Masarykova univerzita
Fakulta informatiky

Formal Biochemical Space for Specification and Analysis of Biochemical Processes

Master thesis

Matej Troják

Brno, autumn 2017

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Matej Troják

Advisor: RNDr. David Šafránek, Ph.D.

Acknowledgement

I would especially like to thank my amazing family for the love, support, and constant encouragement I have gotten over the years. In particular, I would like to thank my parents and my brother.

I would like to thank my advisor, David Šafránek for guiding and supporting me over the years. You have set an example of excellence as a mentor and researcher.

I would like to thank all members of the Systems Biology Laboratory for all discussions, help, and feedback they have provided.

Many thanks to you all.

Abstract

In this thesis, we define a formal rule-based language suitable for description and analysis of biological systems as a formal part of Biochemical Space. Biochemical Space is a semi-formal notation for reaction networks of biological systems. It is being developed as a part of the Comprehensive Modelling Platform – a web-based platform for modelling and analysis of biological processes. It provides a concise mapping of mathematical models to their biological description established at a desired level of abstraction. It is enabled by combining state-of-the-art rule-based techniques with meta-data formats developed in well-known annotation databases.

Keywords

formal specification, biological systems, systems biology, rule-based language

Contents

Introduction

Modelling in computational systems biology often comprises studying a complex biochemical system involving a large number of interacting species. Conventional modelling approaches explicitly specify entire chemical reaction networks. Models have the form of lists of biochemical species and their reactions. The issue with this approach is that it is not always effective or even possible to enumerate all the interactions. Typically, a biological system contains a huge number of species with numerous interactions.

Rule-based modelling approaches describe biological interactions in terms of rules, avoiding the combinatorial explosion of underlying system. Rules are patterns which define how groups of species with some common properties behave in the given system. The rule-based model can be translated into a model such as Markov chains or differential equations, alternatively analysed in its original form. Rule-based modelling is especially effective in cases when the model contains a limited number of patterns. Typical representatives of such cases are biochemical models of living organisms.

In this thesis we focus on development of a novel rule-based language called Biochemical Space language. The motivation was to establish a rule-based *human-readable* language which is easy to read, write, and maintain by people even from other disciplines than computer science. It serves as the description format for Biochemical Space [24], a core part of Comprehensive Modelling Platform [25]. Since the language is used for the specification of biochemical species and interactions within the platform, its dedicated purpose is to be directly presentable to the users.

The goal of this thesis is to formally define the syntax and semantics of Biochemical Space language and provide techniques to analyse models written in the language.

In Chapter 1 we give an introduction to rule-based modelling, describe differences between the rule-based and traditional approaches, reveal key features of existing rule-based formalisms, and explain purpose of Comprehensive Modelling Platform and its sub-parts. In Chapter 2 we specify particular tasks of this thesis in the context of preliminary work. In Chapter 3 we define syntax and semantics

of Biochemical Space language and specify several syntactic extensions, which improve readability of the language. Chapter 4 provides brief description of traditional dynamic analysis techniques and introduces several new static analysis techniques. Chapter 5 shows usability of the language on several case studies and then demonstrate usability of proposed static analyses in practice. In the final Chapter 6 we describe development of BCSgen, a software tool to support the maintenance and analysis of models written in the language.

1 Background

In this chapter, we introduce all the relevant terms which will be used in the thesis. We give an introduction to the rule-based modelling and compare it to the usual reaction-based approach. Then we describe typical representatives of the rule-based languages and highlight their key features. We also introduce Comprehensive Modelling Platform with its core parts and provide motivation for Biochemical Space as a core module of the platform.

1.1 Rule-based modelling

Modelling complex systems in biology, such as individual cells, is necessary with respect to their complexity and size. The more complex the system is, the harder it is to describe it formally. The typical approach in modelling is using chemical reactions or ordinary differential equations [6]. The general problem with these approaches is that they require to enumerate all the interacting objects and interactions themselves. However, this number is often too high to be effectively processed. The rule-based modelling offers a solution for this problem. It is a natural extension of the reaction-based approach. Instead of operating with objects, we operate with *types*. The semantics of the model are defined via *rules* on the given types. The rule-based models are generally more concise.

There are multiple approaches to rule-based modelling, but all of them follow the principle outlined above.

We represent a rule-based model $\mathbb{M}$ as a set of rules and an *initial solution* of interacting objects. We understand the solution as a mixture of individual objects which are randomly distributed (Figure 1.1a). Therefore, we cannot assign them any order and we represent them as multisets. From biological perspective, this representation of solution is as close as is possible to the reality while preserving conciseness.

The rules are patterns which describe behaviour of groups of objects. A rule has the form of an abstract chemical reaction, where substrates and products take place. The difference is that a reaction only operates on particular objects, while in the rule groups of ob-

jects are interacting. Therefore, the reaction can be seen as a special case of the rule (Figure 1.2), where the type represents exactly one object.

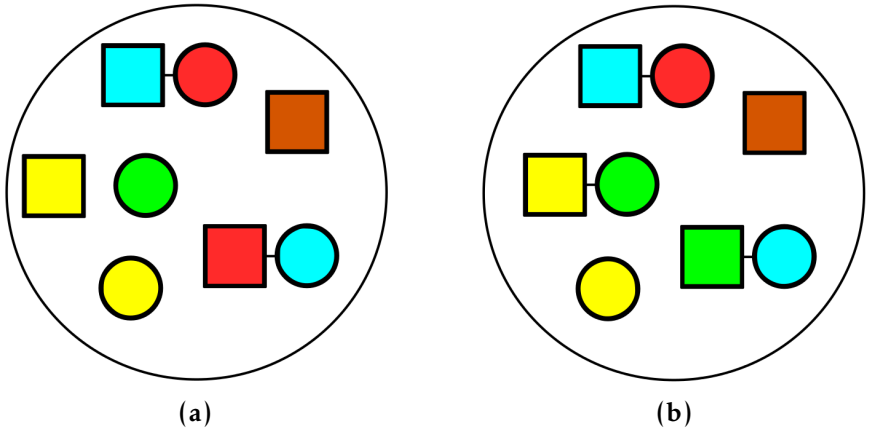


Figure 1.1: Examples of solutions. **(a)** An example of graphical representation of a solution. **(b)** Updated solution after the rules from Figure 1.2 were applied. The first rule (1) was applied on the yellow □ and the green ○ and produced the yellow-green complex □–○. Note there are more options how the rule could be mapped – each combination of free □ and ○. The second rule (2) was applied on the red-cyan complex □–○ where the colour of □ was changed from red to green. The third rule (3) couldn't be applied because there is no such complex with a yellow ○.

The rule is mapped on a solution. Then it can be applied and a new solution is produced. The mapping is not always successful (Figure 1.1, application of rule 3).

The mapping of a rule on a solution can be understood as a particular moment when the objects in the solution have just the conformation suitable for the rule and therefore the rule can be applied. Since the distribution of objects is random, we can assume a sequence of objects needed for rule (if there are such objects) is always available.

The mapping can also be seen as assigning particular objects from the solution to types on the left-hand side of the rule. The rule application then represents the change of the mapped objects to new objects according to the right-hand side of the rule (i.e., particular

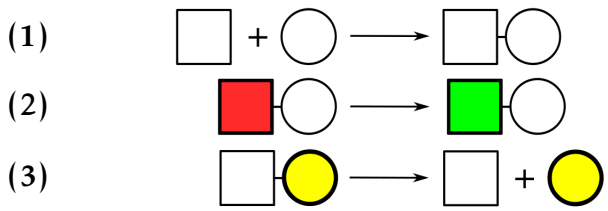


Figure 1.2: Examples of rules. Rule **(1)**: a □ and a ○ can form a complex □–○ regardless their colours. Rule **(2)**: a □ is allowed to change its colour from red to green only if it is in a complex with a ○ regardless its colour. Rule **(3)**: the rule can disassemble the complex only if the ○ is yellow.

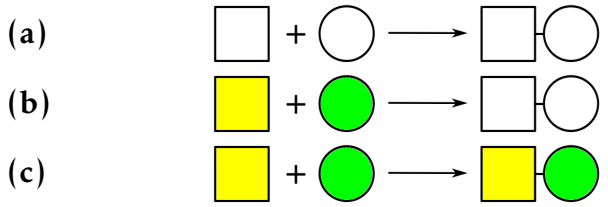


Figure 1.3: Example of a map-apply action. As a solution, we use the solution (a) from Figure 1.1. **(a)** Rule can be mapped on a □ and a ○ regardless their colours and form a complex □–○. **(b)** We randomly choose the yellow □ and the green ○ from our solution. The rule was mapped on chosen objects and they were assigned to the left-hand side of the rule. **(c)** The rule was applied and new object (the yellow-green complex □–○) was created. We obtained the reaction describing the particular action which has just happened.

objects are assigned to the right-hand side). As a by-product, we obtain an instance of the rule – a reaction (Figure 1.3).

Since this kind of graphical representation is not very suitable for machine processing and analysis, we define a formal language which is built on principles described in this chapter.

1.2 Rule-based versus reaction-based modelling

A rule is a generalised reaction (Figure 1.4). This is how the difference can be characterised. From the application point of view, the

reaction can be applied on a solution only in a particular way whilst the rule can be applied in multiple ways. It allows to write models where many similar processes might occur with the rules, by defining a pattern in which the action can happen. Since there can be many features we need to express, there are several languages focusing on different level of details.

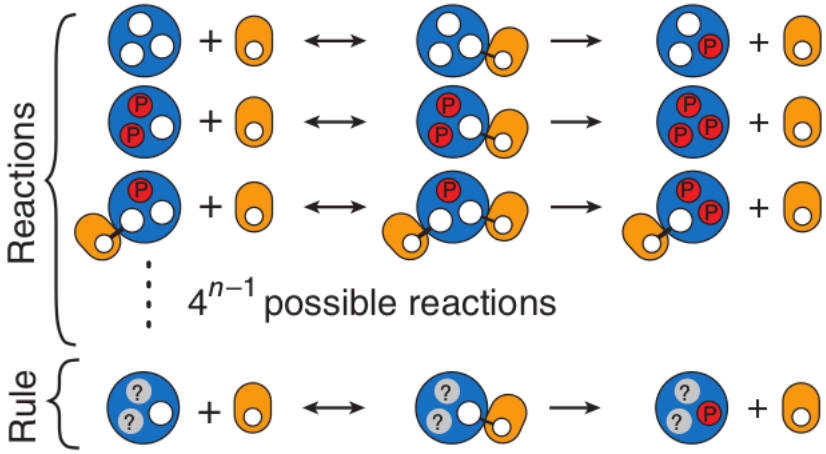


Figure 1.4: Graphical comparison of the rule-based and the reaction-based approaches [32]. There can be seen combinatorial explosion in the number of possible particular interactions (the reactions), which can be depicted in a few simple rules.

Analysing the rules directly is not the best choice. The main reason are *side effects*, which might happen during the rule application. These side effects differ from language to language. There is no such thing for the reactions because the entire context is enumerated explicitly. We cannot know how exactly the rule will be used before it is applied.

The rule and the reaction differ in the level of abstraction. In this context we mean how stringent it is, not what sort of features a particular language uses. This is the universal attribute for all rule-based languages – the rule is always more abstract than its instance – the reaction. The analysis of such abstract objects statically is a challenge. The reason is that most of the structural information is encoded in the rules implicitly. Explicit data are more suitable for static analysis.

1.3 Rule-based languages

There are several rule-based languages dedicated for 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.

1.3.1 Kappa language

The Kappa language [8] was primary 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. Same as all the other rule-based languages, when some context is not relevant, it is omitted. Example of a rule is in Figure 1.5.

(a) A(bsa~{ u, p }) ; B(bsb~{ a, i })
(b) A(bsa), B(bsb~a) → A(bsa!1), B(bsb~a!1)

Figure 1.5: 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) The rule itself. It defines creation of bond on the sites of agents A and B such that site of agent B must be in an active state. Note the context with no importance for the interaction is omitted.

The general problem present in the Kappa language is too detailed description. For biological systems as defined in Chapter 1.1, it might be often difficult to deal with all the bonds, especially in cases when we do not need such details.

1.3.2 BioNetGen Language

The BioNetGen Language (BNGL) [11] is very similar to the Kappa language and also has similar disadvantages. There are, however, a few general differences:

1. it is allowed to define multiple binding sites with same name for an agent,
2. one binding site is allowed to have multiple bonds,
3. in the rules, BNGL uses '+' and '.' for expressing reaction complex and complex of agents respectively.

Example of a rule is given in Figure 1.6.

(a) A(bsa~u~p)
    B(bsb~a~i)
(b) A(bsa) + B(bsb~a) → A(bsa!1).B(bsb~a!1)

Figure 1.6: 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) The rule itself. 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).

1.3.3 Chromar

The Chromar language [18] allows to define attributes for agents and range them over pre-defined domains. The qualitative semantics are 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). The language is very useful when creating a model where we need to create new distinct objects and control 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 the stochastic semantics of this language offers. Compared to the other rule-based languages, it is capable of specifying the rates for individual reactions inherited from the rule (Figure 1.7). It is allowed by variable value bindings and type-determination between 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 1.7: Example of a Chromar rule. Arithmetic operations can be used in order to change 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 is not the best choice. All the biologically relevant terms such as states have to be encoded in natural numbers.

1.3.4 PySB

The PySB language [26] works 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, what significantly increases the expression power of PySB. On the other hand, it might be hard to follow abstractions which are possible this way and models themselves are hard to analyse. Moreover, the core of the language is defined by translating to BNGL (Section 1.3.2). An example of a rule is given in Figure 1.8.

```
# Declare the monomers
Monomer('L', ['s'])
Monomer('R', ['s'])

# Declare the binding rule
Rule(L(s=None) +R(s=None) <>L(s=1) % R(s=1), kf, kr)
```

