

RULE-BASED MODELLING OF BIOCHEMICAL PROCESSES
SPECIFICATION AND ANALYSIS

MATEJ TROJÁK

M U N I
F I

PhD Thesis

Faculty of Informatics
Masaryk University

2023

SUPERVISOR: prof. RNDr. Luboš Brim CSc.

PREFACE

More than fifteen years before this thesis was written, a group of researchers established so-called *CyanoTeam*. The main purpose of the team was to study and model cyanobacteria. One of the primary goals was to create a comprehensive model of cyanobacteria to boost the research of this phenomenal photosynthesis-capable microorganism.

The idea was to compile a series of models targeting individual key parts of the internal gears and combine them into a complete model covering the whole biochemical mechanism inside the bacterium. Putting aside how ambitious this goal was, as a part of this effort, an online platform e-cyanobacterium was created to support this laborious task.

This platform was providing repositories for *in-silico* models and *wet-lab* experiments, centralised around so-called *biochemical space*. The purpose of biochemical space was to provide a common ground for all findings on both the modelling and experimental sides. With enough insight from both of these fundamental parts of systems biology, in theory, the biochemical space should emerge as the targeted comprehensive model. The crucial step in this effort was to specify the common domain of biochemical processes in an understandable form for both parties.

In the early stages of the project, researchers with biologically-oriented background started to formulate the biochemical space. They started with already-known parts of the cyanobacteria processes, such as metabolism, photosynthesis, and the circadian clock. Inevitably, due to the biological expertise of the biochemical space creators, they naturally invented a novel notation that was intuitive for them instead of conducting extensive research on available notations suitable for this purpose.

With the progression of the project, the biochemical space kept increasing its size and became inconsistent, duplicated, and generally, it took a lot of work to follow the intentions of the modellers. That was the moment when members of *Systems biology laboratory (Sybila)* with interest in this project started looking into the biochemical space. Soon they realised that to move forward with the project efficiently, it was necessary to give the notation an exact form and meaning. It became crucial to deploy more systematic and computational methods to keep the biochemical space consistent and unambiguous. And that was the moment the *Biochemical Space Language (BCSL)* emerged...

ABSTRACT

In systems biology, models play a crucial role in understanding the studied systems. There are many modelling approaches, among which rewriting systems provide a framework for describing systems on a mechanistic level. Besides other formalisms based on rewriting, the rule-based approach is one of the most suitable and well-established for modelling in biology. The approach employs abstraction that enables the grouping of multiple biological processes into a single pattern, making the notation compact and precise.

Among many other rule-based languages, *BioChemical Space Language* is a notation used in *Comprehensive Modelling Platform* (CMP), with the purpose of representing a common knowledge base for dynamical models and experimental results, allowing to find connections among them easily. The platform was successfully applied in two instances for the modelling of photosynthesis and processes in cyanobacteria, promoting the impact of this approach.

In this thesis, we formally establish the mathematical foundations of the language. Besides its key role in the modelling platform, the language also employs unique properties that contribute to the field of rule-based modelling. The language relies on the formal basis of the rule-based methodology while preserving the user-friendly syntax of plain chemical equations, making it suitable for the targeted life-sciences oriented audience.

Describing biochemical processes often requires integrating knowledge on an abstract level to simplify the system description or substitute the missing details. For this purpose, we additionally associate the language with regulatory mechanisms. These, in several introduced variations, control the rewriting process of the rules. They can be indirectly used to influence that a particular process is executed under certain conditions (e.g. heating the system, introducing a catalyst or an inhibitor), making it a powerful tool for capturing uncertain or unexplored details.

A novel modelling language, with unique features, abstractions, and modelling techniques such as the regulations, requires analysis techniques which help to investigate the behaviour of the models as well as keep them correct and consistent. Many standard techniques are established for rule-based models, but these are usually approximative, simulation-based methods that provide only limited guarantees on the results. Therefore, we also focus on exact methods, such as model checking and parameter synthesis, to increase the credibility and usefulness of rule-based modelling in general.

With tailor-made analysis techniques for the language, it is necessary to support them on the software side. We developed a tool, eBCSgen (separated from the CMP), to promote the modelling and analysis of models written in the language. It allows performing individual analysis techniques with a focus on usability and interpretation of results, supported by a series of interactive visualisations. Due to the uniqueness of the target user base, we wrapped the tool in the Galaxy framework, a well-established workflow-based ecosystem focused on improving and sharing scientific discoveries.

Finally, to demonstrate the usage of the language in practice, the application of regulations, and the role of analysis techniques in the modelling process, we present several case studies from the biochemical domain, covering the most significant contributions of this thesis.

