyt \\
XPP &:: cyt + MEK(Ser212[p]):: cyt \quad \Rightarrow \quad XPP.MEK(Ser212[p]):: cyt \\
MEK(Ser212[p]).XPP &:: cyt \quad \Rightarrow \quad MEK(Ser212[u]).XPP :: cyt \\
MEK(Ser212[u]).XPP &:: cyt \quad \Rightarrow \quad MEK(Ser212[u]):: cyt + XPP :: cyt \\
XPP &:: cyt + MEK(Ser298[p]):: cyt \quad \Rightarrow \quad XPP.MEK(Ser298[p]):: cyt \\
MEK(Ser298[p]).XPP &:: cyt \quad \Rightarrow \quad MEK(Ser298[u]).XPP :: cyt \\
MEK(Ser298[u]).XPP &:: cyt \quad \Rightarrow \quad MEK(Ser298[u]):: cyt + XPP :: cyt \\
\end{align*}
\]

\[
\begin{align*}
ERK(Thr[u]).MEK &:: cyt \quad \Rightarrow \quad ERK(Thr[p]).MEK :: cyt \\
ERK(Thr[p]).MEK &:: cyt \quad \Rightarrow \quad ERK(Thr[p]):: cyt + MEK :: cyt \\
ERK(Tyr[u]).MEK &:: cyt \quad \Rightarrow \quad ERK(Tyr[p]).MEK :: cyt \\
ERK(Tyr[p]).MEK &:: cyt \quad \Rightarrow \quad ERK(Tyr[p]):: cyt + MEK :: cyt \\
MKP &:: cyt + ERK(Thr[p]):: cyt \quad \Rightarrow \quad MKP.ERK(Thr[p]):: cyt \\
MKP &:: cyt + ERK(Tyr[p]):: cyt \quad \Rightarrow \quad MKP.ERK(Tyr[p]):: cyt \\
MEK(Ser212[p], Ser298[p]):: cyt \quad \Rightarrow \quad MEK(Ser212[p], Ser298[p]):: cyt \\
\end{align*}
\]

\[
\begin{align*}
Ras(Thr[u]) &:: cyt \quad \Rightarrow \quad Ras(Thr[u]).FS(Thr[u]):: cyt \\
\end{align*}
\]

\[
\begin{align*}
FAS(Thr[u]) &:: cyt \quad \Rightarrow \quad FAS(Thr[p]).RGF(Thr[p]):: cyt \\
\end{align*}
\]

\[
\begin{align*}
ERK(Tyr[u]) &:: cyt \quad \Rightarrow \quad ERK(Tyr[u]).MEK(Ser212[p], Ser298[p]):: cyt \\
FAS(Tyr[u]) &:: cyt \quad \Rightarrow \quad FAS(Tyr[u]).ERK(Tyr[p], Thr[p]):: cyt \\
\end{align*}
\]