Figure 1.8: Example of a PySB rule. The rule describes creation of a bond between agents `R` and `L`.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
  <model>
    <listOfCompartments>
      <compartment id="c" constant="true" multi:isType="true" />
      <compartment id="cc" constant="true" multi:isType="true">
        <multi:listOfCompartmentReferences>
          <multi:compartmentReference multi:id="cr1" multi:compartment="c" />
          <multi:compartmentReference multi:id="cr2" multi:compartment="c" />
        </multi:listOfCompartmentReferences>
      </compartment>
    </listOfCompartments>
    <multi:listOfSpeciesTypes>
      <multi:bindingSiteSpeciesType multi:id="stA" multi:compartment="c" />
      <multi:speciesType multi:id="stAA" multi:compartment="cc">
        <multi:listOfSpeciesTypeInstances>
          <multi:speciesTypeInstance multi:id="stiA1" multi:speciesType="stA"
            multi:compartmentReference="cr1" />
          <multi:speciesTypeInstance multi:id="stiA2" multi:speciesType="stA"
            multi:compartmentReference="cr2" />
        </multi:listOfSpeciesTypeInstances>
        <multi:listOfInSpeciesTypeBonds>
          <multi:inSpeciesTypeBond multi:bindingSite1="stiA1" multi:bindingSite2="stiA2" />
        </multi:listOfInSpeciesTypeBonds>
      </multi:speciesType>
    </multi:listOfSpeciesTypes>
    <listOfSpecies>
      <species id="spA" multi:speciesType="stA" compartment="c" ... />
      <species id="spAA" multi:speciesType="stAA" compartment="cc" ... />
    </listOfSpecies>
    <listOfReactions>
      <reaction id="reaction" ...>
        <listOfReactants>
          <speciesReference id="r1" species="spA" multi:compartmentReference="cr1" ... />
          <speciesReference id="r2" species="spA" multi:compartmentReference="cr2" ... />
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="spAA" ... />
        </listOfProducts>
        ...
      </reaction>
      ...
    </listOfReactions>
  </model>
</sbml>
```

Figure 1.9: 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.

1.3.5 SBML-multi package

Systems Biology Markup Language (SBML) [19] is a standard established for systems biology. A SBML-multi package [36] 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 exchange and storage of the models but less for analysis and direct representation to the users (it is an XML format). It serves as an intermediate language. For an example see Figure 1.9.

1.4 Comprehensive Modelling Platform

Comprehensive Modelling Platform (CMP) [25] is an online platform providing tools for public sharing, annotation, analysis, and visualisation of dynamical models and *wet-lab* experiments related to domain-specific systems. The platform is unique in integrating abstract mathematical models with a precise consortium-agreed biochemical description provided in a rule-based formalism. The general aim is to stimulate collaboration between experimental and computational systems biologists to achieve better understanding of the domain-specific system.

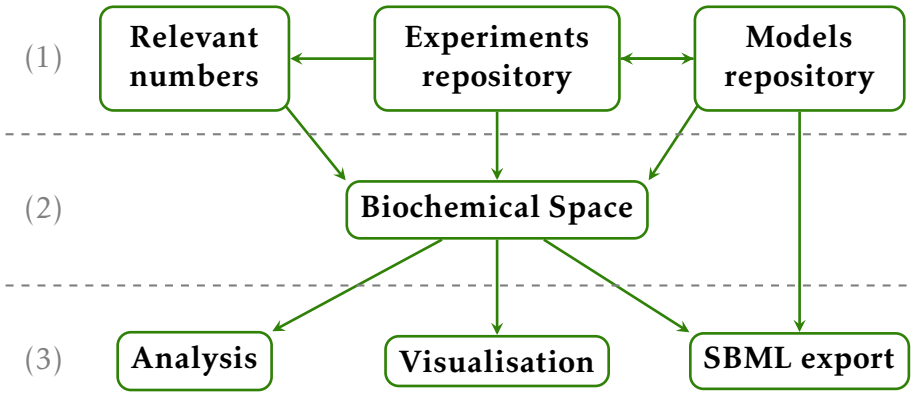


Figure 1.10: An overview of Comprehensive Modelling Platform. It is composed of three layers: (1) input of the platform with all the relevant data; (2) Biochemical Space as an intermediate format for the data; (3) results of analysis, visualisation, and annotation.

1.4.1 Biochemical Space

The platform consists of several dedicated modules (Figure 1.10), all connected to a central module – *Biochemical Space* (BCS) [24] – the backbone of the platform. BCS provides formal description and annotation of the biological system and it is based on the hierarchy of selected biological processes. It is accompanied with schemas representing relevant biological processes in the context of the modelled system. For each process, there are presented the relevant models, chemical entities, and rules. Presentation of every process includes detailed information and links to the relevant internal and external sources.

BCS provides a well-described biological background for mathematical models of processes taking place in a specific organism. Complete BCS provides a connection between existing ontologies and mathematical models. BCS has a *human-readable* format which can be easily edited in a dedicated editor and visualised within the platform. BCS is represented by two parts – a set of *entities* and a set of *rules*.

When building the BCS, emphasis is put on well-defined and complete annotations. Links to the 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 one part of the already mentioned connection between ontologies and models. At this moment, links to KEGG [23], ChEBI [9], CyanoBase [29], and other databases are supported (example of usage is in Figure 1.11). A single entity or a rule can have multiple links to several external databases. An example is the presence of a particular entity in ChEBI as well as in KEGG.

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

Figure 1.11: Description, links, and notes information for an entity.

The fact that most attributes in entity and rule definitions are tightly coupled with information from linked ontologies is the rea-

son why we have started with describing annotation attributes. In the first place, one of these attributes is *name*, which is taken from ontologies or follows the standard naming conventions. 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.

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

Figure 1.12: Complete information given for an atomic entity.

An entity in our interpretation is a bounded space (a so-called *compartment*) or a structural part of a specific organism (for an example see Figure 1.12). BCS covers a hierarchy of *entities* (or *agents*) ranging from small molecules through composite structures to large complex molecules. 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. They enable definition of the set of allowed states, in which an *atomic* entity can occur.

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

An atomic entity can be a part of a structurally more complex entity. We consider two kinds of composite entities: *structure* (Figure 1.13) and *complex* entity. Structure entity represents partially specified composite species, for example a photosystem complex par-

tially specified with prosthetic groups of interest. Complex entity represents fully specified composite species, for example, a homodimer of KaiC protein.

ENTITY ID:	KaiC
STATES:	
LOCATIONS:	cyt
COMPOSITION:	S | T
ENTITY NAME:	circadian clock protein kinase KaiC
CLASSIFICATION:	enzyme
DESCRIPTION:	Monomer component representing a core component of the circadian clock system.
LINKS:	uniprot::Q79PF4, cyanobase::Synpcc7942_1216
ORGANISM:	Synechococcus elongatus PCC 7942

Figure 1.13: Complete information given for a structure entity.

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 the 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 or between the 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. For an example, see Figure 1.14.

In some cases, emphasis on detailed description leads to a very complex BCS models. Abstraction of some processes is 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 attribute 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.

RULE ID:	FGFR2 phosph.
RULE EQUATION:	$Thr\{u\} :: FGF :: FGFR2 :: cyt \Leftrightarrow$ $\Leftrightarrow Thr\{p\} :: FGF :: FGFR2 :: cyt$
MODIFIER:	NDH1
RULE NAME:	FGFR2 threonine residue (de)phosphorylation
CLASSIFICATION:	phosphorylation, dephosphorylation
DESCRIPTION:	FGF enzyme is phosphorylated on threonine residue in FGFR2 complex.

Figure 1.14: A rule employing entity state change. A threonine (Thr) amino acid residue of a FGF protein inside a $FGFR2$ complex can change its state from unphosphorylated (u) to phosphorylated (p) (and vice versa).

Higher abstraction comes into account when several electrons play 'musical chairs' inside protein complexes. The issue is that parts of the processing protein complex can have different unstable states during a short period of time. When one tries to define all rules among these proteins the 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 – we treat a protein complex as a structure entity on which atomic entities change their states (not necessarily proteins) and we abstract out the background processes.

When describing particular process or object we can go almost infinitely deep into the details. Even if the depth is not infinite, it is deep enough to lead into a state when the description is larger than object itself. Imagine that you want to describe a molecule as the position of each atom. You might need more space to store such information than the molecule itself occupies.

When creating BCS for a system, we choose a level of detail which we do not exceed. This ensures the size of BCS will not exponentially increase with additional details. The problem can occur when we try

to map a model which uses more detailed description of the system than BCS does. However, we simply allow relating multiple model objects to BCS objects and vice versa. For example, multiple model reactions might be mapped on a single BCS rule denoting the fact that the process represented by the rule can be described in more details by the reactions. On the other hand, the model might neglect some details. Then for example, a reaction can be mapped on multiple rules with similar meaning than in the previous case.

Since CMP allows export of models in SBML format, we employ resolvable identifier links [22]. These links are included in generated SBML file and point to the appropriate rules and entities in BCS.

1.4.2 Model Repository

Model repository is a collection of implemented mathematical models describing particular parts of the biological processes. Each model is represented as a set of ordinary differential equations generated from the model reaction network. Models are integrated within BCS. In particular, each model component should be related to some BCS entity and each model reaction should be related to some BCS rule. The model is associated with some parameter value sets (*data sets*) that enable simulation in a particular biologically-relevant scenarios.

An implemented model then includes complete biological annotation of all components and reactions that is provided by a mapping to BCS. This can help to find connections and overlaps among models. Further, implemented model can be exported to SBML.

1.4.3 Experiments Repository

Experiments repository is a module for storage and presentation of time-series data from wet-lab experiments. Every experiment is well-grounded by precise description (device, medium, organism, etc.) and appropriate annotations. Experiments are structured – several time series data can be attached to a single experiment. Every time series targets a specific list of measured substances together with time stamps of the individual measurements. Time series can be visualised in a chart.

2 Problem formulation

Biochemical Space as described in Section 1.4.1 can grow to enormous size. To be able to detect inconsistency with automatic tools and enable further analysis of the space it is necessary to keep it on a formal level.

In our last paper [10], we have defined Biochemical Space Language (BCSL) such that syntax has been defined independently, but semantics has been defined via translating to the Kappa language [8]. We wanted to keep the language as concise as possible. The rule-based approach was the best choice, because it was capable to capture required details for biological systems while keeping the description concise. It was supposed to be *human-readable*, which in this context means easy to read, write, and maintain by humans even from other disciplines than computer science. However, there are many rule-based languages, and each of them is slightly different on syntactic and semantic level and also on the level of abstraction. After some research (Section 1.3), we have evaluated all the existing languages as not suitable for our purposes. The closest representation to the desired one was the Kappa language (Section 1.3.1). That is the reason why in the first version we have chosen translation to Kappa for definition of semantics.

However, this approach is not very beneficial since we are not fully using the expressive power of our language. The problem is that Kappa operates purely on expression levels. We want to promote our language to a higher level and have it operate on objects in multisets. By translating to Kappa we loose such abstraction.

In the previous version of language development, both definition and implementation were made via translation to Kappa. This means there was defined a translation function, which was able to convert BCSL expressions to Kappa expressions. The main advantage was that the conversion was quite straightforward and that way we could apply all types of analyses Kappa language offers.

The indirect approach had several disadvantages. Kappa has only one type of agent (no hierarchy). It follows all additional information encoded in different agent types are lost during the conversion. This particularly means atomic agents had to be considered as binding

sites and structure agents as Kappa agents. But it was not always suitable, especially when atomic agents had an independent role in a rule. Treatment of complexes in Kappa requires explicit bonds between individual agents. Therefore, when translating our multiset-based complex agent to a Kappa complex, we need to choose one of the many possible isomorphisms (Figure 2.1, the left-hand side of the figure corresponds to Kappa approach). Multiple default options are available, such as circular layout, linear layout etc. But it was not possible to express the fact that the order does not matter.

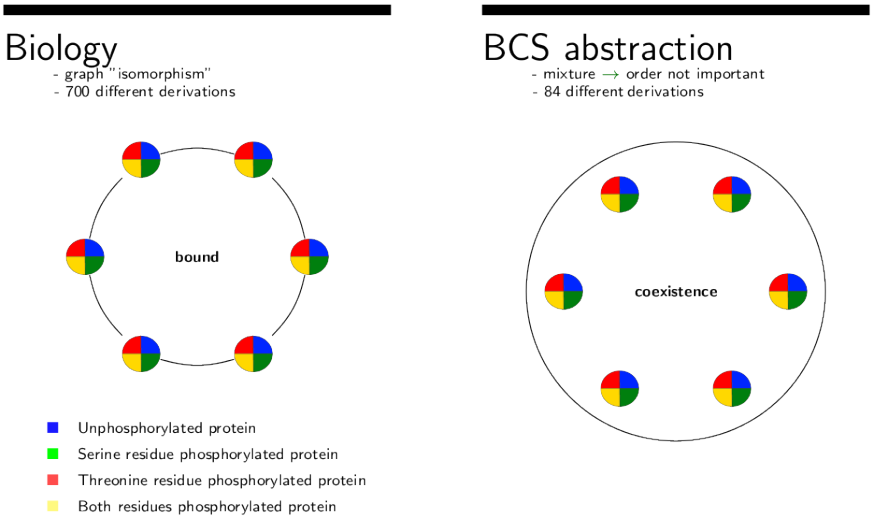


Figure 2.1: Comparison of a complex in biology (on the left) to abstraction used in BCS (on the right). In biology, all possible graph isomorphisms must be considered which results from combinatorial options how edges can be placed to create a connected graph [3] (additionally, we could consider the fact there might be multiple bonds between two entities). In BCS we neglect such details and see created structure only in the sense of coexistence (a multiset), which significantly reduces the number of possible derivations.

From the implementation point of view, execution of the models was delayed by the conversion to Kappa and then calling Kappa core to execute its semantics. Moreover, Kappa uses a network-free simulator, which is very efficient for small models, because the rules are executed directly. Each rule is being matched on current state and

then matched objects are replaced according to the rule. The problem arises when the model is more complex. The number of rules does not cause any significant difference, however the number of interacting agents does. Once there are many agents, the time of one *match & replace* step rises dramatically. Compared to indirect methods when at first the reaction network is enumerated, the initial step can be resources consuming, but then the rule application itself is fast regardless of the number of agents.

For the reasons mentioned above we decided to redefine the BCS language. Particularly, the syntax definition had to be refined in order to make the usability more clear. From the semantic point of view, we needed a straightforward definition, which would have a practical use in implementation. Experience from previous versions showed that using the direct rule-based semantics was not very efficient mechanisms. For that reason, we have chosen an indirect definition via translation to reactions – *practical* definition (Section 3). Practical in this case means that this approach can be efficiently used in implementation.

In Section 1.4.1, we have motivated the level of abstraction we need to use in BCS. To achieve that, we decided to neglect structural details, i.e. we do not employ binding approach. We consider the binding as a too low level of details. Instead, we improved the definition of interacting agents to higher level, when an agent might express some kind of structural description (Figure 2.1). This way we avoid complicated creation of complexes and reduce complicated syntax.

We determine two types of complexes. None of them is literally a complex, we rather see it as a coexistence. It allows us to express even other types of interaction. We also determine whether the expressed complex is complete, i.e. the enumerated subparts are all which form the complex. The other type of complex expresses the fact that subparts play an important role in the complex, but there are more parts which we do not model, and this fact is important for the semantics of the model.

3 Formal definition