To summarise, this thesis introduces a novel formal modelling language with rule-based aspects suitable for representing biological processes. This formalism is additionally enriched by regulatory mechanisms motivated by expressing incomplete knowledge ever-present in biology. Several well-established analysis techniques in systems biology are adopted to allow analysing and developing models written using the language. This is also supported by a software tool that implements an environment for modelling using the language and application of analysis. All the results are demonstrated on several case studies from the biological domain.

PUBLICATIONS

The material covered in this thesis expands on the results presented as part of the following publications (listed chronologically). For each publication, I give a simplified list of my roles in manuscript preparation, and a percentage estimate of my contribution.

[BIOSYS 2023] *Rule-based Modelling of Biological Systems Using Regulated Rewriting* (M. Troják, D. Šafránek, S. Pastva, L. Brim)

A journal paper presenting the theory of regulations in multiset rewriting systems with application to BCSL. I formulated the theory of regulations, applied to the context of rule-based modelling, and created the case studies. The paper was accepted in BioSystems journal with my estimated 85% contribution.

[CMSB 2022] *eBCSgen 2.0: Modelling and Analysis of Regulated Rule-Based Systems* (M. Troják, D. Šafránek, B. Brozmann, L. Brim)

A conference tool paper presenting an improved version of support tool for BCSL, introducing a novel feature of behavioural regulations. I formulated the description of regulations and implemented their support in the software. The paper is published in [Tro+22] with my estimated 70% contribution.

[CMSB 2020] *eBCSgen: A Software Tool for Biochemical Space Language* (M. Troják, D. Šafránek, L. Mertová, L. Brim)

A conference tool paper presenting support tool for BCSL. I implemented core parts of the software, integrated it to Galaxy framework and participated in tutorial preparation. The paper is published in [Tro+20b] with my estimated 65% contribution.

[NFM 2020] *Parameter Synthesis and Robustness Analysis of Rule-Based Models* (M. Troják, D. Šafránek, L. Mertová, L. Brim)

A conference paper presenting quantitative extension of BCSL and introduces parameter synthesis and robustness analysis with respect to PCTL formulae. I formulated the language extension and formulated solutions to proposed analysis methods. The paper is published in [Tro+20a] with my estimated 80% contribution.

[PLOS ONE 2020] *Executable Biochemical Space for Specification and Analysis of Biochemical Systems* (M. Troják, D. Šafránek, L. Mertová, L. Brim)

A journal paper presenting BCSL on an informal level. I formulated the language description and prepared case study demonstrating static analysis and usability of the language on different biological processes. The paper is published in [Tro+20c] with my estimated 75% contribution.

[ICSTCC 2018] *Fully Automated Attractor Analysis of Cyanobacteria Models* (N. Beneš, L. Brim, S. Pastva, D. Šafránek, M. Troják, J. Červený, J. Šalagovič)

A case study paper presenting the technique from [Bar+17] applied to models from the e-cyanobacterium.org repository. I helped with the experimental evaluation and contributed to the text describing the models and the evaluation results. The paper was awarded the best paper award in its category. The paper is published in [Ben+18] with my estimated 14% contribution.

[SASB 2018] *Executable Biochemical Space for Specification and Analysis of Biochemical Systems* (M. Troják, D. Šafránek, L. Brim, J. Šalagovič, J. Červený)

A conference paper presenting a new version of BCSL. I formulated the formal definition of the language and static analysis, including their application examples. The paper is published in [TvB20] with my estimated 70% contribution.

[CMSB 2016] *E-cyanobacterium.org: A Web-based Platform for Systems Biology of Cyanobacteria* (M. Troják, D. Šafránek, J. Hrabec, J. Šalagovič, F. Romanovská, J. Červený)

A tool paper presenting e-cyanobacterium.org as an instance of CMP. I helped with writing most of the text and brought several ideas to the web platform development. The paper is published in [Tro+16] with my estimated 30% contribution.

[SASB 2015] *Formal Biochemical Space with Semantics in Kappa and BNGL* (T. Děd, D. Šafránek, M. Troják, M. Klement, J. Šalagovič, L. Brim)

A conference paper describing the first version of BCSL defined by embedding to another language. I helped to formulate the formal definition and algorithms for the embedding. The paper is published in [Děd+16] with my estimated 30% contribution.)

The relationship between the contents of the thesis and the listed publications is as follows: Chapters 1 and 2 are partially covered by [BIOSYS 2023] and then mostly as a compilation of existing background. The informal part of Chapter 3 corresponds to [PLOSONE 2020], while the formal parts are compilation of [SASB 2018] and [BIOSYS 2023]. Chapter 4 completely corresponds to [BIOSYS 2023], enriched by formal proofs and theoretical results. Chapter 5 is mostly covered by [SASB 2018] and [NFM 2020]. Chapter 6 is covered by [CMSB 2020] and [CMSB 2022]. Chapter 7 is composed of multiple parts of all papers.