In this chapter, we define the Biochemical Space Language formally. At first, we define all the required objects, then we define syntax of the language and semantics of the BCSL models.

The defined semantics is indirect, i.e. we do not give semantic sense to rules themselves, rather provide a procedure how to generate reactions from the rules and give semantics to them.

3.1 Formal preliminaries

Before we proceed, we provide some basic definitions and notations in order to build the formal definition for the language. Note we usually use the colon to denote the condition in a logic formula.

Definition 3.1.1 *Multiset*

We define a multiset *O as a pair $(A, \mathtt{m})$ where A is a set and $\mathtt{m} : A \to \mathbb{N}$ is a function from A to the set of natural numbers. The set A is called the* underlying set of elements. *For each* a *in A the* multiplicity *(that is, number of occurrences) of* a *is the number* $\mathtt{m}(\mathrm{a})$.

Notation 3.1.2 *Let $O = (A, m)$ be a multiset. By $x \in O$ we denote the fact that there exists $x \in A$ such that $m(x) \geq 1$.*

Definition 3.1.3 *Projection*

Projection $\pi_i : X^n \to X$ *for some $i \in \mathbb{N}$ is defined as follows:*

$$\pi_i(x_1, \ldots, x_n) = \begin{cases} x_i & \ldots \quad 1 \leq i \leq n \\ 0 & \ldots \quad otherwise \end{cases}$$

Definition 3.1.4 *Tuples concatenation*

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

$$X + Y = (x_1, \ldots, x_n, y_1, \ldots, y_m).$$

Definition 3.1.5 *Sum of concatenations*

Let $T = (T_1, T_2, \ldots, T_n)$ *be sequence of tuples.* Concatenation *of sequence of tuples* $+\!\!+_{i=1}^{n} T_i$ *is defined as:*

$$+\!\!+_{i=1}^{n} T_i = T_1 +\!\!+ T_2 +\!\!+ \ldots +\!\!+ T_n$$

Definition 3.1.6 *Non-crossing partition*

A partition $\mathcal{P} = B_1/B_2/\ldots/B_k$ *of* $S = [1, \ldots, n]$ *is* non-crossing *if whenever a quadruple of elements* $1 \leq a < b < c < d \leq n$ *satisfies* $a, c \in B_i$ *and* $b, d \in B_j$ *for some* $1 \leq i, j \leq k$*, then in fact* $i = j$*; thus, the blocks do not cross.*

Notation 3.1.7 *Let* $\vec{v} = (x_1, \ldots, x_n)$ *be a vector. By* $\sigma(\vec{v})$ *we denote a set of all possible permutations of length n of the vector* $\vec{v}$*.*

Definition 3.1.8 *Labelled transition system*

Labelled transition system *(LTS)* $\mathcal{L}$ *is a quadruple* (S, A, T, s_0) *where:*

- S *is a set of states,*
- A *is a set of labels,*
- $T \subseteq S \times A \times S$ *is a transition relation,*
- $s_0 \in S$ *is the initial state.*

Definition 3.1.9 *Path in LTS*

Let $\mathcal{L} = (S, A, T, s_0)$ *be an LTS. We define* path *as a finite sequence of states:*

$$\mathrm{s}_1 \mathrm{s}_2 \ldots \mathrm{s}_n$$

such that

$$\forall \mathrm{s}_i, \mathrm{s}_{i+1} : (\mathrm{s}_i, \mathrm{a}, \mathrm{s}_{i+1}) \in T \text{ for some } \mathrm{a} \in A.$$

Definition 3.1.10 *Petri Net*
We define Petri Net *as a quadruple* $\mathcal{N} = (P, T, F, m_0)$ *where:*

- P *is a finite non-empty set of* places,
- T *is a finite non-empty set of* transitions,
- $F \subset ((P \times T) \cup (T \times P))$ *is a set of* arcs,
- $m_0 : P \to \mathbb{N}$ *is an initial* marking.

3.2 Objects definition

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

For better readability, we provide examples of syntax for the most important object with their definitions. The formal definition of syntax and the relation to the objects will be given in subsequent sections.

3.2.1 Signature

Definition 3.2.1 *Signature*
Atomic signature $\Sigma_{\mathsf{A}} \subseteq \mathcal{N}_A \times 2^{\mathcal{N}_\delta}$ *associates atomic names with set of state names.*
Structure signature $\Sigma_{\mathsf{T}} \subseteq \mathcal{N}_T \times 2^{\mathcal{N}_A}$ *associates structure names with 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. The deeper meaning will be revealed after the next section of definitions.

Example 3.2.2 *Signature*

- An atomic signature $\left\{ (S, \{u, p\}), (Q, \{a, i\}) \right\}$
- A structure signature $\left\{ (KaiC, \{S, Q\}) \right\}$

3.2.2 Atomic agent

Definition 3.2.3 *Atomic agent*
We define an atomic agent A *as a pair* (η, δ) *where* $\eta \in \mathcal{N}_A$ *is a name and* $\delta \in \mathcal{N}_\delta \cup \{\varepsilon\}$ *is a state.*

Atomic agents are the simplest objects used for describing biological entities. An atomic signature with the same name as the atomic agent defines all allowed states for it (with additional ε state).

Notation 3.2.4 *Let* A *be an atomic agent. We denote by* $\eta(\mathsf{A})$ *its name and by* $\delta(\mathsf{A})$ *its state.*

Definition 3.2.5 *Equivalence of atomic agents*
Let A, A' *be atomic agents.* A *is* equivalent *to* A', *written* $\mathsf{A} \equiv \mathsf{A}'$, iff $\eta(\mathsf{A}) \equiv \eta(\mathsf{A}') \wedge \delta(\mathsf{A}) \equiv \delta(\mathsf{A}')$.

Notation 3.2.6 *We use the symbol* $\mathbb{A}$ *to denote the universe of all possible atomic agents.*

Universe $\mathbb{A}$ represents all atomic agents which can be formed according to defined sets of atomic names and states.

Atomic agents are usually used to express small biological entities which can change their state, for example amino acids, small inorganic molecules etc.

Example 3.2.7 *Atomic agents*

- An atomic agent $\mathsf{A}_1 = (S, u)$
  - written as $S\{u\}$.
- An atomic agent $\mathsf{A}_2 = (Q, \varepsilon)$
  - written as $Q\{\varepsilon\}$.

Note the state ε does not mean the agent is in *none* state, but rather the state is unknown or not important.

3.2.3 Structure agent

Definition 3.2.8 *Structure agent*
We define a structure agent T *as a pair* (η, γ) *where* $\eta \in \mathcal{N}_T$ *is a name and* $\gamma \subseteq \mathbb{A}$ *is a set of atomic agents called partial composition such that* $\forall \mathsf{A}, \mathsf{A}' \in \gamma : \eta(\mathsf{A}) \neq \eta(\mathsf{A}')$.

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 it is constructed only from atomic agents considered in the same physical compartment.

Notation 3.2.9 *Let* T *be a structure agent. We denote by* $\eta(\mathsf{T})$ *its name and by* $\gamma(\mathsf{A})$ *its partial composition.*

Definition 3.2.10 *Equivalence of structure agents*
Let T, T' *be structure agents.* T *is* equivalent *to* T'*, written* $\mathsf{T} \equiv \mathsf{T}'$*,* iff $\eta(\mathsf{T}) \equiv \eta(\mathsf{T}') \wedge \gamma(\mathsf{T}) \equiv \gamma(\mathsf{T}')$.

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

Universe $\mathbb{T}$ represents all structure agents which can be formed according to defined sets of structure names w.r.t. universe of atomic agents.

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. For example, imagine that in our system only three out of a few hundreds amino acids are able to undergo some post-translational modifications, such as phosphorylation, metylation etc.

It is suitable to model only these three acids instead of entire primary structure of the protein.

Example 3.2.12 *Structure agent*

- A structure agent $\mathsf{T}_1 = (K, \{(S, p), (Q, i)\})$
  - written as $K(S\{p\}, Q\{i\})$.
- A structure agent $\mathsf{T}_2 = (K, \{(Q, a)\})$
  - written as $K(Q\{A\})$.

3.2.4 Complex agent

Definition 3.2.13 *Complex agent*

We define a complex agent X *as a pair* $(\mu, \mathtt{com})$ *where* $\mu \in (\mathbb{A} \cup \mathbb{T})^n$ *is a sequence of agents,* $\mathtt{com} \in \mathcal{N}_c$ *is a compartment, and* $n \in \mathbb{N}$.

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 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 space position.

Notation 3.2.14 *Let* X *be a complex agent. We denote by* $\mu(\mathsf{X})$ *its sequence of agents and by* $\mathtt{com}(\mathsf{X})$ *its compartment.*

Definition 3.2.15 *Equivalence of complex agents*

Let X, X' *be complex agents.* X *is* equivalent *to* X', *written* $\mathsf{X} \equiv \mathsf{X}'$, iff $|\mu(\mathsf{X})| \equiv |\mu(\mathsf{X}')| \wedge \mathtt{com}(\mathsf{X}) \equiv \mathtt{com}(\mathsf{X}') \wedge \exists \mu' \in \sigma(\mu(\mathsf{X}'))$ *such that* $\forall i \in [1, n]$: $\pi_i(\mu(\mathsf{X})) \equiv \pi_i(\mu')$ *such that* $n = |\mu(\mathsf{X})|$.

The key element of a complex agent is *sequence* describing inductively constructed coexistence expressions from existing agents.

In contrast to partial composition in structure agent, we allow replication at the level of sequence (an agent of a certain name can appear more than once in a sequence).

Notation 3.2.16 *We use the symbol $\mathbb{X}$ to denote the universe of all possible structure agents. Additionally, $\mathbb{U} = \mathbb{A} \cup \mathbb{T} \cup \mathbb{X}$ denotes universe of all possible agents.*

Example 3.2.17 *Complex agent*

- A complex agent $\mathtt{X} = \big((K, \{(S,p),(Q,i)\}),(S,p)), cytosol\big)$
  - written as $K(S\{p\}, Q\{i\}).S\{p\} :: cytosol$

The complex agents encapsulate other agents – an atomic or 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 space location – the compartment.

3.2.5 Rule

Definition 3.2.18 *Rule*

We define a rule $\mathtt{R}$ *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 left side of χ,*
- $\varphi \in \mathbb{N}^m$ *is an index map from ω to χ,*
- $\psi \in ((\{-\} \cup \mathbb{N})^2)^n$ *is an index map from left-hand side agents to right-hand side agents*

where $n, m \in \mathbb{N}$.

The reason for this particular definition of the rule is a so-called *practical* approach of semantics which can be seen in following sections. Also, it is necessary to capture the relationship between the

left and right-hand side of the rule. This is done by enumerating all atomic and structure agents in ω and consequently creating index map ψ between the agents in ω. Another index map φ serves for relating agents from ω back to original sequence of complexes χ. Finally, by index ι we determine the end of left-hand side of the rule. Note the index is zero in the situation when we have no agents on the left-hand side.

Notation 3.2.19 *We use symbol* $\mathbb{R}$ *to denote the universe of all possible rules.*

Example 3.2.20 *Rule*

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

written as:

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

Not every rule makes sense. For example, a rule with no change or a rule with ambiguous relation of left and right-hand sides. In order to avoid such cases we need to specify when a rule is *well-formed*, i.e. it makes sense semantically.

Definition 3.2.21 *Well-formed rule*

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

1. $\exists (i,j) \in \psi : \pi_i(\chi) \not\equiv \pi_j(\chi)$,
2. $\forall (i,j) \in \psi : \eta(\pi_i(\omega)) \equiv \eta(\pi_j(\omega))$.

A rule is well-formed if it performs any action. This means that some change has to occur during rule application. This is ensured by condition (1), which requires at least one pair of complexes from left and right-hand side of the rule to be different. The second condition (2) has to hold in order to guarantee pairs of structure and atomic agents in ω of the rule have the same name. This condition is crucial for the ground form function $\mathcal{G}$ defined in Section 3.7, where adding context to different agents do not make sense.

Please note the conditions for well-formed rules do not apply on those agents which do not have a pair on the other side of the rule.

3.2.6 Reaction

Definition 3.2.22 *Reaction*

We define reaction r *as a pair* (seq, ι) *where:*

- $\mathrm{seq} \in \mathbb{X}^n$ *is a sequence of complexes,*
- $\iota \in \{0, \ldots, n\}$ *is an index determining the end of left side of* seq

where $n \in \mathbb{N}$.

A reaction is similar structure to rule, but much simpler. The reaction is just a mediate format for giving semantics to BCSL models and it only holds information about complex contents of the rule.

Notation 3.2.23 *Let* r *be a reaction. We denote left-hand side of the reaction* $(\pi_1(\mathrm{seq}), \ldots, \pi_\iota(\mathrm{seq}))$ *as* $LHS(\mathrm{r})$ *and right-hand side of the reaction* $(\pi_{\iota+1}(\mathrm{seq}), \ldots, \pi_{|\mathrm{seq}|}(\mathrm{seq}))$ *as* $RHS(\mathrm{r})$.

3.3 Syntax

In Definition 3.3.1, we provide a grammar defining syntax of the BCSL. As proposed in Section 2, the syntax must be simple and readable. It was already defined in [10], here we provide an improved and simplified version of the definition.

Definition 3.3.1 *Grammar*

atomic expression	$\alpha ::= \eta\{s\} \mid \eta\{\varepsilon\}$ $\eta ::= n \in \mathcal{N}_A$ $s ::= n \in \mathcal{N}_\delta$
structure expression	$\tau ::= \eta(\gamma) \mid \eta(\varnothing)$ $\gamma ::= \alpha_1, \ldots, \alpha_k$ $\eta ::= n \in \mathcal{N}_T$
complex expression	$\Gamma ::= \beta_1 \,.\, \ldots \,.\, \beta_k :: c$ $\beta_i ::= \alpha \mid \tau$ $c ::= n \in \mathcal{N}_c$
rule expression	$\rho ::= \Gamma_1 + \ldots + \Gamma_n \Rightarrow \Gamma_{n+1} + \ldots + \Gamma_m$

where $m, n \in \mathbb{N}_0 \wedge m > n \wedge m + n \neq 0$ *and* $k \in \mathbb{N}$.

Examples of syntax can be seen in the previous examples (Example 3.2.7 to 3.2.20) and also in case study (Section 5).

3.4 BCSL model

When we have defined syntax of BCSL, we can proceed to the BCSL model. We always consider an initialised model, which means the definition contains an initial state of the system. The definition also contains rule expressions and signatures.

Definition 3.4.1 *BCSL model*

BCSL model $\mathbb{M}$ *is a quadruple* $(\mathcal{R}, \Sigma_\mathsf{A}, \Sigma_\mathsf{T}, \texttt{init})$ *where* $\mathcal{R}$ *is a set of rule expressions,* Σ_A *is an atomic signature,* Σ_T *is a structure signature, and* `init` *is an initial multiset of complex expressions.*

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

Remark 3.4.2 *At this point it is useful to sum up what is Biochemical Space as described in Section 1.4.1 formally. The BCS is formed from a database of entities and rules. These entities are actually a signature, i.e. they predefine a domain on which rules can be defined. Neglecting an initial state (which is not relevant for entire space), the BCS is actually a BCSL model.*

Additionally, BCS serves as a reference model for external kinetic models from Models repository. What we do in the process of relation is that we are actually assigning them BCSL notation. This way, smaller BCSL models are build with a reference to BCS. They inherit all annotation information provided by the reference model and gain formal representation of BCSL.

3.5 Additional definitions

Before we proceed, there are a few necessary definitions. These definitions are required in next section in order to generate reactions from rules.

Reassembly operation Ξ is quite a simple action on tuples. We need to reorganise a tuple which has form $((a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n))$ to a tuple of the form $(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n)$. We define reassembly operation Ξ generally on a set of tuples.

Definition 3.5.1 *Reassembly operation*

Let Y be a set of tuples, where each tuple $y \in Y$ has a particular form $((a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n))$ for some $n \in \mathbb{N}$. Then reassembly operation *Ξ is defined as follows:*

$$\Xi(Y) = \{\, x \mid y \in Y \wedge x = (\biguplus_{i=1}^{|y|} \pi_1(y_i)) \uplus (\biguplus_{i=1}^{|y|} \pi_2(y_i)) \,\}$$

We need to define difference of partial compositions of structure agents $\gamma \ominus \gamma'$. This difference is not just set difference, because we are not comparing atomic agents by overall equality, only equality on agent names. This operation is necessary in process of generating reactions from rules, which basically means adding details to partial compositions according to the defined signatures.

Definition 3.5.2 *Difference of partial compositions*
Let γ, γ' be two partial compositions. We define difference of partial compositions *as follows:*

$$\gamma \ominus \gamma' \overset{def}{\Leftrightarrow} \{\eta_\alpha \mid \exists \mathsf{A}\ (\mathsf{A} \in \gamma \wedge \eta(\mathsf{A}) \equiv \eta_\alpha) \wedge \nexists \mathsf{A}'\ (\mathsf{A}' \in \gamma' \wedge \eta(\mathsf{A}') \equiv \eta_\alpha)\}$$

Similarly as in the previous definition, we need to be able to find difference in a context of atomic agent names between partial composition and structure signature.

Definition 3.5.3 *Difference of partial composition and structure signature*
Let T be a structure agent, η its name, and $\Sigma_T(\mathsf{T})$ an appropriate signature. We define difference of partial composition $\gamma(\mathsf{T})$ and structure signature $\Sigma_T(\mathsf{T})$ *as follows:*

$$\Sigma_T(\eta) \ominus \gamma(\mathsf{T}) \overset{def}{\Leftrightarrow} \{\eta_\Sigma \mid \eta_\Sigma \in \Sigma_T(\eta) \wedge \nexists \mathsf{A}\ (\mathsf{A} \in \gamma(\mathsf{T}) \wedge \eta(\mathsf{A}) \equiv \eta_\Sigma)\}$$

The previous two definitions are very similar. The general difference is while in the first case we work purely with two partial compositions, in the second case we have a signature of appropriate structure agent name.

3.6 Semantic function

In the previous sections, we defined BCSL model using given syntax and BCSL objects with semantic meaning. Now, we need to connect them in order to give semantic meaning to the model $\mathbb{M}$. For this

purpose, we define semantic function F (Definition 3.6.1). It is defined recursively for a model $\mathbb{M}$ according to expression given as an argument.

Definition 3.6.1 *Semantic function*

We define semantic function F *according to the expression given in double square brackets* $[\![\ldots]\!]$ *as follows:*

$$\begin{aligned}
\mathsf{F}[\![\ \eta\{\varepsilon\}\ ]\!] &= (\eta, \varepsilon) \in \mathbb{A} \\
\mathsf{F}[\![\ \eta\{s\}\ ]\!] &= (\eta, s) \in \mathbb{A} \\
\mathsf{F}[\![\ \eta(\varnothing)\ ]\!] &= (\eta, \emptyset) \in \mathbb{T} \\
\mathsf{F}[\![\ \eta(\mathsf{a}_1, \ldots, \mathsf{a}_n)\ ]\!] &= \Big(\eta, \{\ \mathsf{F}[\![\mathsf{a}_1]\!], \ldots, \mathsf{F}[\![\mathsf{a}_n]\!]\ \}\Big) \in \mathbb{T} \\
\mathsf{F}[\![\ \alpha_1\ .\ \ldots\ .\ \alpha_n :: c\ ]\!] &= \Big(\ (\mathsf{F}[\![\ \alpha_1\ ]\!], \ldots, \mathsf{F}[\![\ \alpha_n\ ]\!]),\ c\Big) \in \mathbb{X}
\end{aligned}$$

$$\mathsf{F}[\![\ \Gamma_1 + \ldots + \Gamma_n \Rightarrow \Gamma_{n+1} + \ldots + \Gamma_m\ ]\!] = (\chi, \omega, \iota, \varphi, \psi) \in \mathbb{R} \textit{ such that}$$

- $\chi = \Big(\mathsf{F}[\![\ \Gamma_1\ ]\!], \ldots, \mathsf{F}[\![\ \Gamma_n\ ]\!], \mathsf{F}[\![\ \Gamma_{n+1}\ ]\!], \ldots, \mathsf{F}[\![\ \Gamma_m\ ]\!]\Big)$,
- $\omega = +\!\!+_{i=1}^{|\chi|} \mu(\pi_i(\chi))$,
- $\iota = n$,
- $\varphi = (J_1, \ldots, J_m)$ *where* $J_k = \sum_{i=1}^{k} |\mu(\pi_i(\chi))|$,
- $\psi = \{\ (i,j) \mid i \in [1, \pi_\iota(\varphi)] \wedge j \in [\pi_\iota(\varphi)+1, |\omega|] \wedge |i-j| \equiv \pi_\iota(\varphi)\ \} \cup$
  $\{\ (i,-) \mid i \in [k, \pi_\iota(\varphi)] \wedge k = |\omega| - \pi_\iota(\varphi) + 1\ \mid\ \} \cup$
  $\{\ (-,j) \mid j \in [k, |\omega|] \wedge k = 2 \times \pi_\iota(\varphi) + 1\ \}$

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

Note that by applying semantic function F on expressions from Examples 3.2.7 to 3.2.20, we obtain agents (resp. rules) from appropriate examples. Moreover, semantic function works *only* on BCSL object and rules well-defined by the given grammar (Section 3.3).

3.7 Ground forms

Ground form represents appending context to agents according to signature (we assume signature is always available and therefore we omit it in ground from function arguments). Particularly, this process means adding details to partial compositions and adding states of atomic agents according to defined signatures. Since there are typically multiple states allowed for an atomic agent, this processes leads to multiple options how to add a context. This is natural consequence of fact that a rule is an abstract representation of multiple reactions.

Definition 3.7.1 *Ground form function*

We define ground form function $\mathcal{G}$ *for objects (middle column) of particular type (left column) as:*

Type	***Notation***	***Ground form***
$\mathbb{A}$	$\mathcal{G}(\eta, \varepsilon)$ $\mathcal{G}(\eta, \delta)$	$\{ (\eta, \delta) \mid \delta \in \Sigma_A(\eta) \}$ $\{ (\eta, \delta) \}$
$\mathbb{A}^2$	$\mathcal{G}(\mathsf{A}, \mathsf{A}')$	$\{ (a, a') \mid a \in \mathcal{G}(\mathsf{A}) \wedge a' \in \mathcal{G}(\mathsf{A}') \wedge \delta(a) \equiv \delta(a') \}$
$\mathbb{T}$	$\mathcal{G}(\eta, \gamma)$	$\left\{ (\eta, \gamma') \;\middle\vert\; \begin{array}{l} \gamma' \equiv \gamma \cup \gamma_\Sigma \wedge \gamma_\Sigma \subseteq \mathcal{G}(\eta', \varepsilon) \wedge \\ \wedge \lvert\gamma_\Sigma\rvert = \lvert\Sigma_T(\eta) \ominus \gamma\rvert \wedge \eta' \in \Sigma_T(\eta) \ominus \gamma \wedge \\ \wedge \forall \widetilde{\eta} \in \Sigma_T(\eta) \ominus \gamma : \widetilde{\eta} \in \gamma_\Sigma \end{array} \right\}$
$\mathbb{T}^2$	$\mathcal{G}(\mathsf{T}, \mathsf{T}')$	$\left\{ (t, t') \;\middle\vert\; \begin{array}{l} t \in \mathcal{G}(\mathsf{T}) \wedge t' \in \mathcal{G}(\mathsf{T}') \wedge \\ \wedge \gamma(t) \ominus \gamma(\mathsf{T}) \equiv \gamma(t') \ominus \gamma(\mathsf{T}') \end{array} \right\}$
$\mathbb{R}$	$\mathcal{G}(\mathsf{R})$	$\prod_{k=1}^{\lvert\mathcal{G}(\psi)\rvert} \mathcal{G}(\psi)_k$

where $\mathcal{G}(\psi)_k = \mathcal{G}(\pi_i(\omega), \pi_j(\omega))$ *such that* $(i, j) = \pi_k(\psi)$ *and* $\mathsf{R} = (\chi, \omega, \iota, \varphi, \psi)$. *Moreover,* $\mathsf{A} = (\eta, \delta)$, $\mathsf{A}' = (\eta', \delta')$, $\mathsf{T} = (\eta, \gamma)$, *and* $\mathsf{T}' = (\eta', \gamma')$.

In Definition 3.7.1, objects (column *notation*) of a particular type (column *type*) are transformed to their ground forms by ground form function $\mathcal{G}$ (column *ground from*).

In case of single atomic agent A, there are two options – either the state is ε or the state δ is given. In the first case, we construct a set of all possible atomic agents according to the given atomic signature $\Sigma_A(\eta(\mathsf{A}))$. In the second case, we construct a set containing only the original agent itself.

For a pair of atomic agents A, A' we create a set of all possible pairs (a, a') such that these pairs have equal names.

Ground form of a structure agent T is a set of structure agents T' such that we *insert* all neglected context information into its partial composition. In other words, we need to determine which atomic agents are missing in the partial composition, which can be obtained from the signature $\Sigma_T(\eta(\mathsf{T}))$. Then we create all possible combinations of inserted atomic agents recursively.

For a pair of structure agents T, T' we expand both partial compositions recursively such that the expanded parts are equal, i.e. we add the same context to both agents. Similarly as in previous cases, we create a set of all possibilities of such pairs.

Finally, ground form of a rule R is a Cartesian product of sets created for each pair of agents by applying ground from function. The pairs are given by order in index map ψ.

3.8 Reactions construction

Given a rule R, we can produce appropriate reactions using ground forms and subsequent application of reassembly operation Ξ.

Definition 3.8.1 *Reactions construction*

$$\begin{aligned}
&\mathtt{Reactions}(\mathsf{R}) = \{\, (\mathtt{seq}, \iota) \mid y \in \Xi(\mathcal{G}(\mathsf{R})) \,\} \\
&\quad \textit{where } \mathtt{seq} = (\mathsf{X}_1, \mathsf{X}_2, \ldots, \mathsf{X}_n) \textit{ such that} \\
&\forall i \in [1, \ldots, n] : \mathsf{X}_i = \Big((\pi_{\pi_{i-1}(\varphi)+1}(y), \ldots, \pi_{\pi_i(\varphi)}(y)), \mathtt{com}(\pi_i(\chi)) \Big) \\
&\qquad\qquad \textit{for } n = |\chi|,\ \varphi_0 = 0,\ \iota \in \mathsf{R},\ \chi \in \mathsf{R},\ \varphi \in \mathsf{R}.
\end{aligned}$$

In Defintion 3.8.1 we create the reassembled ground forms for a rule R. Then, we create a set of reactions from created tuples of

agents such that we preserve structure of original complex agents of rule $\mathtt{R}$. They are constructed using sequence of indices φ, which indicates the last positions in complex structures. In the case when for a particular complex agent $\mathtt{X}_i$ holds that $\pi_{\pi_{i-1}(\varphi)+1}(y) = \pi_{\pi_i(\varphi)}(y)$, the produced sequence μ is a singleton.

3.9 Chemical reaction networks

In this section, we define a simplified model that we can use to define operational semantics of a BCSL model.

Definition 3.9.1 *Chemical reaction model*

Chemical reaction model *(CRM)* $\mathcal{M}$ *is a triple* $(\mathfrak{K}, \vec{v}, \theta)$ *such that* $\mathfrak{K} \subseteq \mathbb{Z}^n$ *is set of chemical reactions,* $\vec{v} \in \mathbb{N}_0^n$ *is the initial vector, and* $\theta \in \mathbb{X}^n$ *is the vector of reference complexes for some* $n \in \mathbb{N}$.

Chemical reaction model is a simple model, where chemical reactions are vectors over integers and initial state is vector over natural numbers. Moreover, there is a reference vector of complex agents, which serves for identification of the underlying agent on particular position in the vectors.

Notation 3.9.2 *We denote* $N(\vec{\mathrm{u}})$ *condition* $\forall i \in [1, \ldots, |\vec{\mathrm{u}}|] : \vec{\mathrm{u}}_i \in \mathbb{N}_0$.

By Notation 3.9.2 we denote the condition that a vector is formed only by natural numbers including zero.

Definition 3.9.3 *Chemical reaction application*

Chemical reaction application $\zeta \subseteq \mathbb{N}_0^n \times \mathbb{Z}^n \times \mathbb{N}_0^n$ *is a relation such that*

$$(\vec{v}, \vec{\mathrm{r}}, \vec{\mathrm{u}}) \in \zeta \overset{def}{\Leftrightarrow} \vec{\mathrm{u}} = \vec{v} + \vec{\mathrm{r}} \wedge N(\vec{\mathrm{u}}).$$

Note that some values in chemical reactions might be negative. When applying a reaction on a state, the resulting state cannot contain any negative values (negative number of entities does not give biological sense). Therefore, application is allowed only if the resulting vector contains non-negative values.

Definition 3.9.4 *Semantics of chemical reaction model*

Let $\mathcal{M} = (\mathfrak{R}, \vec{\nu}, \theta)$ *be a chemical reaction model. The* semantics of chemical reaction model $\mathcal{M}$ *are given in terms of the labelled transition system,* $LTS(\mathcal{M})$*, is defined as a quadruple* (S, A, T, s_0) *such that:*

1. *S is defined inductively:*

   Basis*:* $\vec{\nu} \in S$

   Induction*:* $\vec{s} \in S \Rightarrow \forall \vec{r} \in \mathfrak{R} : \vec{p} \in S$ *such that* $(\vec{s}, \vec{r}, \vec{p}) \in \zeta$

2. $A = \mathfrak{R}$,

3. $T = \{ (\vec{s}, \vec{r}, \vec{p}) \mid \vec{s}, \vec{p} \in S \wedge \vec{r} \in \mathfrak{R} \wedge (\vec{s}, \vec{r}, \vec{p}) \in \zeta \}$,

4. $s_0 = \vec{\nu}$.

The quantitative semantics of model $\mathcal{M}$ are given in terms of a *discrete-states* produced by inductive reaction application on the initial vector $\vec{\nu}$ and all consequently created state vectors. This way we obtain an $LTS(\mathcal{M})$.

CRM semantics might lead to possibly infinite LTS. This process can be limited by a global bound, which makes sure the produced LTS is finite (Section 4.2.6).

3.10 Model construction

In this section, we define a relation between CRM and BCSL models and show how can their LTSs be translated to each other.

Definition 3.10.1 *Reference vector*

Let Θ *be a set of all possible unique complexes constructed from reactions of model* $\mathbb{M}$ *defined as follows:*

$$\Theta = \{ X \mid X \in \mathtt{seq}(r) \wedge r \in \bigcup_{R \in \mathcal{R}} \mathtt{Reactions}(R) \}$$

Then, the reference vector θ *is a tuple* $\theta \in \Theta^n$ *such that* $\forall i, j \leq n : \theta_i \not\equiv \theta_j$ *and* $n = |\theta|$*. Note there are* $n!$ *possibilities how to choose* θ*; an arbitrary one can be chosen.*

The reference vector enumerates all possible complex agents created for a model and assigns them a unique index.

Definition 3.10.2 *Multiset to vector translation*
Translation of a multiset $\mathsf{S} \subseteq \mathbb{X}$ to a vector $(a_1, a_2, \ldots, a_n) \in \mathbb{N}^n$ *is defined as follows:*

$$\lambda(\mathsf{S}, \theta) = (a_1, a_2, \ldots, a_n) \text{ such that } a_i = \begin{cases} |\mathsf{S}(\theta_i)| & \ldots \quad \theta_i \in \mathsf{S} \\ 0 & \ldots \quad \theta_i \notin \mathsf{S} \end{cases}$$

where $n = |\theta|$ *and* $|\mathsf{S}(\theta_i)|$ *is number of occurrences of agent* θ_i *in the multiset* S.

Let $\mathbb{M} = (\mathcal{R}, \Sigma_{\mathsf{A}}, \Sigma_{\mathsf{T}}, \mathtt{init})$ be a BCSL model. In order to construct chemical reaction model $\mathcal{M} = (\mathfrak{R}, \vec{\nu}, \theta)$ from the model $\mathbb{M}$, we need to apply the following steps:

1. generate reactions for all rules $\mathsf{F}(\rho) \in \mathcal{R}$ (Definition 3.8.1),
2. construct reference vector θ (Definition 3.10.1),
3. create initial vector $\vec{\nu} = \lambda(\mathsf{S}, \theta)$ from initial multiset S where $\mathsf{S} = \{\mathsf{X} \mid \mathsf{X} = \mathsf{F}(\Gamma) \wedge \Gamma \in \mathtt{init}\}$ (Definition 3.10.2),
4. construct set of chemical reactions $\mathfrak{R}$ from the set of reactions, where individual vector reaction $\vec{r}$ is created from a reaction r as following:

$$\vec{r} = \lambda(RHS(\mathrm{r}), \theta) - \lambda(LHS(\mathrm{r}), \theta)$$

In Definition 3.9.4 we show how to create an LTS from a chemical reaction model. In order to accomplish the process of giving semantics to an BCSL model, it is necessary to show how we revive information which is missing in the produced LTS. This procedure is quite straightforward since we have defined reference vector θ. Similarly as we translated multisets of BCSL agents to vectors (Definition 3.10.2), we can apply a reverse procedure, when we build a multiset of agents from a vector according to the reference vector θ (Definition 3.10.3).

Definition 3.10.3 *Vector to multiset translation*

Translation of a vector $(a_1, a_2, \ldots, a_n) \in \mathbb{N}^n$ to a multiset $\mathsf{S} = (A, \mathtt{m})$ *is defined as follows:*

$$\lambda^{-1}((a_1, a_2, \ldots, a_n), \theta) = \mathsf{S} \text{ such that}$$
$$A = \{\ \theta_i \mid i \in [1, \ldots, n] \wedge a_i > 0\ \}$$
$$\mathtt{m} = \{\ (\theta_i, a_i) \mid i \in [1, \ldots, n] \wedge a_i > 0\ \}$$

where $n = |\theta|$.

The procedure provided in Definition 3.10.3 can be extended to the labels of the LTS. The labels, which represent reactions applied, might contain negative numbers.

Definition 3.10.4 *Negative part of vector to multiset translation*

Translation of the negative *part a vector* $\overline{\lambda^{-1}}$ *is defined equally than the translation in Definition 3.10.3 with exception that the condition* $a_i > 0$ *is modified to* $a_i < 0$.

The positive and negative parts of labels are treated independently. The negative numbers represent left-hand side of the reaction $\overline{\lambda^{-1}}$ (the reactants) while positive numbers represent right-hand side of the reaction λ^{-1} (the products).

Definition 3.10.5 *LTS of BCSL model*

Let $\mathbb{M} = (\mathcal{R}, \Sigma_{\mathsf{A}}, \Sigma_{\mathsf{T}}, \mathtt{init})$ *be a BCSL model and* $\mathcal{M} = (\mathfrak{K}, \vec{v}, \theta)$ *be a chemical reaction model constructed from model* $\mathbb{M}$.

We define the labelled transition system of the BCSL model $\mathbb{M}$, $LTS(\mathbb{M}) = (S, A, T, s_0)$, *from the* $LTS(\mathcal{M}) = (S', A', T', s_0')$ *such that:*

1. $S = \{\ \mathrm{s} \mid \mathrm{s} = \lambda^{-1}(\mathrm{s}', \theta) \wedge \mathrm{s}' \in S'\ \}$,
2. $A = \{\ (\mathrm{l}, \mathrm{r}) \mid \mathrm{l} = \overline{\lambda^{-1}}(\mathrm{a}, \theta) \wedge \mathrm{r} = \lambda^{-1}(\mathrm{a}, \theta) \wedge \mathrm{a} \in A'\ \}$,
3. $$T = \left\{ (\mathrm{s}, (\mathrm{l}, \mathrm{r}), \mathrm{t}) \;\middle|\; \begin{array}{c} \mathrm{s} = \lambda^{-1}(\vec{\mathrm{s}}, \theta) \wedge \mathrm{t} = \lambda^{-1}(\vec{\mathrm{p}}, \theta) \wedge \\ \wedge\ l = \overline{\lambda^{-1}}(\vec{\mathrm{r}}, \theta) \wedge r = \lambda^{-1}(\vec{\mathrm{r}}, \theta) \wedge \\ \wedge\ (\vec{\mathrm{s}}, \vec{\mathrm{r}}, \vec{\mathrm{p}}) \in T' \end{array} \right\}$$
4. $s_0 = \lambda^{-1}(s_0', \theta)$.

The LTS of a BCSL model is formed by states, labels, transitions, and an initial state. The states are multisets of complex agents, labels are individual reactions (the left and right-hand side multisets of complex agents respectively), and transitions are triples of (state, label, state) which represent edges. The initial state is a special state representing the origin of the system.

3.11 Syntactic extensions

We define several syntactic extensions for better readability of the rule expressions. Note that each rule expression in an extended form can be always translated to basic form defined above. All rule expressions containing the following extensions must be converted to basic form before semantics can be applied.

For better demonstration we provide a running example, which will go through all syntactic extensions. Please note there is no biological sense of the model, it has only purpose to effectively demonstrate all defined syntactic extensions.

Running example 3.11.1 *The example model* $\mathbb{M}$
Definition of rule expressions:

1. $KaiC(S\{u\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt \Rightarrow$
   $\Rightarrow KaiC(S\{p\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt$
2. $KaiC(S\{u\}, T\{\varepsilon\}).KaiB(\varnothing) :: cyt \Rightarrow KaiC(S\{p\}, T\{\varepsilon\}).KaiB(\varnothing) :: cyt$
3. $KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt + KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt + KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt \Rightarrow$