[SASB 2015] does not directly contribute to the content of this thesis, although it provided some formal fundamentals of BCSL as well as basics for some of the case studies in Chapter 7. [CMSB 2016] is a demonstration of the usage of CMP, only indirectly using BCSL. Finally, [ICSTCC 2018] was only a demonstration of the usefulness of models hosted in a CMP instance.

ACKNOWLEDGEMENTS

I want to express my gratitude for the opportunity to experience the journey of PhD studies. Although the path was quite demanding and had many ups and downs, it was a significant and beneficial period of my life that has transformed me in many ways for the better. The personal enrichment by many travelling opportunities made this period of life fascinating.

I want to thank my mentor David Šafránek for his everlasting willingness to help and provide technical discussions. He introduced me to the ways of science long before my PhD studies and ignited my passion for doing research.

I would like to thank my supervisor Luboš Brim for his endless patience and support over the years, who led me with his experience inherited from our common academic genealogy of supervisors, dating back to 17th century. In their honour, I added some famous quotes at the beginning of each chapter.

Gottfried Wilhelm Leibniz (Universität Leipzig 1666)

└─ Nicolas Malebranche

└─ Johann Bernoulli (Universität Basel 1690)

└─ Leonhard Euler (Universität Basel 1726)

└─ Joseph-Louis Lagrange

└─ Jean-Baptiste Joseph Fourier (École Normale Supérieure)

└─ Gustav Peter Lejeune Dirichlet (GAU Göttingen 1823)

└─ Leopold Kronecker (Universität Berlin 1845)

└─ Mathias (Matyas) Lerch (Universität Berlin 1885)

└─ Otakar Borůvka (Masaryk University Brno 1926)

└─ Jiří Hořejš (Masaryk University Brno 1963)

└─ Luboš Brim (Masaryk University Brno 1986)

I would also like to thank all members of Systems biology laboratory (Sybila) for the healthy work environment, rich social life, and mutual support in both work and life. Namely, I am grateful to my PhD candidate colleagues, collaborators and students over the years, including Samuel Pastva, Eva Šmijáková, Martin Demko, Jan Červený, Lukrécia Mertová, Radoslav Doktor, Ondřej Lošťák, and many others.

I want to thank my family, particularly my parents, who have shaped my beliefs and attitude throughout my life. Their continuous support, along with my brother, has been immensely helpful. I also consider my closest friends as my family, including those who stayed with me despite long distances.

Last but not least, I am grateful to my colleagues from CzechGlobe and RECETOX, who helped me to keep my feet on the ground during the numerous downs of my studies.



CONTENTS

PREFACE	iii
ABSTRACT	v
PUBLICATIONS	vii
ACKNOWLEDGEMENTS	xi
INTRODUCTION	1
1 PRELIMINARIES	11
1.1 Multiset rewriting systems	12
1.2 Rule-based modelling in general	13
2 STATE OF THE ART	17
2.1 Rule-based languages	19
2.1.1 Kappa language	19
2.1.2 BioNetGen language	19
2.1.3 PySB	21
2.1.4 Chromar	21
2.1.5 Language for Biochemical Systems	23
2.1.6 SBML-multi package	25
2.2 Analysis methods	27
2.2.1 Simulation	28
2.2.2 Model checking	29
2.2.3 Parameter synthesis	31
2.2.4 Monitoring	32
2.2.5 Robustness analysis	32
2.2.6 Static analysis	32
2.2.7 Regulations	33
2.3 Summary	34
3 BIOCHEMICAL SPACE LANGUAGE	37
3.1 Introduction to Biochemical Space Language	37
3.1.1 Agents	37
3.1.2 Rules	39
3.1.3 Semantics	41
3.2 Biochemical Space Language definition	44
3.2.1 Non-parametrised fragment	48
3.2.2 Qualitative fragment	49
3.2.3 Syntactic extensions	50
3.3 MRS encoding	54
3.4 BCSL construction	57
3.4.1 Objects construction	58
3.4.2 Translation function	63
3.4.3 Matching and replacement	63
3.5 Summary	64

4	REGULATIONS	67
4.1	Classes of regulated systems	67
4.1.1	Regular rewriting	68
4.1.2	Ordered rewriting	68
4.1.3	Programmed rewriting	69
4.1.4	Conditional rewriting	70
4.1.5	Concurrent-free rewriting	71
4.2	Properties of regulated multiset rewriting systems . . .	71
4.2.1	Generative power comparison	72
4.2.2	Expressive power	79
4.3	Regulated BCSL	81
4.3.1	Regular rewriting	81
4.3.2	Ordered rewriting	82
4.3.3	Programmed rewriting	83
4.3.4	Conditional rewriting	83
4.3.5	Concurrent-free rewriting	84
4.4	Summary	85
5	ANALYSIS TECHNIQUES	87
5.1	Simulation	87
5.2	Model checking	91
5.3	Parameter synthesis	96
5.4	Robustness analysis	99
5.5	Static analysis	100
5.5.1	Rule redundancy elimination	101
5.5.2	Context-based reduction	103
5.5.3	Static non-reachability analysis	105
5.6	Summary	106
6	SOFTWARE TOOL	107
6.1	Overview	107
6.1.1	Core	109
6.1.2	Model syntax	111
6.1.3	Parsing	112
6.1.4	State transition system	113
6.1.5	Regulations	115
6.2	Analysis methods	116
6.2.1	Simulation	116
6.2.2	Model checking	117
6.2.3	Parameter synthesis	118
6.2.4	Static analysis	119
6.3	Visualisation	119
6.3.1	Transition system	119
6.3.2	Simulation	121
6.3.3	Parameter synthesis	121
6.4	Distribution	122
6.4.1	Conda	122
6.4.2	Docker	123

6.4.3	Galaxy	123
6.4.4	SBML-multi	132
6.5	Documentation	132
6.6	Testing	133
6.7	Summary	133
7	CASE STUDIES	135
7.1	Modelling with BCSL	135
7.1.1	Circadian clock	135
7.1.2	Fibroblast growth factor signalling pathway	138
7.1.3	Tumour growth	139
7.2	Parameter synthesis	141
7.2.1	Circadian clock	141
7.2.2	Tumour growth	145
7.3	Static analysis	146
7.4	Regulations	148
7.4.1	MAPK pathway	148
7.4.2	Cell cycle	150
7.4.3	Methane combustion	153
7.4.4	Circadian clock	154
7.4.5	Metabolism of sugars in E. Coli	155
7.5	Summary	158
	CONCLUSIONS	161
A	APPENDIX: DIGITAL ATTACHMENTS	163
	BIBLIOGRAPHY	165

MODELLING in systems biology is a key tool for understanding the studied system [Kito2; TN15; TCNNo2]. The modelling becomes challenging with the increasing complexity of systems, which is typical in biology due to the high heterogeneity of molecules and their possible modifications. The combinatorial complexity of biological systems is recognisable on two levels – the specification level and the computational level [Ste+14].

The specification level is concerned with how such a system can be specified. For example, it is crucial for a modeller to specify all types of proteins, complexes they can form, and changes they can undergo, respecting conditions governing those changes in a robust and efficient way. It is crucial to use modelling approaches that allow expressing such complexity on a detailed mechanistic level in a concise, readable, and elegant form.

On the other hand, the computational level deals with the problem of whether an already-specified model is computationally feasible given a large number of states (possible configurations) and the even larger number of transitions between such states, how such a model can be efficiently stored and represented digitally, and whether some properties of interest can be analysed and evaluated in a reasonable amount of computing time.

While many specifications use an explicit description of biochemical systems [CHO8; Pet81], there are also approaches that use implicit description by grouping reactions and parameters that apply to many types of molecules into one reaction template. Among them, a promising approach is *rule-based* modelling [Dan+07; DKo7; Chy+13; Joh+11], where a rule provides a pattern to describe a reaction template. The existing rule-based model specification systems (see [Chapter 2](#) for details) often concentrate on model specification only, allowing the user to export the specified model into a dedicated format. However, many solutions to the specification problem also contain a method of interpreting the specified model. This is done by providing a method to simulate the model or a method to convert it into a format that can be used for simulations in other programs.

To that end, for a rule-based language to be eligible in systems biology, it is crucial to combine the advantages of rule-based abstraction with the simplicity of chemical reactions. It should emerge into key aspects such as *human-readability* – the model is easy to read, write, and maintain, *executability* – formal executable semantics is defined allowing efficient static analysis and consistency checking, *universality* – principally different cellular processes can be sufficiently com-

bined in a single specification, and *compactness* — the combinatorial explosion of the description is avoided.

The rule-based approach, like other mechanistic approaches, provides a compact way to represent biochemical systems on a detailed level. However, another typical issue in modelling biological systems is a lack of knowledge about the system of interest. With the increasing complexity of a model also raises the number of unknown or uncertain details. Describing incomplete information is often challenging or not even possible to express in a suitable way. Moreover, sometimes we need a mechanism to influence that a particular reaction is executed (resp. not executed) under certain conditions (e.g. heating the system, introducing a catalyst or an inhibitor).