   $\Rightarrow KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt$
4. $KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}).KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt \Rightarrow$
   $\Rightarrow KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt + KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt + KaiC(S\{\varepsilon\}, T\{\varepsilon\}) :: cyt$

Definition of Σ_A:	Definition of Σ_T:
1. $(S, \{u, p\})$	1. $(KaiC, \{S, T\})$
2. $(T, \{a, i\})$	2. $(KaiB, \emptyset)$

We omit `init` definition just for simplicity of the example.

3.11.1 Partial composition context elimination

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

Running example 3.11.2 *The example model* $\mathbb{M}$
Definition of rule expressions:

1. $KaiC(S\{u\}).KaiC(\varnothing).KaiC(\varnothing) :: cyt \Rightarrow KaiC(S\{p\}).KaiC(\varnothing).KaiC(\varnothing) :: cyt$
2. $KaiC(S\{u\}).KaiB(\varnothing) :: cyt \Rightarrow KaiC(S\{p\}).KaiB(\varnothing) :: cyt$
3. $KaiC(\varnothing) :: cyt + KaiC(\varnothing) :: cyt + KaiC(\varnothing) :: cyt \Rightarrow$
   $\Rightarrow KaiC(\varnothing).KaiC(\varnothing).KaiC(\varnothing) :: cyt$
4. $KaiC(\varnothing).KaiC(\varnothing).KaiC(\varnothing) :: cyt \Rightarrow$
   $\Rightarrow KaiC(\varnothing) :: cyt + KaiC(\varnothing) :: cyt + KaiC(\varnothing) :: cyt$

Additionally, this extension can be done more precisely by omitting the $(\varnothing)$ part completely. Since we have the structure signature defined, we can unambiguously determine which names belong to structure expressions and this syntactic part can be easily reconstructed.

Running example 3.11.3 *The example model* $\mathbb{M}$
Definition of rule expressions:

1. $KaiC(S\{u\}).KaiC.KaiC :: cyt \Rightarrow KaiC(S\{p\}).KaiC.KaiC :: cyt$
2. $KaiC(S\{u\}).KaiB :: cyt \Rightarrow KaiC(S\{p\}).KaiB :: cyt$
3. $KaiC :: cyt + KaiC :: cyt + KaiC :: cyt \Rightarrow KaiC.KaiC.KaiC :: cyt$
4. $KaiC.KaiC.KaiC :: cyt \Rightarrow KaiC :: cyt + KaiC :: cyt + KaiC :: cyt$

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

3.11.2 Complex signature

We extend the model definition by complex signature Σ_χ. In this signature, there are defined aliases for valid complex expressions. Then, simple *replace* method is used to substitute original expression by the alias.

Running example 3.11.4 *The example model* $\mathbb{M}$
Definition of complex signature Σ_χ:

1. $KaiC3 :: cyt = KaiC.KaiC.KaiC :: cyt$
2. $KaiBC :: cyt = KaiC.KaiB :: cyt$

Definition of rule expressions:

1. $KaiC(S\{u\}).KaiC.KaiC :: cyt \Rightarrow KaiC(S\{p\}).KaiC.KaiC :: cyt$
2. $KaiC(S\{u\}).KaiB :: cyt \Rightarrow KaiC(S\{p\}).KaiB :: cyt$
3. $KaiC :: cyt + KaiC :: cyt + KaiC :: cyt \Rightarrow KaiC3 :: cyt$
4. $KaiC3 :: cyt \Rightarrow KaiC :: cyt + KaiC :: cyt + KaiC :: cyt$

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

3.11.3 Directions

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

Running example 3.11.5 *The example model* $\mathbb{M}$
Definition of rule expressions:

1. $KaiC(S\{u\}).KaiC.KaiC :: cyt \Rightarrow KaiC(S\{p\}).KaiC.KaiC :: cyt$
2. $KaiC(S\{u\}).KaiB :: cyt \Rightarrow KaiC(S\{p\}).KaiB :: cyt$
3. $KaiC :: cyt + KaiC :: cyt + KaiC :: cyt \Leftrightarrow KaiC3 :: cyt$

Definition of rules 3 and 4 from Running example 3.11.4 was replaced by one bi-directional rule 3 in Running example 3.11.5.

3.11.4 Stoichiometry

For a rule expression of form

$$\beta_1 :: \mathrm{c} + \beta_2 :: \mathrm{c} + \ldots + \beta_n :: \mathrm{c} \Rightarrow \beta_1.\beta_2. \ \ldots \ .\beta_n :: \mathrm{c}$$

we might reorder both sides such that we get non-crossing partition $\mathcal{P} = B_1/B_2/\ldots/B_k$ from its indices $[1,\ldots,n]$ such that:

$$\begin{gathered}\forall B \in \mathcal{P}\ \forall i,j \in B : \beta_i \equiv \beta_j \\ \text{and} \\ \forall B,B' \in \mathcal{P}\ \forall \beta \in B\ \forall \beta' \in B' : \beta \not\equiv \beta'. \\ \text{such that } B \neq B'.\end{gathered}$$

For the left-hand side $\beta_1 :: \mathrm{c} + \beta_2 :: \mathrm{c} + \ldots + \beta_n :: \mathrm{c}$ of the reordered rule expression we can replace all rule expressions $[\beta_i,\ldots,\beta_j]$ which belong to the same non-crossing partition B by notation '$k\ \beta$', where β is a representative from $\beta_i,\ldots,\beta_j$ (they are all equivalent) and k is the number of the expressions in partition B.

Such a new rule expression is equivalent with the original rule expression, which follows from the appropriate rules equivalence definition (Definition 3.2.15).

Note that this process is fully reversible – we can simply enumerate all expressions for each partition.

Running example 3.11.6 *The example model* $\mathbb{M}$
Definition of rule expressions:

1. $KaiC(S\{u\}).KaiC.KaiC :: cyt \Rightarrow KaiC(S\{p\}).KaiC.KaiC :: cyt$
2. $KaiC(S\{u\}).KaiB :: cyt \Rightarrow KaiC(S\{p\}).KaiB :: cyt$
3. $3\ KaiC :: cyt \Leftrightarrow KaiC3 :: cyt$

Definition of rule expression 3 from Running example 3.11.5 was replaced by a new rule expression using stoichiometry.

3.11.5 Locations

Probably the most complex syntactic extension is application of locations. The localisation operator is intended for allowing an alternative way of expressing the hierarchically constructed agent expressions. The main idea is to allow zooming into individual parts of complex and structure expressions.

We define necessary condition which must hold when semantic function F is applied on individual expressions. This notation is allowed only when the conditions hold. The condition is expressed using *compatibility* operator defined in Section 4.2.

Definition 3.11.7 *Locations conditions.*

1. $\mathsf{A} :: \mathsf{T} \Leftrightarrow$ *there exists* $\mathsf{A}' \in \gamma(\mathsf{T})$ *such that* $\mathsf{A} \lhd \mathsf{A}'$,
2. $\mathsf{A} :: \mathsf{X} \Leftrightarrow$ *there exists* $\mathsf{A}' \in \mu(\mathsf{X})$ *such that* $\mathsf{A} \lhd \mathsf{A}'$,
3. $\mathsf{T} :: \mathsf{X} \Leftrightarrow$ *there exists* $\mathsf{T}' \in \mu(\mathsf{X})$ *such that* $\mathsf{T} \lhd \mathsf{T}'$.

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

Running example 3.11.8 *The example model* $\mathbb{M}$
Definition of rule expressions:

1. $S\{u\} :: KaiC :: Kai C3 :: cyt \Rightarrow S\{p\} :: KaiC :: Kai C3 :: cyt$
2. $S\{u\} :: KaiC :: KaiBC :: cyt \Rightarrow S\{p\} :: KaiC :: KaiBC :: cyt$
3. $3\ KaiC :: cyt \Leftrightarrow Kai C3 :: cyt$

Definition of rule expressions 1 and 2 from Running example 3.11.6 was replaced using locations.

You can see the localisation operator allowed us to fully use the complex signatures.

3.11.6 Variables

In the Running example 3.11.8 there are additional options for syntax reduction. Rule expressions 1 and 2 are very similar except for the context of complex expression they take place in. Therefore, we will substitute this context with a variable with a given domain.

In a rule expression, one rule agent expression might be referenced using a variable as a set of rule agent expressions it can be replaced with. Such a rule agent expression is referenced as $?X$. Moreover, in the case when a $?X$ is used in a location, it must hold conditions from Definition 3.11.7.

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 3.11.9 *The example model* $\mathbb{M}$
Definition of rule expressions:

1. $S\{u\} :: KaiC :: ?X :: cyt \Rightarrow S\{p\} :: KaiC :: ?X :: cyt \; ; \; ?X = \{KaiC3, KaiBC\}$
2. $3\ KaiC :: cyt \Leftrightarrow KaiC3 :: cyt$

Definition of rule expressions 1 and 2 from Running example 3.11.8 was replaced as a single rule expression with a variable.

This is the final syntactic extension. Compared to the original model (Running example 3.11.1), the resulting model is more concise and readable. More examples are in Section 5.

4 Analysis

In the previous chapters, we defined the BCS language. From the provided examples and the case study (Section 5) it should be clear that it provides concise and readable notation when it comes to describing biological systems. Moreover, accompanied with relevant annotation information, it is suitable for building extensive models.

However, descriptive properties are not enough for using computational methods in order to analyse particular processes. In this chapter we describe basic approaches we employ for our language and propose several methods which could be potentially used in the context of static analysis.

We distinguish two general approaches in model analysis – dynamic and static. Dynamic analysis is enabled by some form of model execution. It requires either enumerating all the possible scenarios of behaviour of a model (called *transition system*) or applying assumptions in order to calculate an average behaviour (such as deterministic simulation). Static analysis provides information without execution of the model, just by investigating its structure. Both approaches require to build particular mechanisms which enable application of the analysis.

4.1 Dynamic analysis

Dynamic analysis is a very common way for analysing models written in formal representations. There are several types of analyses and they are usually trying to predict behaviour of the modelled system using formal description of the system – the model.

4.1.1 Model checking

Model checking is an approach of checking whether a given model meets a given specification [5]. Typically this is done by specifying given requirements in a temporal logic formula and then checking whether a given structure satisfies the formula.

The model checking of BCSL models can be enabled because we are able to generate LTS (Definition 3.1.8), but it is not a goal of this

thesis. However, we are interested in reachability analysis, what is a particular task of model checking for determining whether a chosen state is reachable in the LTS of a model from a given set of initial states.

Definition 4.1.1 *Reachability relation*
For an LTS $\mathcal{L} = (S, A, T, s_0)$ *the* reachability relation *of* $\mathcal{L}$ *is the transitive closure of* T*, which is the set of all ordered pairs* (s, t) *of states in* S *for which there exists a sequence of states* $s = s_0, s_1, s_2, \ldots, s_k = t$ *such that the transition* (s_{i-1}, a, s_i) *is in* T *for all* $1 \leq i \leq k$ *and for some* a *in* A*.*

In our case, reachability analysis can be generalised to checking whether for some multiset of complex agents $\mathfrak{s}'$ there exists a reachable state $s \in S$ of an LTS $\mathcal{L}$ such that $\mathfrak{s}' \subseteq s$. In other words, we are interested only in particular agents regardless of other context in the reachable state. In general, it might lead to multiple reachable states and then we can for example consider a shortest path.

In the case when we consider reachability *only* from the initial state s_0, the task becomes simpler. The reason is all other states are actually reachable, because the entire transition system was generated from this initial state. Then we are only searching for a state $s \in S$ such that $\mathfrak{s}' \subseteq s$ in the case when we are not interested in *how* was the state reached (the particular path).

4.1.2 Simulation

Computer simulations reproduce the behaviour of a system using a mathematical model. For the purpose of this thesis, the simulation is a numerical solution which tries to mimic the behaviour of the real system over time. Once we want to simulate BCSL model, regardless of the approach we choose, it is necessary to enrich the rules by kinetic rates [27]. The rates basically express the speed of the rule or the probability that the rule will be performed.

Assigning a rate to the rule is not a simple task. The simplest solution is to reference only fully defined objects in the kinetic rates. However, we would like to express the context of the rule also in the rate. That typically means changing context in rate and the rule accordingly. This approach is non-trivial and we keep it as a future work.

Assigning rates to the rules enables deterministic simulation [31] by translating rules to reactions and consequently construct Ordinary Differential Equations (ODEs) from them [17]. It is possible to enable also stochastic simulation, either by network-free simulators such as NFsim [32] or using indirect approach on generated reactions (e.g. Gillespie algorithm [13]).

4.2 Static analysis

The BCS language offers interesting capabilities in order to provide static analysis of given models. We define the *compatibility* operator $\lhd$ for each type of agents, which will help us to formulate properties suitable for the static analysis.

Definition 4.2.1 *Compatibility of atomic agents*
Let A_1, A_2 *be atomic agents. The agent* A_1 *is* compatible with *agent* A_2, *written* $\mathsf{A}_1 \lhd \mathsf{A}_2$, *if either* $\mathsf{A}_1 \equiv \mathsf{A}_2$ *or* $\eta(\mathsf{A}_1) \equiv \eta(\mathsf{A}_2) \wedge \delta(\mathsf{A}_1) \equiv \varepsilon$.

Note that compatibility of two atomic agents is very close to their equivalence. Additionally the left-hand side agent is allowed to have its state undefined while the state of right-hand side agent differs.

Definition 4.2.2 *Compatibility of structure agents*
Let $\mathsf{T}_1, \mathsf{T}_2$ *be structure agents. The agent* T_1 *is* compatible with *agent* T_2, *written* $\mathsf{T}_1 \lhd \mathsf{T}_2$, iff *either* $\mathsf{T}_1 \equiv \mathsf{T}_2$ *or* $\eta(\mathsf{T}_1) \equiv \eta(\mathsf{T}_2) \wedge \forall \mathsf{A}_1 \in \gamma(\mathsf{T}_1)\, \exists \mathsf{A}_2 \in \gamma(\mathsf{T}_2) : \mathsf{A}_1 \lhd \mathsf{A}_2$.

Structure agents are compatible if we are able to create pairs from atomic agents of composition of left-hand side agent with the right-hand side ones such that these atomic agents are all unique. For such pairs, the agents in each pair must be compatible.

Definition 4.2.3 *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* $\mathsf{X}_1 \lhd \mathsf{X}_2$, iff *either* $\mathsf{X}_1 \equiv \mathsf{X}_2$ *or* $\mathrm{com}(\mathsf{X}_1) \equiv \mathrm{com}(\mathsf{X}_2) \wedge \exists \mu' \in \sigma(\mu(\mathsf{X}_2))$ *such that* $\forall i \in [1, n] : \pi_i(\mu(\mathsf{X}_1)) \lhd \pi_i(\mu')$, *where n is length of sequence which is the same for both sequences.*

Complex agents are compatible if there exists a permutation of sequence of the first agent such that individual agents on the same position in both sequences are compatible.

The insight from the previous chapters is that an agent might have its ancestors and successors. The compatibility operator formulates this relationship formally. Each successor is compatible with its ancestor, but the reverse relation does not hold. The successor has always more details specified.

Notation 4.2.4 *Let x, y be two agents. We denote $x \Delta y$ the fact the agents are compatible* iff *either $x \lhd y$ or $y \lhd x$ holds.*

Definition 4.2.5 *Partial order of agents*

Let $x_1, x_2 \in P$ be agents and P a universe of agents. Agents x_1 and x_2 are in partial order *relation, written $x_1 \leq x_2$,* iff *$x_1 \lhd x_2$.*

Notation 4.2.6 *The universe of complex agents $\mathbb{X}$ ordered by partial order relation $\leq$ is denoted as $\mathbb{X}_{\leq}$.*

The compatibility operator generates partial order on the $\mathbb{A}, \mathbb{T}$, and $\mathbb{X}$ sets. For our purposes, partially ordered set $\mathbb{X}_{\leq}$ is relevant. The reason is that complex agents actually encapsulate all the other agent types. However, partial order of entire universe of complex agents is not very useful, since most of agents cannot be compared. We are interested in particular subsets which are formed from all complexes of the same type, i.e. those sets where for each pair of complexes exists another one such that they are compatible with this complex.

Definition 4.2.7 *Compatible set*

A finite set $\mathscr{X} \subseteq \mathbb{X}$ is a compatible set *if:*

1. $\forall \mathsf{X}_1, \mathsf{X}_2 \in \mathscr{X}\ \exists \mathsf{X}' \in \mathscr{X} : \mathsf{X}_1 \lhd \mathsf{X}' \wedge \mathsf{X}_2 \lhd \mathsf{X}'$ *such that* $\mathsf{X}_1 \neq \mathsf{X}_2$,

   and for each finite set $\mathscr{X}' \subseteq \mathbb{X}$ such that $\mathscr{X} \neq \mathscr{X}'$ holds:

2. $\forall \mathsf{X} \in \mathscr{X}\ \forall \mathsf{X}' \in \mathscr{X}' : \neg(\mathsf{X} \Delta \mathsf{X}')$.

Remark 4.2.8 *The compatible set $\mathscr{X}$ inherits partial order relation defined on $\mathbb{X}_{\leq}$, which follows from conditions (1) and (2) of Definition 4.2.7.*

A compatible set $\mathscr{X}$ contains partially ordered complex agents. Assuming we ignore states of individual atomic agents (including those inside partial compositions of structure agents), these complex agents are all equal. In other words, they all contain the same sequence of agent names. Example of a compatible set is given in Figure 4.1.