This motivates to employ *regulation* mechanisms [Das04; IH88] to rule-based systems. The regulations can have many different forms, but generally, they provide an additional mechanism for controlling the rule application by restricting the conditions when a rule can be executed. The regulations can be beneficial in multiple applications, such as simplifying the system description, substituting the missing details yet to be discovered, or even sketching systems properties in synthetic biology. It becomes particularly useful when newly discovered biological processes need to be introduced into the system. To incorporate the novel processes into the model, detailed knowledge of them is usually necessary. However, in biology, these processes are not always known and discovering their mechanistic components is a tedious and challenging task. Nevertheless, the effects of such processes are crucial for capturing the dynamic behaviour of the modelled system. As we will see later in this thesis, the regulations can be used for the purpose of describing incomplete information on an abstract level.

While the specification aspect of the rule-based modelling is one thing, it is also crucial to ensure there are ways to analyse such models [Ste+14]. This becomes especially crucial with the addition of regulations, which make such a process unique. The main effort is to develop exact methods which provide global guarantees overcoming the limitations of approximative, sampling-based methods. It is also essential to leverage the employed abstraction [And+17] and develop methods operating on the level of the model specification.

There are multiple ways to analyse rule-based models through existing computational methods established in computer science (see [Section 2.2](#) for details). Nonetheless, the general drawback of available methods for rule-based languages is the lack of support for exact approaches. Usually, approximate and simulation-based methods are employed, providing only limited insight into the global properties of models. To the best of our knowledge, the crucial problems in model analysis, such as model parameter synthesis or robustness analysis, are not covered properly.

With established analysis methods, software support becomes necessary to allow performing analysis tasks automatically. It is also beneficial when the tool supports modelling tasks, such as model creation and maintenance. With the life-sciences oriented audience, it is crucial for the tool to be easily accessible (in a standardised way) and easy to use to attract users.

OBJECTIVE I: THE LANGUAGE

In [Kle+13], a general modelling framework called *Comprehensive Modelling Platform* (CMP) was introduced. CMP combines model building, model analysis, experimental validation, and annotation tasks in a single public website related to a particular system. It respects the need for maintaining existing ordinary differential equations (ODE) models (which is still a typical scenario in systems biology) but allows to align them with a common notation for biochemical processes enriched by annotations, forming *Biochemical space* (BCS) [Kle+14]. Such a comprehensive solution supports modellers in building mathematical models that have clear biochemical meaning and can be easily integrated.

With the usage of the platform in practice, namely in modelling of photosynthesis [Šaf+11] and cyanobacteria processes [Tro+16], it became evident that keeping the BCS understandable by biologists, compact in size, and executable in terms of allowing analysis tasks ensuring consistency of the description was crucial [Cla+12]. The core specification notation was extracted and properly formalised, giving rise to *Biochemical Space Language* (BCSL), with rule-based features. We point the reader to the preface of this thesis for more details on this process. The first attempt [Děd+16] to formalise the language was by encoding it to a different rule-based language BNGL [Har+16]. However, such an approach was not satisfactory in terms of the proper interpretation of rules and maintaining the desired level of detail, as well as customising software support.

OUR CONTRIBUTION The need to formally anchor the language in computer science is raised by the intention of providing exact methods in systems biology with guarantees on the expected behaviour of models with respect to a given specification. In Chapter 3, we focus on the first and primary objective of this thesis, which is to define the rigorous mathematical basis of the language.

The definition is twofold – we provide precise definitions of individual constructs employed in the language using fundamental mathematical objects and operations and relate the language to the well-defined low-level formalism of multiset rewriting systems in order to establish a link to a more general class of languages. This allows us

to define specific tailor-made analysis methods and reuse existing ones defined for more general or related formalisms.

In [Section 7.1](#), we also provide several case studies from the biological domain to claim the usefulness of BCSL in modelling. In particular, we show complex protein modifications on an example of a circadian clock in cyanobacteria, an example of signal transduction in a model of fibroblast growth factor signalling pathway, and a population-based model of tumour growth with quantitative properties.

OBJECTIVE II: REGULATIONS

Biological processes are often governed by complex regulatory mechanisms that ensure the correct execution sequence of individual processes and their mutual effects. Such mechanisms can be observed at the *molecular* level, e.g. in enzymatic reactions [[FHo7b](#)], where the regulatory molecule influences the course of the reaction in a non-trivial way. It can be also observed on the *cellular* level, e.g. in cell cycle [[Sch98](#)], where high-level decision-making is done based on the current state of the cell. Finally, it can even be recognised on the *phenotype* level, e.g. in the development of a multicellular organism, where cell differentiation [[SD21](#)] occurs in the proper phase and location.