Lemma 4.2.9 *In every compatible set $\mathscr{X}$, there always exists a global supremum* $\mathsf{sup}(\mathscr{X})$*, which is an ancestor of all the remaining agents in the set (Remark 4.2.8).*

Proof: The lemma follows from Definition 4.2.7 condition (1) which basically claims that there is a supremum (in manner of compatibility) for each two complex agents in the compatible set $\mathscr{X}$. Since there exists a supremum for every two items in the set and the set is finite, there must exists a global supremum for the entire set. ■

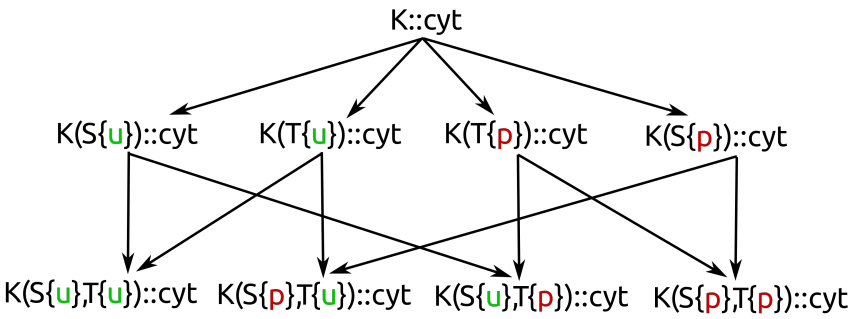


Figure 4.1: An example of a compatible set $\mathscr{X}$. For better readability, the agents are written in BCSL syntax. The set is formed by a complex in *cyt* compartment, which has only one structure agent *K* in its sequence. The structure agent *K* has allowed atomic agents *T* and *S* in its partial composition. These two atomic agents might occur in two states – *u* and *p*. The set is complete – there are all relevant agents bounded by compatibility operator.

Lemma 4.2.10 *For every complex agent* X *there exists exactly one compatible set* $\mathscr{X} \subseteq \mathbb{X}$ *such that* $\mathsf{X} \in \mathscr{X}$.

Proof: Let us assume a complex agent X *belongs to two compatible sets, namely* $\mathsf{X} \in \mathscr{X}_1, \mathscr{X}_2$. *From Definition 4.2.7 condition (1) follows that there exists a* $\mathsf{X}_1 \in \mathscr{X}_1$ *such that* $\mathsf{X} \lhd \mathsf{X}_1$.

Then, the condition (2) claims that no complex agent from $\mathscr{X}_1$ *and no complex agent from* $\mathscr{X}_2$ *can be compatible. Namely,* $\mathsf{X}_1 \in \mathscr{X}_1$ *cannot be compatible with* $\mathsf{X} \in \mathscr{X}_2$. *However,* X *and* X_1 *are compatible (namely* $\mathsf{X} \lhd \mathsf{X}_1$*). It follows* $\mathsf{X} \notin \mathscr{X}_2$, *what is a contradiction.* ■

Practically, compatible sets can be used for static analysis on level of complexes 4.2.2 and for finding non-trivial relationships between the rules 4.2.1.

Note that there always exists a compatible set $\mathscr{X}$ for a complex agent X which follows from Lemma 4.2.10.

Definition 4.2.11 *Compatible subset*

Let $\mathscr{X} \subseteq \mathbb{X}$ *be a compatible set and* $\mathsf{X} \in \mathscr{X}$ *a complex agent. A set* $\overline{\mathscr{X}} \subseteq \mathscr{X}$ *is called* compatible subset *of* $\mathscr{X}$ *w.r.t.* X *if the following holds:*

$$\forall \mathsf{X}' \in \overline{\mathscr{X}} : (\mathsf{X}' \lhd \mathsf{X} \wedge \forall \mathsf{X}'' \in \overline{\mathscr{X}} : \neg(\mathsf{X}'' \lhd \mathsf{X}'))$$

where $\mathsf{X}' \neq \mathsf{X}''$.

Note that for any complex agent X there exists just one compatible subset. The reason follows from Lemma 4.2.10 and the definition of the subset (Definition 4.2.11).

Remark 4.2.12 *In the process of generating reactions from the rules (Section 3.8), we are using ground form function (Definition 3.7.1). The function basically adds context to individual complex agents (and their substructures). In the definition of the function we provided exact procedure how to generate such agents.*

When we look at the process more theoretically, a set of appropriate complex agents is generated for every complex agent X *in the rule* R *and then they are connected to multiple unique reactions. This set is actually a compatible subset* $\overline{\mathscr{X}}$ *of compatible set* $\mathscr{X}$ *such that* $\mathsf{X} \in \mathscr{X}$. *In other words, agents of* $\overline{\mathscr{X}}$ *are the minimal elements of set* $\mathscr{X}$ *w.r.t. compatibility with complex agent* X.

4.2.1 Rule redundancy elimination

There might be cases where there are redundant rules defined in a model (Definition 4.2.13). This might be done for example by inattention of modeller in combination with high level of abstraction language uses. These rules do not cause any semantic difference, but can possibly slow down dynamic and/or static analysis, eventually undesirably affecting the results of analysis.

Definition 4.2.13 *Redundant rule expression*
Let $\mathbb{M}_1 = (\mathcal{R} \cup \{\rho\}, \Sigma_\mathsf{A}, \Sigma_\mathsf{T}, \mathtt{init})$ *and* $\mathbb{M}_2 = (\mathcal{R}, \Sigma_\mathsf{A}, \Sigma_\mathsf{T}, \mathtt{init})$ *be BCSL models where* ρ *is a rule expression such that* $\rho \notin \mathcal{R}$. *We define the rule expression* ρ *as* redundant *if* $LTS(\mathbb{M}_1) \equiv LTS(\mathbb{M}_2)$.

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

Theorem 4.2.14 *Let* ρ, ρ' *be two rule expressions of the form:*

$$\Gamma_1 + \Gamma_2 + \ldots + \Gamma_n \Rightarrow \Gamma_{n+1} + \Gamma_{n+2} + \ldots + \Gamma_m$$

for some $m, n \in \mathbb{N}$ *such that* $\mathsf{F}(\rho), \mathsf{F}(\rho')$ *are well-formed rules. The rule expression* ρ *is* redundant *if* $\forall i \in [1, m] : \mathsf{F}(\Gamma_i) \lhd \mathsf{F}(\Gamma'_i)$.

Proof: The problem whether the elimination of a redundant rule expression preserves semantics can be reduced to a simple question – if it holds for 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.*

The complex agents $\mathsf{X}_k = \mathsf{F}(\Gamma_k)$ *and* $\mathsf{X}'_k = \mathsf{F}(\Gamma'_k)$ *both belong to the same compatible set* $\mathscr{X}$ *since* $\mathsf{X}_k \lhd \mathsf{X}'_k$, *which follows from the condition of the theorem.*

Then, we can create subsets $\overline{\mathscr{X}}$, $\overline{\mathscr{X}'} \subseteq \mathscr{X}$ *for both complex agents respectively (Definition 4.2.11). Because the complex agents are compatible (*$\mathsf{X}_k \lhd \mathsf{X}'_k$*), the subset* $\overline{\mathscr{X}}$ *of agent* X_k *is either subset or equal to set* $\overline{\mathscr{X}'}$ *of agent* X'_k *:*

$$\overline{\mathscr{X}} \subseteq \overline{\mathscr{X}'}.$$

Applied generally on the entire rule, the produced set of reactions from the redundant rule is actually a subset of reactions produced from the non-redundant rule. ■

For the proof we used compatible sets of complex agents and the fact that when we are generating reactions from rules, we are actually enumerating all agents from the compatible set which are *compatible with* original agent in the rule (Remark 4.2.12).

Example 4.2.15 *Redundant rule expression*
Let us have two rule expressions:

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

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

Considering structure signature $\Sigma_{\mathsf{T}}(K) = \{S, T\}$ and atomic signatures $\Sigma_{\mathsf{A}}(S) = \{u, p\}$ and $\Sigma_{\mathsf{A}}(T) = \{a, i\}$, the rule expression (1) produces following set of eight reactions:

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

while the rule expression (2) produces set of four reactions which is a subset of the previous one:

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

4.2.2 Context-based reduction

There might be cases when eliminating context of the given BCSL model preserves some properties while making the analysis of the model simpler. This is particularly case of dynamic analysis, where minor change in the model specification can dramatically affect the behaviour. To make it possible, we have to define such a reduced model and show what kind of information does it preserve.

Definition 4.2.16 *Reduced model*
Let $\mathbb{M} = (\mathcal{R}, \Sigma_A, \Sigma_T, \texttt{init})$ *be a BCSL model. We define* reduced model $\widetilde{\mathbb{M}}$ *as a pair* $(\widetilde{\mathcal{R}}, \mathrm{I})$ *where* $\widetilde{\mathcal{R}}$ *is set of rule expressions and* I *is initial multiset of complex agent expressions where the following conditions hold:*

1. *for every well-formed rule* $\mathsf{F}(\rho') = (\chi', \omega', \iota', \varphi', \psi') \in \mathbb{R}$ *such that* $\rho' \in \widetilde{\mathcal{R}}$ *and appropriate original rule* $\mathsf{F}(\rho) = (\chi, \omega, \iota, \varphi, \psi) \in \mathbb{R}$ *such that* $\rho \in \mathcal{R}$ *holds:*

$$\forall i \in [1, k] : \pi_i(\chi') \equiv \mathsf{sup}(\mathscr{X})$$

   where $\mathscr{X}$ *is a compatible set such that* $\pi_i(\chi) \in \mathscr{X}$*, length* $k = |\chi'| = |\chi|$ *(i.e., the number of complex agents in both rules is the same), and* $\iota = \iota'$*;*

2. *for every complex agent* $\mathsf{X}' = \mathsf{F}(\Gamma') \in \mathrm{I}$ *and appropriate original complex agent* $\mathsf{X} = \mathsf{F}(\Gamma) \in \texttt{init}$ *the following condition holds:*

$$\mathsf{X}' \equiv \mathsf{sup}(\mathscr{X}) \text{ such that } \mathsf{X} \in \mathscr{X}.$$

Reduced model $\widetilde{\mathbb{M}}$ is created form the given BCSL model by reducing the context of complexes in the rules to maximum level. This is achieved by taking supremum from compatible set $\mathscr{X}$. This procedure might produce rules with no action, i.e. not well-formed rules – such rules are omitted. Then, only rules creating/destroying agents and complex formation/dissociation should remain, providing reduced network. Since we are reducing context, the resulting network can be equal or smaller than the original one.

Definition 4.2.17 *Compatibility of states*

Let $\mathbb{M}$ *be a BCSL model and* $\mathsf{s}_1, \mathsf{s}_2 \in S$ *of* $LTS(\mathbb{M})$ *two states from its LTS. The state* s_1 *is* compatible with *state* s_2*, written* $\mathsf{s}_1 \lhd \mathsf{s}_2$, iff

$$\forall \mathsf{X}_1 \in \mathsf{s}_1 \; \exists \mathsf{X}_2 \in \mathsf{s}_2 : \mathsf{sup}(\mathscr{X}) \equiv \mathsf{X}_2$$

such that $\mathscr{X} \subseteq \mathbb{X}$ *is a compatible set and* $\mathsf{X}_1 \in \mathscr{X}$.

Definition 4.2.18 *Over-approximation of LTS*

Let $LTS(\mathbb{M}), LTS(\mathbb{M}')$ *be labelled transition systems of some BCSL models* $\mathbb{M}, \mathbb{M}'$*. The* $LTS(\mathbb{M}')$ *is* over-approximation *of* $LTS(\mathbb{M})$ *if for every path* $\mathsf{s}'_1\mathsf{s}'_2\mathsf{s}'_3 \ldots \mathsf{s}'_n$ *in* $LTS(\mathbb{M}')$ *there exists a path* $\mathsf{s}_1\mathsf{s}_2\mathsf{s}_3 \ldots \mathsf{s}_m$ *in* $LTS(\mathbb{M})$ *such that:*

$$\forall \mathsf{s}'_i, \mathsf{s}'_{i+1} \; \exists \mathsf{s}_k, \mathsf{s}_l : (l > k \wedge \mathsf{s}_k \lhd \mathsf{s}'_i \wedge \mathsf{s}_l \lhd \mathsf{s}'_{i+1}).$$

A reduced model $\widetilde{\mathbb{M}}$ is actually an over-approximation of a BCSL model $\mathbb{M}$ in the context of their LTSs (Definition 4.2.18). It can be used for some types of analyses which avoid combinatorial explosion of the original model $\mathbb{M}$.

Theorem 4.2.19 *Let* X *be a complex agent,* $\mathscr{X}$ *be a compatible set for* X*,* $\mathbb{M}$ *be a given BCSL model, and* $\widetilde{\mathbb{M}}$ *be an appropriate reduced model of model* $\mathbb{M}$*. If supremum* $\mathsf{sup}(\mathscr{X})$ *is non-reachable in* $LTS(\widetilde{\mathbb{M}})$*, then agent* X *is also non-reachable in the* $LTS(\mathbb{M})$*.*

Proof: Let us assume a complex agent $\mathsf{sup}(\mathscr{X})$ *is non-reachable in* $LTS(\widetilde{\mathbb{M}})$*, but* $\mathsf{X} \in \mathscr{X}$ *is reachable in* $LTS(\mathbb{M})$*.*

Generally, there is a path formed from rules in the $LTS(\mathbb{M})$ *such that we transform complex agents from initial agents to desired complex agent* X*. When we move to context of* $LTS(\widetilde{\mathbb{M}})$*, there is no such path for* $\mathsf{sup}(\mathscr{X})$*. According to Definition 4.2.16, 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 original rule on a complex agent, we can do the same with reduced rule and its supremum. It follows there must exist such path also in* $LTS(\widetilde{\mathbb{M}})$ *and the complex agent* $\mathsf{sup}(\mathscr{X})$ *is reachable, what is a contradiction.* ■

If we are checking whether an agent is reachable in $LTS(\mathbb{M})$ for given model $\mathbb{M}$, we might first check whether its super-ancestor is reachable in $LTS(\widetilde{\mathbb{M}})$ of appropriate reduced model $\widetilde{\mathbb{M}}$. If it holds, then we are still not certain about reachability of agent itself and it has to be checked in the $LTS(\mathbb{M})$. However, agent which is not reachable in $LTS(\widetilde{\mathbb{M}})$ is also not reachable in $LTS(\mathbb{M})$ (Theorem 4.2.19).

4.2.3 Static reachability analysis

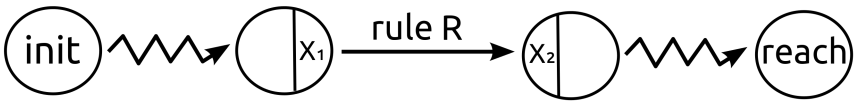
Since we have defined the compatibility operator for agents, we can apply static reachability analysis before enumerating the entire transition system of the model $\mathbb{M}$. This analysis has its limitations but can serve for checking whether a complex agent X is *non-reachable* (similar to Section 4.2.2).

Theorem 4.2.20 *Let $\mathcal{R}_f$ be a set of rules created from a set of rule expressions $\mathcal{R}$ of given model $\mathbb{M}$ using semantic function F. Let X be a complex agent. The complex agent X is non-reachable w.r.t. set of rules $\mathcal{R}_f$ if the following holds:*

$$\forall\, \mathsf{R} \in \mathcal{R}_f\ \forall i \in [\iota+1,\ldots,|\chi|] : \neg(\pi_i(\chi) \lhd \mathsf{X})$$

such that $\mathsf{R} = (\chi, \omega, \iota, \varphi, \psi)$.

Proof: Let us assume we have a path of states constructed by applying corresponding reactions created from rules $\mathcal{R}_f$ where X is reachable. At some point on the path, we inevitably have to create a complex agent $\mathsf{X}_2 \lhd \mathsf{X}$ from a complex agent X_1 applying a rule R.



It requires there has to be a complex agent X_2' in the rule which is compatible with the complex agent X_2. If there is not such an agent, the agent X is non-reachable. ■

Theorem 4.2.20 states a very powerful result in context of non-reachability analysis. Compared to dynamic non-reachability analysis, it completely avoids any combinatorial explosion and gives answer only by testing structural properties of rules.

This procedure can be extended for checking reachability of multiple complex agents. However, it does not consider stoichiometry.

4.2.4 Automatic synthesis of signatures

The BCSL model definition as given in Section 3.4 can be reduced by signatures. Note we do not mean extended signatures by complex names (Section 3.11.2) – just the basic signatures for atomic and structure agents. These can be obtained statically from the rules (and the initial state) in a linear process. The process comprises these steps:

1. prepare empty signature sets Σ_{A} and Σ_{T},
2. create rule $\mathsf{R} = (\chi, \omega, \iota, \varphi, \psi)$ for every rule expression ρ using semantic function F,
3. iterate through all $\omega \in \mathsf{R}$ and:
   (a) for every atomic agent A do the following:
      i. **if** $\exists x \in \Sigma_{\mathsf{A}} : \eta(\mathsf{A}) \equiv \pi_1(x)$
      then $\delta(\mathsf{A}) \mapsto \pi_2(y)$
      else $(\eta(\mathsf{A}), \{\ \delta(\mathsf{A})\ \}) \mapsto \Sigma_{\mathsf{A}}$;
   (b) for every structure agent T do the following:
      i. **if** $\exists x \in \Sigma_{\mathsf{T}} : \eta(\mathsf{T}) \equiv \pi_1(x)$
      then $\forall \mathsf{A} \in \gamma(\mathsf{T}) : \eta(\mathsf{A}) \mapsto \pi_2(y)$ and
      do step 3a with every atomic agent A
      else $(\eta(\mathsf{T}), \{\eta(\mathsf{A}) \mid \mathsf{A} \in \gamma(\mathsf{T})\}) \mapsto \Sigma_{\mathsf{T}}$
      do step 3a with every atomic agent A.

The described process basically iterates through all rules and collects all atomic names and states associated with them. Similarly, it collects all structure names and appropriate names of atomic agents used in their partial compositions. In this case, atomic information has to be collected recursively because some atomic agents might be used only in partial composition, not independently in a complex.

We can discuss the fact that desired signature might be in some manner *larger* than the collected one. However, for example, if we

want to have more states defined for an atomic agent, they should be definitely used somewhere in the model. In other case, it does not play any role in the model behaviour and therefore its definition is senseless. In other words, the described procedure guarantees collecting all signature details which play any role in the model.

4.2.5 Translation to Petri Nets

By generating the reaction-based model from the rule-based, which is always possible (Section 3.8), we can furthermore translate such model to a Petri Net [30].

The translation can be summarised in the following steps:

1. since BCSL rules produce a finite number of reactions, we have also a finite number of unique agents – these are the places P,
2. the transitions T are the reactions,
3. set of arcs F is created from all reactions as following:
   - place input arc from each place which has label in left-hand side of the reaction (with appropriate multiplicity),
   - do the same for output arcs for right-hand side of the reaction,
4. create initial marking m from given initial state of the model by similar procedure described above.