The ability to capture such phenomena in modelling the biological system at the mechanistic level is often difficult due to insufficient knowledge of biochemical details and sometimes undesirable due to their high complexity. Similarly, capturing them using the quantitative properties [[Tro+20a](#)] is usually not easy or even possible. Nevertheless, the effects of such mechanisms are crucial for capturing the dynamic behaviour of the modelled system. To that end, we want to introduce a method that allows describing the regulatory mechanisms at the level of modelling language in an abstract way, regardless of detailed information on how such a mechanism works.

For example, consider the transport and metabolism of sugars in *Escherichia Coli*. Under favourable conditions, the bacteria preferentially metabolise glucose over secondary carbon sources [[Esc+12](#)]. Particular enzymes involved in the phosphotransferase system (PTS), responsible for the transport and phosphorylation of sugars, were identified, and their role was explored. Two main inhibitory mechanisms, catabolite repression (inhibition of the synthesis of enzymes involved in the catabolism) and inducer exclusion (direct inhibition of the transport membrane proteins), govern this phenomenon. Even though these mechanisms are known in quite detail nowadays, it was not always like that. The first research articles [[CH76](#); [HD82](#)] reported the observed effects of this phenomenon, but the minimal knowledge did not allow to efficiently capture these processes at the

level of a model. More long-term research [AOB13; Hog+98] advanced the knowledge sufficiently enough such that it was possible to capture comprehensively, model, and analyse the processes. With regulations, it is possible to model the observed behaviour even in the early stages of the research, as we demonstrate in the case study in [Section 7.4.5](#).

OUR CONTRIBUTION Since regulations are a general approach well-established in computer science [Das04; FMVPO4], we first introduced them to a more general formalism of rewriting systems. Rewriting systems are reduction systems that can be used to directly represent (bio)chemical reactions by rewriting rules. The rules apply to terms typically constructed from variables and constants, representing molecules and biological objects. For biological systems without spatial information, the molecules can be present in multiple copies with no particular arrangement. Therefore, a suitable representation of the reality for a term is *multiset*, resulting in a multiset rewriting system (MRS) [Mes92; Bar+08].

In [Chapter 4](#), we introduce several variants of regulation mechanisms of multiset rewriting systems. Each class of regulated systems has its custom regulatory mechanism applicable to different modelling scenarios. Namely, we introduced these mechanisms: *regular* rewriting restricted by regular language over rules, *ordered* rewriting restricted by a strict partial order on rules, *programmed* rewriting restricted by successors for each rule, *conditional* rewriting restricted by prohibited context for each rule, and *concurrent-free* rewriting restricted by resolving concurrent behaviour of multiple rules.

While MRS is a general formalism and provides a formal basis for the theory of regulations, it is considered a low-level formalism, which is not particularly suitable for describing complex biological systems due to its explicit form and minimal tool support. However, for its simplicity and natural representation of biology in it, MRS can serve as a base for any (biologically oriented) rewriting-based formalism. To support this claim, we establish regulations in the context of rule-based modelling on an example of BCSL by relating it to MRS.

Using BCSL, we demonstrate the applicability of regulations on several short examples as well as a more detailed case study from the biological domain ([Section 7.4](#)). Namely, we show on a model of MAPK signalling pathway how *conditional* regulation is used to explicitly represent the inhibition activity of a kinase; we show on a model of cell cycle how *programmed* regulation is used to enforce programmed death as an immediate response to cell damage; we show on a model of methane combustion how *concurrent-free* regulation can prioritise complete combustion over incomplete one; and we show on a model of the circadian clock in cyanobacteria how *regular* regulation is used to extract a typical behavioural scenario from otherwise highly com-

plex network. Finally, on the aforementioned case study of the sugars transport and metabolism in *E. Coli*, we illustrate a typical research process in biology and how regulations can support this process.

OBJECTIVE III: ANALYSIS METHODS

While a suitable specification for biological systems improves the whole modelling process, it is usually not sufficient. With a new language, it is necessary to build an infrastructure to analyse such models and support their construction. As a next step, we formally establish an analysis base for BCSL. The goal is to employ well-established analysis methods usually used in systems biology. These include, for example, simulation, model checking, and parameter synthesis. The methods should be generally applicable to any rule-based formalism. The focus is put on model checking, parameter synthesis, and static analysis, as we primarily focus on methods providing exact results.

Model checking

To the best of our knowledge, the existing model checking methods [Cla+08] for rule-based languages use only an indirect approach, i.e. a particular model can be exported to a more general formalism. There do not exist methods for model checking of rule-based models with exact guarantees. Therefore, one of the goals is to establish a method available directly for the rule-based description or to provide an improvement of an existing approach such that they produce exact results (no heuristics). The emphasis is placed on the reachability problem as one of the most important problems when analysing biological systems.