Once the model is translated, we are allowed to apply all available types of analyses which we can perform on Petri Nets. For example, P and T invariants, deadlock detection, coverability etc. (more details in [30]).

4.2.6 Minimal optimal bound of the system

Transition systems produced by semantics of BCSL models are infinite in general. Despite the fact that the underlying reaction networks are always finite (Section 4.2.6), there are types of reactions

which can increase number of occurrences in a state with no limitations. Typically, such reactions form an interface between the modelled system and the environment.

However, the process of applying vector reactions on states can be extended by a control system. This system does not allow to grow individual values inside the state vector above given *bound* B. Basically, if the produced vector should obtain an integer higher than the bound, the action is not taken. This procedure is very similar to checking whether all values in the vector are positive. Instead of only the lower bound, we have two bounds $< 0, B >$.

The remaining issue is what is an optimal value for the bound. The universal solution is to let the modeller decide. It follows extension of model definition by the bound. However, this process requires the user to know additional information about the model and additional effort. It does not fit the purposes of the language.

Therefore we decided to apply a static analysis on the model. We will not provide a formal definition of this analysis, rather give intuition and reason about this particular solution.

We decided to find a minimal bound, which is in this context optimal for generating all required objects. It means that all the complexes which are potentially encoded in rules will be produced in at least one repetition. In other words, with this bound a minimal network should be created. In such network, every agent reachable in boundless network is reachable also in minimal network (which does not hold when we take stoichiometry into the account).

To actually calculate the minimal bound we take the maximum value from the following measurements:

1. count the highest number of occurrences from all complex agents in initial state,

2. count the highest number of unique atomic and structure agents in every complex agent of all rules,

3. count the highest stoichiometry in every left and right-hand side of the rules.

The step (1) ensures the bound B will not be lower than highest number of equivalent agents in the initial state. In the step (2) we

count repetitions of equivalent atomic and structure agents inside all complexes, which ensures that there will be enough agents for creation of every complex agent. The last step (3) takes into account the stoichiometry used in context of complex agents themselves.

This should guarantee that it is possible to achieve high enough number of occurrences to produce any complex or combination of complexes.

5 Case study

E-cyanobacterium.org [34] is an online platform providing tools for public sharing, annotation, analysis, and visualisation of dynamical models and wet-lab experiments related to cyanobacteria. The general aim is to stimulate collaboration between experimental and computational systems biologists in order to achieve better understanding of cyanobacteria. As an instance of CMP, it covers all the features of the platform.

The following processes of cyanobacteria are covered in BCS: environmental processes, respiration, photosynthesis, carbon concentrating mechanism, circadian clock, and metabolism. Currently, the repository of models contains two models describing circadian clock (Miyoshi et al. 2007 [28], Hertel et al. 2013 [16]) and a kinetic model of metabolism (Jablonsky et al. 2014 [21]). Two other models present unpublished results – dynamics of carbon fluxes (Müller et al.) and photosynthesis (Plyusnina et al.). Additionally, there are two models in development describing carbon concentrating mechanism (Clark et al. 2014 [4] and Fridlyand et al. 1995 [12]). Each of them was properly mapped on the BCS and helped to form suitable space covering all the mentioned processes.

Entire biochemical space of cyanobacteria is formed from over 1000 entities and 600 rules. Therefore, we will not explain all of them here, we provide a several cases which demonstrate how different BCSL features can be used for describing particular biological problems.

The circadian clock of cyanobacteria [14] is formed by three proteins $KaiA$, $KaiB$, and $KaiC$ (with additional presence of ATP). The key role is played by $KaiC$. Compared to the other two proteins, it is physically the largest one. Its primary structure is a sequence of amino acids, where two particular are important for the modelling – threonine 432 and serine 431. These two amino acids can be enriched by phosphoryl group in process commonly known as phosphorylation [7]. However, the sites are accessible only when the protein forms a homohexamer – a structure of six $KaiC$ proteins. For the phosphorylation $KaiA$ proteins are responsible while $KaiB$ proteins inactivate $KaiA$s, which leads to autodephosphorylation of

KaiC proteins. There are several complexes which can be formed from mentioned proteins according to the current situation with the *KaiC* hexamer (Figure 5.1).

It is important to note the *KaiA* and *KaiB* proteins only *enhance* (resp. *suppress*) the phosphorylation process, but do not enable (resp. disable) it completely. When it comes to quantitative description of the system, it is necessary to depict all possible actions. Particularly, we need to express the fact that the phosphorylation process happens regardless of the particular complex *KaiC* protein is currently in.

$$S\{u\} :: KaiC :: ?X :: cyt \Leftrightarrow S\{p\} :: KaiC :: ?X :: cyt \ ;$$
$$?X = \{\ KaiC6, KaiA2C6, KaiA4C6, KaiB6C6, KaiA4B6C6, KaiA6B6C6\ \}$$

Figure 5.1: Phosphorylation rule of serine residue for all possible complexes with *KaiC* included. A variable $?X$ is used in place of all possible complexes. Notice there is also a *KaiC*6 complex, which corresponds to auto-(de)phosphorylation.

Another interesting aspect of the cyanobacterial circadian clock mechanism is the formation of *KaiC* hexamers. The condition that the *KaiC* protein can be (de)phosphorylated *only* inside a hexamer leads to one important fact – it does not mean the hexamer must be assembled from (resp. dissociated to) unphosphorylated *KaiC* proteins. The hexamer can be dissociated at any point in time, for example when three *KaiC*s are phosphorylated on the serine residue, other two are phosphorylated on both residues, and the last one is not phosphorylated at all. The number of possible combinations is quite huge.

$$6\ KaiC :: cyt \Leftrightarrow KaiC6 :: cyt$$

Figure 5.2: Formation of KaiC hexamer from six KaiC proteins and its dissociation. The rule requires to have a defined complex name *KaiC*6 in the corresponding BCS. The complex can be formed (resp. dissolved) regardless of the context of individual *KaiC* proteins.

Rule-based languages including BCSL are very suitable for such cases. What we actually need to express is the situation when a com-

plex is formed (or dissolved), regardless of the context (particular states) of the interacting entities (Figure 5.2).

Another important process inside cyanobacteria is photosynthesis. It is performed on internal membranes called thylakoids. In general, photosynthesis in cyanobacteria uses water as an electron donor and produces oxygen as a product. Electron transport is performed through multiple protein complexes, which together form the thylakoid membrane itself. Such complexes are for example photosystem I and II (Figure 5.3). However, direct modelling of the complexes is not very efficient. The typical rules in photosynthesis are just electron transfers between the complexes or reduction/oxidation of individual parts. Therefore, such action usually happen regardless of the context of the rest of the complexes in a photosystem. The best approach with respect to BCSL language is to use structure agent for defining the protein complexes and its individual parts model as partial composition.

$$ps2(oec\{2+\}, yz\{+\}) :: tlm \Leftrightarrow ps2(oec\{3+\}, yz\{n\}) :: tlm$$

Figure 5.3: Oxidation of S2-state of the oxygen-evolving complex $oec\{2+\}$ by $Yz\{+\}$ in photosystem II. There are many more subparts of photosystem II. (e.g. chlorophyl, pheophytin), but their states are not important in this case.

We want to demonstrate practical purposes of a few types of static analyses defined in Section 4.2. Yamada et al. 2004 [35] 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 if something goes wrong on a particular point in the cascade, it will influence the rest of the pathway. Entire model written in BCSL syntax is provided in Appendix B. It consists of 20 entities interacting in 56 rules. Most of proteins can undergo phosphorylation (state change from u to p on some amino acid residues). We consider initial conditions such that there are all required entities in one or two repetitions (some complexes require multiplicity). In such case, the number of reachable state grows up to 2^{72}, which too high to be effectively enumerated.

In Figure 5.4, there is the model defined in BCSL syntax. Note this model is a fragment of the entire model. The rule (4) 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.

For example, we want to check whether a given complex agent $FRS(Thr\{u\}, Tyr\{u\}).FGF(Thr\{u\}).R.FGF(Thr\{u\}).R :: cyt$ is reachable for the given model. The agent is formed from FGF proteins which are unphosphorylated (u) on threonine residues (Thr). With the traditional approach, we have to enumerate entire transition system of the model and then use dynamic model checking method to check it. In our case, we can use Theorem 4.2.20 and check if it is non-reachable using static reachability analysis. The conclusion is that there is no compatible agent on any right-hand side of the rules. It follows the complex agent is non-reachable.

1. $FGF :: cyt + R :: cyt \Leftrightarrow FGF.R :: cyt$
2. $2\ FGF.R :: cyt \Leftrightarrow FGF.R.FGF.R :: cyt$
3. $FGF(Thr\{u\}).R.FGF.R :: cyt \Leftrightarrow FGF(Thr\{p\}).R.FGF.R :: cyt$
4. $FRS(Thr\{u\}) :: cyt + FGF(Thr\{p\}).R.FGF(Thr\{p\}).R :: cyt \Rightarrow$
   $\Rightarrow FRS(Thr\{u\}).FGF(Thr\{p\}).R.FGF(Thr\{p\}).R :: cyt$
5. $FRS(Thr\{u\}).FGF.R.FGF.R :: cyt \Rightarrow FRS(Thr\{p\}).FGF.R.FGF.R :: cyt$

$\vdots$

Initial conditions:
$2\ FGF(Thr\{u\}) :: cyt$
$2\ R :: cyt$
$1\ FRS(Thr\{u\}, Tyr\{u\}) :: cyt$

$\vdots$

Figure 5.4: A fragment of the model Yamada et al. 2004 [35] of FGF signalling pathway written in BCSL. The complete model can be seen in Appendix B.

Demonstration of context-based reduction (Section 4.2.2) is provided on the same model as in the previous case. Now we can compute with the entire model since we will reduce its context to minimum. In Figure 5.5 there are 16 bidirectional rules which we have created by applying Definition 4.2.16. The size of transition system has significantly decreased – it has 606 states and 2015 edges.

For the analysis itself, we could, for example, ask about the reachability of a complex agent $Raf(Thr\{p\}).ERK(Tyr\{p\}, Thr\{p\}) :: cyt$ in the original model (Appendix B). We can first check whether its supremum (from the corresponding compatible set $\mathscr{X}$) $Raf.ERK :: cyt$ is non-reachable in the reduced model. Since the transition system of the model is relatively small, it is not a problem using dynamical model checking. Using Theorem 4.2.19, the answer is non-reachable.

$$
\begin{array}{rcll}
FGF :: cyt & + & R :: cyt & \Leftrightarrow \quad FGF.R :: cyt \\
FGF.R :: cyt & + & FGF.R :: cyt & \Leftrightarrow \quad FGF.R.FGF.R :: cyt \\
FGF.R.FGF.R :: cyt & + & FRS :: cyt & \Leftrightarrow \quad FGF.R.FGF.R.FRS :: cyt \\
FRS :: cyt & + & SHP :: cyt & \Leftrightarrow \quad FRS.SHP :: cyt \\
GS :: cyt & + & GPP :: cyt & \Leftrightarrow \quad GS.GPP :: cyt \\
GS :: cyt & + & ERK :: cyt & \Leftrightarrow \quad GS.ERK :: cyt \\
FRS :: cyt & + & GS :: cyt & \Leftrightarrow \quad FRS.GS :: cyt \\
FRS.GS :: cyt & + & Ras :: cyt & \Leftrightarrow \quad FRS.GS.Ras :: cyt \\
GAP :: cyt & + & Ras :: cyt & \Leftrightarrow \quad GAP.Ras :: cyt \\
Ras :: cyt & + & Raf :: cyt & \Leftrightarrow \quad Ras.Raf :: cyt \\
PP :: cyt & + & Raf :: cyt & \Leftrightarrow \quad PP.Raf :: cyt \\
Raf :: cyt & + & MEK :: cyt & \Leftrightarrow \quad Raf.MEK :: cyt \\
XPP :: cyt & + & MEK :: cyt & \Leftrightarrow \quad XPP.MEK :: cyt \\
MEK :: cyt & + & ERK :: cyt & \Leftrightarrow \quad MEK.ERK :: cyt \\
MKP :: cyt & + & ERK :: cyt & \Leftrightarrow \quad MKP.ERK :: cyt \\
ERK :: cyt & + & FRS :: cyt & \Leftrightarrow \quad ERK.FRS :: cyt
\end{array}
$$

Figure 5.5: Model Yamada et al. 2004 [35] with reduced context according to Definition 4.2.16.

6 Implementation

In order to support usability of the BCSL language, we developed a prototype software tool BCSgen. The general goal of the tool is to provide environment suitable for editing and maintenance of the BCSL models and provide functionality to analyse the models. The software was developed in several iterations. At first as a project MUNI33/092015 in *Dean's Program of the Faculty of Informatics MU for support of student research and development projects*, later as a part of the project MUNI33/062017 in the same program. It was further extended to the current stage in this thesis.

The tool is developed in Python language (version 2.7) as open-source desktop software under GPL-3.0 license. The source code, a short tutorial for usage and installation, and current stable version are available on GitHub[1].

In the first version, there was no support for editing the models. The only functionality provided by the tool was import of the model from a text file (in a predefined format) and enumerating entire transition system of the model. The implementation was done by translating to Kappa and it was rather slow and non-efficient. The single transaction (application of one rule) was delayed by translation to Kappa and then by exact matching and replacement, which might get quite complicated. However, it was a good start for the language and it was possible to analyse exported networks in external tools. There was a simple GUI made in Tkinter [15], which had several issues. The most important one was the fact that Tkinter was not designed for multithreading programming. For example, while computing a transition system for the model, the whole GUI was not responsible.

In the second version, we focused on speed improvement and development of a user-friendly GUI. For the speed improvement, we implemented semantics as defined in this thesis (Section 3.6.1). Compared to previous version, in this case there was no exact rule-based execution of the rule – we rather translated the rules to the reactions and subsequently encoded as vector operations over integers. Therefore, one transaction was just addition of one vector (left-

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

hand side) and subtraction of another one (right-hand side). The GUI was reimplemented in PyQt4 [33], which naturally works with multithreading. It was significant improvement from the performance point of the view. Moreover, we improved input/output formats. For input we defined BCSL model format, which serves for import. As output format we have chosen JSON [2], which is very suitable for its simplicity and support by many external tools.

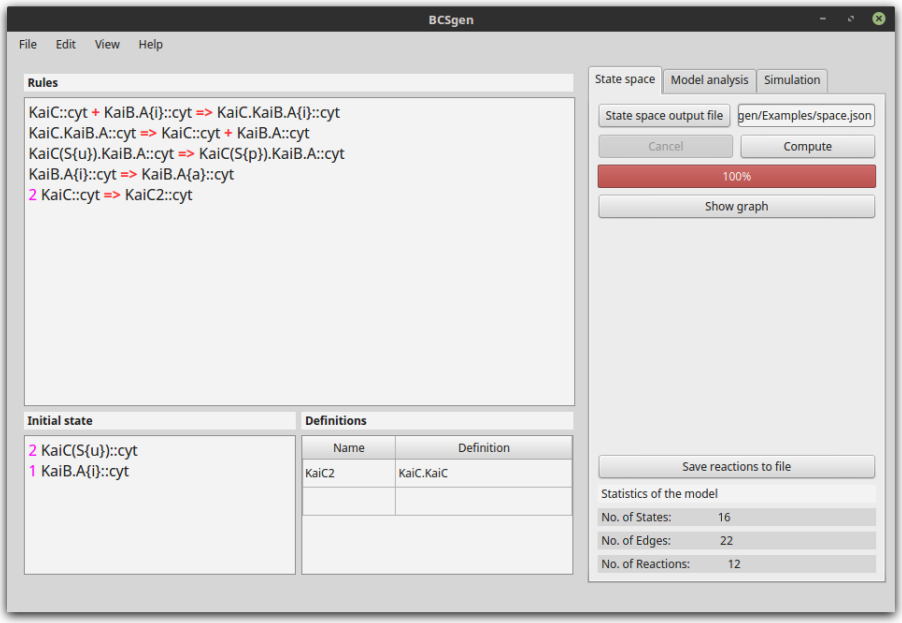


Figure 6.1: A screenshot of Graphical User Interface of BCSgen tool. There is a field for editing rules with integrated rule parser, field for initial state, and a table for defining custom names for complexes. On the right, there are options to analyse the model and display visualisations.

In the last version, which is part of this thesis, we focused on improvement of models maintenance, additional analysis services, and visualisation. We have integrated an editor which is able to highlight syntax errors while building the model (Figure 6.1). To achieve this, we needed an efficient rule-parser which would be able to parse the rules very fast. The parser is able to parse even syntactic extensions defined in Section 3.11. The rules engine then translates the rule to

ground form and creates reactions (in case there are no errors in the model). Since this engine runs in the background, user does not notice any decrease in performance. When the reactions are computed, it is indicated by the possibility to export them and generate transition system of the model.