Computer science offers several approaches for model checking of models, which can be potentially used in the case of rule-based models. One option is to use model checking methods developed for reaction-based systems, which avoid the exploration of entire state space. Another option would be to use an existing approach which requires complete state space and encode it symbolically or to explore on-the-fly techniques. For the efficiency reasons, the ideal solution avoids building the state space completely. It is the most challenging task and requires the development of new symbolic and static methods.

OUR CONTRIBUTION We develop model checking methods to analyse rule-based models with respect to computation tree logic (CTL) and *probabilistic* computation tree logic (PCTL) properties. For the CTL model checking, we employ an explicit method and adapt it for BCSL. With the PCTL model checking, we use a method for probabilistic reachability analysis of DTMC, which constructs a regular expres-

sion (RE) using a state elimination algorithm. The evaluation of RE gives the probability of reaching targeted species. The details on both approaches are available in [Section 5.2](#). Additionally, in [Section 5.5.3](#), we offer a method for non-reachability properties, solved purely on the syntactic level of the language.

Parameter synthesis

Parameter synthesis is a crucial branch of model analysis when the parameters of the modelled system are not completely known. However, in the rule-based community, the only known work [LF16] uses statistical model checking to solve the parameter estimation problem (see [Section 2.2.3](#) for details). Therefore, another goal of this thesis is to develop a parameter synthesis algorithm for rule-based models, which provides exact results avoiding heuristics and statistical approaches.

It is also extremely useful to provide a way how to compute the robustness of a property with respect to specified parameter space. Robustness analysis characterises the mean validity of a formula over all admissible parameter values. To the best of our knowledge, this problem was not yet formulated for rule-based systems. Another goal of this thesis is to define and solve this problem in the context of rule-based systems.

OUR CONTRIBUTION In [Section 5.3](#), we employ a solution to this problem based on the model checking approach. The parameter synthesis can be formulated in terms of an efficient model checking method. For this purpose, we use the mentioned method for PCTL model checking, which constructs RE. It can be generalised to the parametrised case. Instead of a single probability evaluation, we obtain a probability function of parameters, which can then be used for parameter space exploration. This function of parameters can also be integrated over the parameter space, and this way, the robustness can be computed.

Static analysis

Static analysis methods are well-studied in the rule-based community. They provide limited yet often crucial insight into the model properties on the syntactic level, making them extremely efficient. The key reason is the abstraction employed by this approach, which enables the detection of non-trivial relationships among interacting molecules (see [Section 2.2.6](#) for details on existing methods). For our intentions, the main objective to achieve using static methods is to find and eliminate rules that can be potentially in conflict or are redundant in the context of the model and to provide reduction tech-

niques that allow performing the dynamic analysis techniques more efficiently at the price of reducing the obtained information.

OUR CONTRIBUTION In [Section 5.5](#), we describe three methods that contribute to the field of static analysis of rule-based models. As we already mentioned in the model checking part, we provide a method for non-reachability that can, under certain circumstances, avoid dynamic explorative methods and answer the inverse reachability properties statically. We also developed a method to detect redundant rules that typically do not bring additional information to the model, but they keep making it larger. Finally, the rule-based approach typically employs modifications of the inner structure of molecules as well as complex formations of such molecules. With a context-based reduction, we simplify the model to such an extent that only complex formations and dissociations are kept, preserving some properties while making the dynamical model analysis simpler.

OBJECTIVE IV: SOFTWARE TOOL

With a novel modelling language, regulation mechanisms, and analysis techniques, it is appropriate to provide suitable software support. The final objective of the thesis is to develop a tool to support the modelling and analysis of BCSL, which includes the implementation of the proposed analysis methods. For the specificity of the life-science oriented community, it is necessary to ensure that the tool is easily accessible and can be used even by users with no particular education in computer science.

OUR CONTRIBUTION In [Chapter 6](#), we present the tool eBCSgen. It implements an environment suitable for editing and maintenance of the BCSL models by providing the following functionality: an editor with syntax highlighting and validation during the model building process, a simulation engine offering both deterministic and stochastic simulations, reaction network generator, transition system generator, and all the proposed techniques for model checking and parameter synthesis.

To satisfy specific needs of the target audience, we implement the tool as a series of Galaxy [\[Afg+18\]](#) wrapper tools, a standardised well-established framework used mostly in bioinformatics. We also employ several visualisation techniques to display transition systems, simulation plots, and results of parameter synthesis. All this is then documented in a detailed tutorial describing individual wrappers and visualisations. Finally, to support the usage of other tools, we allow exporting the models to the appropriate SBML [\[Huc+03\]](#) standard format.