From dynamical analysis techniques, we implemented model simulation and reachability analysis. For the model simulation, we allow stochastic or deterministic approach (Figure 6.2). In the first case, random rules are fired according to Poisson distribution while in the latter average trace is computed by solving ODEs build from the reactions. For the stochastic case, there are options to choose number of runs and apply interpolation on the resulting time series chart in order to achieve smoother results. It is important to note that only models with defined rates can be simulated. For simplicity, approach with only fully specified agents is used.

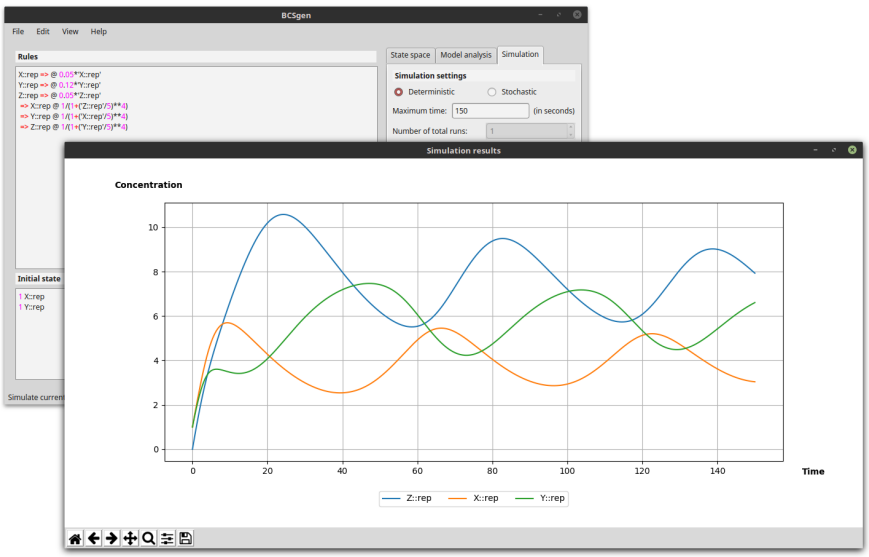


Figure 6.2: Example of a deterministic simulation for simple repressilator model. For the simulation to be enabled, the rates for individual rules have to be defined. In the right panel, deterministic approach and the maximum simulation time are chosen.

Since there are some graphs and charts generated in BCSgen, we decided to integrate visualisation tools into it. We used two libraries

– vis.js [1] for interactive networks visualisation and MatPlotLib [20], a Python library suitable for visualisation of time series.

Vis.js is used for exploration of transition system with highlighting the chosen node and announcing content of the node. It is possible to display paths from the initial state to other states, which satisfy reachability condition given in reachability analysis part (Figure 6.3). This is very demonstrative and suitable for educational purposes. However, such visualisation has its limitations. For example, huge graphs are very difficult to display – on the other hand, for a high number of nodes it is not even useful.

The MatPlotLib provides a visualisation of simulation results. It possible to zoom the chart, hide/show individual curves, and export the current view as a picture.

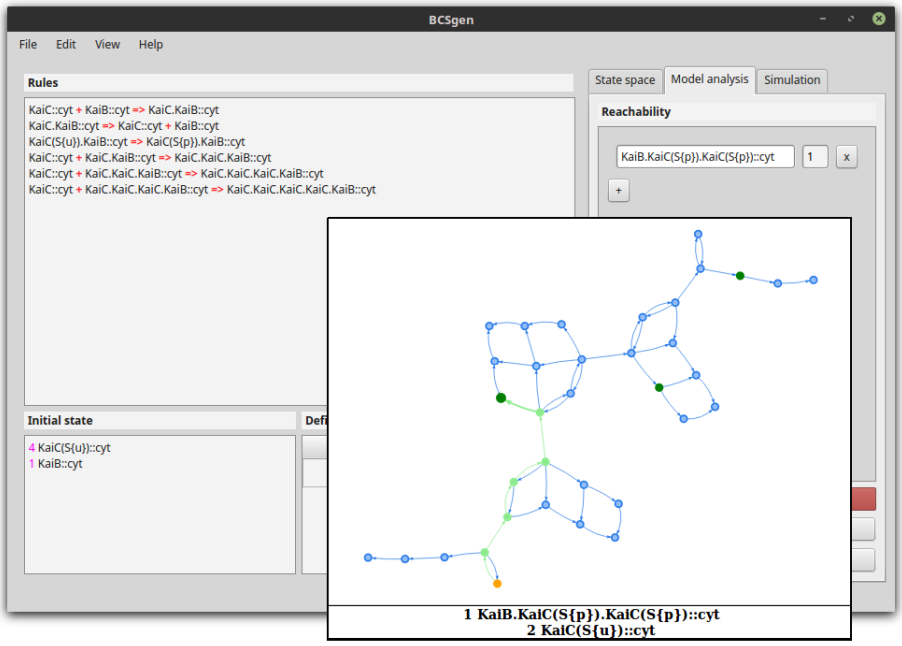


Figure 6.3: A graph showing LTS of given model. There are a few types of vertices in the graph: **orange** – the initial state; **dark green** – states satisfying condition given in right panel of the GUI; **bright green** – path from initial to chosen reachable state.

7 Conclusions

In the first part of this thesis, we have explained the background of the work. Particularly, we provided explanation of the principles of rule-based modelling using schematic examples. Several existing rule-based languages have been introduced, together with their key features. Comprehensive Modelling Platform as a framework for the modelling biochemical processes has been described with all its important modules such as Model repository and Biochemical Space.

In the second part, we have formally defined Biochemical Space Language as a description format for Biochemical Space. Together with very concise and *human-readable* syntax we have defined operational semantics. An indirect approach has been chosen for practical purposes. Moreover, several syntactic extensions were defined to increase readability and maintenance of the language.

In the third part, we have introduced analysing approaches which could be directly used on models defined using the language. Those include dynamic analysis techniques such as model checking and simulation. We have also defined a few types of static analysis techniques which can be used without explicitly enumerating entire behaviour of the model.

The last part was dedicated to practical usage of the language and provided analysis formalisms. Namely, in the case study we have demonstrated how can be abstractions of the language used in particular modelling situations. We have also shown practical impact of defined static analysis techniques on investigating behaviour of the model effectively. Finally, we have presented implementation of a software tool BCSgen, which was developed in order to provide support for maintenance and analysis of the models.

As a future work, we want to improve specification of kinetic rates for the BCSL rules. In the current stage, we allow to reference only fully defined objects in the kinetic rates. This is a very limiting restriction, because it would be natural to change context in rate and the rule simultaneously. Also we want to develop some additional static analysis techniques, since the capabilities of the language are wide in this area. Particularly, we would like to define some properties solving reachability analysis instead non-reachability.

The next goal is to make some of the static analysis techniques more general. Particularly, some of the non-reachability techniques were defined only for a single complex agent. We want to promote these techniques in order to be applied on sets of agents.

Finally, as a future work for the tool BCSgen we are planning to reimplement it to a web-based version. It should be integrated in Comprehensive Modelling Platform (Section 1.4) to allow direct analysis of the models and appropriate parts of Biochemical Space.

Bibliography

[1] B. V. Almende. vis.js – a Dynamic, Browser-based Visualization Library. 2016.

[2] P. C. Bryan and K. Zyp. The JSON Data Interchange Format. *ECMA International*, 404, 2013.

[3] G. Chartrand. Connected Graphs. *Introductory Graph Theory*, pages 41 – 45, 1985.

[4] R. L. Clark, J. C. Cameron, T. W. Root, and B. F. Pfleger. Insights into the Industrial Growth of Cyanobacteria from a Model of the Carbon-Concentrating Mechanism. *AIChE Journal*, 60:1269 – 1277, 2014.

[5] E. M Clarke. *Model checking*. MIT press, 1999.

[6] E. A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations*. Tata McGraw-Hill Education, 1955.

[7] P. Cohen. The Origins of Protein Phosphorylation. *Nature cell biology*, 4:127 – 130, 2002.

[8] V. Danos and C. Laneve. Formal Molecular Biology. *Theoretical Computer Science*, 325:69 – 110, 2004.

[9] P. De Matos, R. Alcántara, A. Dekker, M. Ennis, J. Hastings, K. Haug, I. Spiteri, S. Turner, and C. Steinbeck. Chemical Entities of Biological Interest: an update. *Nucleic Acids Research*, 38:249 – 254, 2010.

[10] T. Děd, D. Šafránek, M. Troják, M. Klement, J. Šalagovič, and L. Brim. Formal Biochemical Space with Semantics in Kappa and BNGL. *Electronic Notes in Theoretical Computer Science*, 326:27 – 49, 2016.

[11] J. R. Faeder, M. L. Blinov, and W. S. Hlavacek. Rule-based Modeling of Biochemical Systems with BioNetGen. *Systems biology*, pages 113 – 167, 2009.

[12] L. Fridlyand, A. Kaplan, and L. Reinhold. Quantitative Evaluation of the Role of a Putative CO_2-scavenging Entity in the Cyanobacterial CO_2-concentrating Mechanism. *Biosystems*, 37:229 – 238, 1996.

[13] D. T. Gillespie. A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *Journal of Computational Physics*, 22:403 – 434, 1976.

[14] S. S. Golden, M. Ishiura, C. H. Johnson, and T. Kondo. Cyanobacterial Circadian Rhythms. *Annual Review of Plant Physiology and Plant Molecular Biology*, 48:327 – 354, 1997.

[15] J. E. Grayson. *Python and Tkinter Programming*, volume 140. Manning Greenwich, 2000.

[16] S. Hertel, C. Brettschneider, and I. M. Axmann. Revealing a Two-Loop Transcriptional Feedback Mechanism in the Cyanobacterial Circadian Clock. *PLoS Computational Biology*, 9:1 – 16, 2013.

[17] D. J. Higham. Modeling and Simulating Chemical Reactions. *SIAM review*, 50:347–368, 2008.

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

[19] M. Hucka et al. The Systems Biology Markup Language (SBML): a Medium for Representation and Exchange of Biochemical Network Models. *Bioinformatics*, 19:524 – 531, 2003.

[20] J. D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing In Science & Engineering*, 9:90 – 95, 2007.

[21] J. Jablonsky, D. Schwarz, and M. Hagemann. Multi-Level Kinetic Model Explaining Diverse Roles of Isozymes in Prokaryotes. *PLoS ONE*, 9:1 – 8, 2014.

[22] N. Juty, N. Le Novère, and C. Laibe. Identifiers.org and MIRIAM Registry: Community Resources to Provide Persistent Identification. *Nucleic acids research*, 40:580 – 586, 2011.

[23] M. Kanehisa, Y. Sato, M. Kawashima, M. Furumichi, and M. Tanabe. KEGG as a Reference Resource for Gene and Protein Annotation. *Nucleic Acids Research*, 44:457 – 462, 2016.

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

[25] M. Klement, D. Šafránek, T. Děd, A. Pejznoch, L. Nedbal, R. Steuer, J. Červený, and S. Müller. A Comprehensive Web-based Platform for Domain-specific Biological Models. *Electronic Notes in Theoretical Computer Science*, 299:61 – 67, 2013.

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

[27] A. D. McNaught. *Compendium of Chemical Terminology*, volume 1669. Blackwell Science Oxford, 1997.

[28] F. Miyoshi, Y. Nakayama, K. Kaizu, H. Iwasaki, and M. Tomita. A Mathematical Model for the Kai-Protein-Based Chemical Oscillator and Clock Gene Expression Rhythms in Cyanobacteria. *Journal of Biological Rhythms*, 22:69 – 80, 2007.

[29] M. Nakao, S. Okamoto, M. Kohara, T. Fujishiro, T. Fujisawa, S. Sato, S. Tabata, T. Kaneko, and Y. Nakamura. Cyanobase: the Cyanobacteria Genome Database update 2010. *Nucleic Acids Research*, 38:379 – 381, 2010.

[30] C. A. Petri. Petri net. *Scholarpedia*, 3, 2008.

[31] D. Poole and A. E. Raftery. Inference for Deterministic Simulation Models: The Bayesian Melding Approach. *Journal of the American Statistical Association*, 95:1244 – 1255, 2000.

[32] M. W. Sneddon, J. R. Faeder, and T. Emonet. Efficient Modeling, Simulation and Coarse-graining of Biological Complexity with NFsim. *Nature methods*, 8:177 – 183, 2011.

[33] M. Summerfield. *Rapid GUI Programming with Python and Qt: the Definitive Guide to PyQt Programming*. Pearson Education, 2007.

[34] M. Troják, D. Šafránek, J. Hrabec, J. Šalagovič, F. Romanovská, and J. Červený. E-cyanobacterium.org: A Web-Based Platform for Systems Biology of Cyanobacteria. In *Proceedings of International Conference on Computational Methods in Systems Biology (CMSB 2016)*, volume 9859 of *LNBI*, pages 316 – 322. Springer, 2016.

[35] S. Yamada, T. Taketomi, and A. Yoshimura. Model Analysis of Difference Between EGF Pathway and FGF Pathway. *Biochemical and biophysical research communications*, 314:1113–1120, 2004.

[36] F. Zhang and M. Meier-Schellersheim. Multistate, Multicomponent and Multicompartment Species Package for SBML Level 3. *COMBINE specifications*, 2017.

Appendices

A List of used notation

$\sigma(\vec{v})$	set of permutations of vector $\vec{v}$ of length $\lvert\vec{v}\rvert$
$\pi_i(\vec{v})$	projection on dimension i of vector $\vec{v}$
$X + Y$	tuples concatenation
$+_{i=1}^{n} T_i$	generalised tuples concatenation
$\Xi(Y)$	reassembly of tuples
Σ_{A}	atomic signature
Σ_{T}	structure signature
A	atomic agent
η	name
δ	state
ε	empty state (not specified)
$\mathbb{A}$	universe of all possible atomic agents
T	structure agent
η	name
γ	partial composition
$\mathbb{T}$	universe of all possible structure agents
X	complex agent
μ	sequence of agents
$\mathtt{com}$	compartment
$\mathbb{X}$	universe of all possible complex agents
$\mathbb{U}$	universe of all possible agents
R	rule
χ	sequence of complexes
ω	sequence of atomic and structure agents
ι	index determining start of right-hand side
φ	index map between ω and χ
ψ	is an index map between agents from left- and right-hand side
$\mathbb{R}$	universe of all possible rules
r	reaction
$LHS(\mathrm{r})$	left-hand side of the reaction
$RHS(\mathrm{r})$	right-hand side of the reaction

α	atomic expression
τ	structure expression
Γ	complex expression
ρ	rule expression
β	atomic or structure expression
$\mathbb{M}$	BCSL model
$\mathcal{R}$	set of rule expressions
`init`	initial multiset of complex expressions
$\gamma \ominus \gamma'$	difference of partial compositions
$\Sigma_T(\eta(\mathsf{T})) \ominus \gamma(\mathsf{T})$	difference of partial composition and structure signature
F	semantic function
$\mathcal{G}$	ground form function
$\mathcal{M}$	Chemical reaction model
$\mathfrak{R}$	set of chemical reactions
$\vec{v}$	initial vector
θ	vector of reference complexes
ζ	chemical reaction application
S	a multiset of agents
$\lambda(\mathsf{S}, \theta)$	multiset to vector translation

B Model Yamada et al. 2004

$$
\begin{aligned}
FGF :: cyt + R :: cyt &\Leftrightarrow FGF.R :: cyt \\
2\ FGF.R :: cyt &\Leftrightarrow FGF.R.FGF.R :: cyt \\
FGF(Thr\{u\}).R.FGF.R :: cyt &\Leftrightarrow FGF(Thr\{p\}).R.FGF.R :: cyt \\
FRS(Thr\{u\}).FGF.R.FGF.R :: cyt &\Rightarrow FRS(Thr\{p\}).FGF.R.FGF.R :: cyt \\
FRS(Thr\{p\}).FGF.R.FGF.R :: cyt &\Rightarrow FRS(Thr\{p\}) :: cyt + FGF.R.FGF.R :: cyt \\
SHP :: cyt + FRS(Thr\{p\}) :: cyt &\Rightarrow SHP.FRS(Thr\{p\}) :: cyt \\
FRS(Thr\{p\}).SHP :: cyt &\Rightarrow FRS(Thr\{u\}).SHP :: cyt \\
FRS(Thr\{u\}).SHP :: cyt &\Rightarrow FRS(Thr\{u\}) :: cyt + SHP :: cyt \\
GPP :: cyt + GS(Thr\{p\}) :: cyt &\Rightarrow GPP.GS(Thr\{p\}) :: cyt \\
GS(Thr\{p\}).GPP :: cyt &\Rightarrow GS(Thr\{u\}).GPP :: cyt \\
GS(Thr\{u\}).GPP :: cyt &\Rightarrow GS(Thr\{u\}) :: cyt + GPP :: cyt \\
ERK(Tyr\{p\}, Thr\{p\}) :: cyt + GS(Thr\{u\}) :: cyt &\Rightarrow ERK(Tyr\{p\}, Thr\{p\}).GS(Thr\{u\}) :: cyt \\
GS(Thr\{u\}).ERK :: cyt &\Rightarrow GS(Thr\{p\}).ERK :: cyt \\
GS(Thr\{p\}).ERK :: cyt &\Rightarrow GS(Thr\{p\}) :: cyt + ERK :: cyt \\
FRS(Thr\{p\}, Tyr\{u\}) :: cyt + GS(Thr\{u\}) :: cyt &\Leftrightarrow FRS(Thr\{p\}, Tyr\{u\}).GS(Thr\{u\}) :: cyt \\
Ras(Thr\{u\}).FRS.GS :: cyt &\Rightarrow Ras(Thr\{p\}).FRS.GS :: cyt \\
Ras(Thr\{p\}).FRS.GS :: cyt &\Rightarrow Ras(Thr\{p\}) :: cyt + FRS.GS :: cyt \\
GAP :: cyt + Ras(Thr\{p\}) :: cyt &\Rightarrow GAP.Ras(Thr\{p\}) :: cyt \\
Ras(Thr\{p\}).GAP :: cyt &\Rightarrow Ras(Thr\{u\}).GAP :: cyt \\
Ras(Thr\{u\}).GAP :: cyt &\Rightarrow Ras(Thr\{u\}) :: cyt + GAP :: cyt \\
Ras(Thr\{p\}) :: cyt + Raf(Thr\{u\}) :: cyt &\Rightarrow Ras(Thr\{p\}).Raf(Thr\{u\}) :: cyt \\
Raf(Thr\{u\}).Ras :: cyt &\Rightarrow Raf(Thr\{p\}).Ras :: cyt \\
Raf(Thr\{p\}).Ras :: cyt &\Rightarrow Raf(Thr\{p\}) :: cyt + Ras :: cyt \\
PP :: cyt + Raf(Thr\{p\}) :: cyt &\Rightarrow PP.Raf(Thr\{p\}) :: cyt \\
Raf(Thr\{p\}).PP :: cyt &\Rightarrow Raf(Thr\{u\}).PP :: cyt \\
Raf(Thr\{u\}).PP :: cyt &\Rightarrow Raf(Thr\{u\}) :: cyt + PP :: cyt \\
Raf(Thr\{p\}) :: cyt + MEK(Ser212\{u\}) :: cyt &\Rightarrow Raf(Thr\{p\}).MEK(Ser212\{u\}) :: cyt \\
MEK(Ser212\{u\}).Raf :: cyt &\Rightarrow MEK(Ser212\{p\}).Raf :: cyt \\
MEK(Ser212\{p\}).Raf :: cyt &\Rightarrow MEK(Ser212\{p\}) :: cyt + Raf :: cyt \\
Raf(Thr\{p\}) :: cyt + MEK(Ser298\{u\}) :: cyt &\Rightarrow Raf(Thr\{p\}).MEK(Ser298\{u\}) :: cyt \\
MEK(Ser298\{u\}).Raf :: cyt &\Rightarrow MEK(Ser298\{p\}).Raf :: cyt \\
MEK(Ser298\{p\}).Raf :: cyt &\Rightarrow MEK(Ser298\{p\}) :: cyt + Raf :: cyt
\end{aligned}
$$

$$
\begin{aligned}
XPP :: cyt + MEK(Ser212\{p\}) :: cyt &\Rightarrow XPP.MEK(Ser212\{p\}) :: cyt \\
MEK(Ser212\{p\}).XPP :: cyt &\Rightarrow MEK(Ser212\{u\}).XPP :: cyt \\
MEK(Ser212\{u\}).XPP :: cyt &\Rightarrow MEK(Ser212\{u\}) :: cyt + XPP :: cyt \\
XPP :: cyt + MEK(Ser298\{p\}) :: cyt &\Rightarrow XPP.MEK(Ser298\{p\}) :: cyt \\
MEK(Ser298\{p\}).XPP :: cyt &\Rightarrow MEK(Ser298\{u\}).XPP :: cyt \\
MEK(Ser298\{u\}).XPP :: cyt &\Rightarrow MEK(Ser298\{u\}) :: cyt + XPP :: cyt \\
ERK(Thr\{u\}).MEK :: cyt &\Rightarrow ERK(Thr\{p\}).MEK :: cyt \\
ERK(Thr\{p\}).MEK :: cyt &\Rightarrow ERK(Thr\{p\}) :: cyt + MEK :: cyt \\
ERK(Tyr\{u\}).MEK :: cyt &\Rightarrow ERK(Tyr\{p\}).MEK :: cyt \\
ERK(Tyr\{p\}).MEK :: cyt &\Rightarrow ERK(Tyr\{p\}) :: cyt + MEK :: cyt \\
MKP :: cyt + ERK(Thr\{p\}) :: cyt &\Rightarrow MKP.ERK(Thr\{p\}) :: cyt \\
ERK(Thr\{p\}).MKP :: cyt &\Rightarrow ERK(Thr\{u\}).MKP :: cyt \\
ERK(Thr\{u\}).MKP :: cyt &\Rightarrow ERK(Thr\{u\}) :: cyt + MKP :: cyt \\
MKP :: cyt + ERK(Tyr\{p\}) :: cyt &\Rightarrow MKP.ERK(Tyr\{p\}) :: cyt \\
ERK(Tyr\{p\}).MKP :: cyt &\Rightarrow ERK(Tyr\{u\}).MKP :: cyt \\
ERK(Tyr\{u\}).MKP :: cyt &\Rightarrow ERK(Tyr\{u\}) :: cyt + MKP :: cyt \\
FRS(Tyr\{u\}).ERK :: cyt &\Rightarrow FRS(Thr\{u\}, Tyr\{p\}).ERK :: cyt \\
FRS(Thr\{u\}, Tyr\{p\}).ERK :: cyt &\Rightarrow FRS(Thr\{u\}, Tyr\{p\}) :: cyt + ERK :: cyt \\
FRS(Tyr\{p\}) :: cyt &\Rightarrow FRS(Tyr\{u\}) :: cyt \\
ERK(Thr\{u\}) :: cyt \;+\; &MEK(Ser212\{p\}, Ser298\{p\}) :: cyt \;\Rightarrow \\
&\Rightarrow ERK(Thr\{u\}).MEK(Ser212\{p\}, Ser298\{p\}) :: cyt \\
Ras(Thr\{u\}) :: cyt \;+\; &FRS(Thr\{p\}, Tyr\{u\}).GS(Thr\{u\}) :: cyt \;\Rightarrow \\
&\Rightarrow Ras(Thr\{u\}).FRS(Thr\{p\}, Tyr\{u\}).GS(Thr\{u\}) :: cyt \\
FRS(Thr\{u\}) :: cyt \;+\; &FGF(Thr\{p\}).R.FGF(Thr\{p\}).R :: cyt \;\Rightarrow \\
&\Rightarrow FRS(Thr\{u\}).FGF(Thr\{p\}).R.FGF(Thr\{p\}).R :: cyt \\
ERK(Tyr\{u\}) :: cyt \;+\; &MEK(Ser212\{p\}, Ser298\{p\}) :: cyt \;\Rightarrow \\
&\Rightarrow ERK(Tyr\{u\}).MEK(Ser212\{p\}, Ser298\{p\}) :: cyt \\
FRS(Tyr\{u\}) :: cyt \;+\; &ERK(Tyr\{p\}, Thr\{p\}) :: cyt \;\Rightarrow \\
&\Rightarrow FRS(Tyr\{u\}).ERK(Tyr\{p\}, Thr\{p\}) :: cyt
\end{aligned}
$$

Figure B.1: A complete model Yamada 2004 [35] of FGF signalling pathway written in BCSL. It consists of 20 entities interacting in 56 rules. To have this model complete, initial state has to be defined – particularly, all entities have their subparts in unphosphorylated state and each of them is present once but FGF and R are present in two repetitions.