SUMMARY

In this work, we contribute to the field of rule-based modelling in systems biology by introducing a novel Biochemical Space language with its unique features, arising from the requirements for a formal knowledge base for the Comprehensive modelling platform. In addition to the employment of the rule-based approach, we enrich the language with regulatory mechanisms that go beyond the standard expressiveness of the rule-based approach and allow the formulation of non-trivial mechanisms concisely. With a novel language and, moreover, extension with regulations, the need for tailor-made analysis methods is covered by developing dynamical and static techniques for simulation, model checking, parameter synthesis, and consistency checking. Finally, modelling using the language and analysis of implemented models are supported by the software tool eBCSgen.

Nature does not make leaps.

GOTTFRIED WILHELM LEIBNIZ

LET US start with some definitions and notations used throughout the thesis. These include multisets as a central data structure, multiset rewriting systems used as a low-level base formalism, and the description of the rule-based approach in a general and semi-formal way, with the purpose to provide to the reader an intuition behind the approach.

Intuitively, a multiset is a set of elements with allowed repetitions. Let \mathcal{S} be a finite set of *elements*. A *multiset* over \mathcal{S} is a total function $M : \mathcal{S} \rightarrow \mathbb{N}$ (where \mathbb{N} is the set of natural numbers including 0). For each $a \in \mathcal{S}$ the *multiplicity* (the number of occurrences) of a is the number $M(a)$. By $\mathbb{M}_{\mathcal{S}}$, we denote the set of all possible finite multisets over elements \mathcal{S} . We often omit the subscript in cases when the domain of used elements is clear or not important. Operations and relations over multisets are defined in a standard way, taking into account the repetition of elements:

- Union $M_1 \cup M_2 : \forall a \in \mathcal{S}. (M_1 \cup M_2)(a) = M_1(a) + M_2(a)$
- Difference $M_1 \setminus M_2 :$

$$\forall a \in \mathcal{S}. (M_1 \setminus M_2)(a) = \begin{cases} M_1(a) - M_2(a) & \text{if } M_2(a) \leq M_1(a) \\ 0 & \text{otherwise} \end{cases}$$
- Intersection $M_1 \cap M_2 : \forall a \in \mathcal{S}. (M_1 \cap M_2)(a) = \min\{M_1(a), M_2(a)\}$
- Submultiset $M_1 \subseteq M_2 : \forall a \in \mathcal{S}. M_1(a) \leq M_2(a)$
- Equality $M_1 = M_2 : \forall a \in \mathcal{S}. M_1(a) = M_2(a)$
- Occurrence $a \in M : \exists a \in \mathcal{S}. M(a) \geq 1$

We will also often rely on operations over tuples, especially in [Section 3.4](#). Let $O = (o_1, \dots, o_n)$ be an n -tuple. By $M(O)$, we denote a multiset constructed from tuple O . Finally, we abuse the notation, and by $|A|$, we denote either the dimension of tuple A or the cardinality of (multi)set A .

To simplify notation of tuples concatenation, we define *concatenation* of two tuples $X = (x_1, \dots, x_n), Y = (y_1, \dots, y_m)$, as $X \uparrow\uparrow Y = (x_1, \dots, x_n, y_1, \dots, y_m)$. We also generalise this notation to k number of tuples $T = (T_1, T_2, \dots, T_k)$ as $\uparrow\uparrow_{i=1}^k T = T_1 \uparrow\uparrow T_2 \uparrow\uparrow \dots \uparrow\uparrow T_k$.

We often use the term *state* in the context of state transition systems. The state in this thesis usually represents a multiset if not stated otherwise. Therefore, these two terms are freely interchangeable. Additionally, by the state transition system, we usually refer to a labelled transition system $LTS = (S, T, L)$, where S is a set of states (or multisets), $T \subseteq S \times L \times S$ is a transition relation, and L is a set of labels. Assuming an initial state $s_0 \in S$, the *path* is a sequence of states $s_0 s_1 s_2 \dots$ such that $\forall s_i, s_{i+1} : (s_i, l, s_{i+1}) \in T$ for some $l \in L$.

1.1 MULTISSET REWRITING SYSTEMS

We recall some definitions and known results about rewriting systems over *multisets*. A multiset rewriting *rule* describes how a particular multiset is transformed into another one. A multiset rewriting system consists of a set of multiset rewriting rules, defining how the system can evolve, and an initial multiset, representing the starting point for the rewriting.

A *multiset rewriting rule* over \mathcal{S} is a pair $\mu = (\bullet\mu, \mu\bullet)$ of multisets over \mathcal{S} , written as $\mu : \bullet\mu \rightarrow \mu\bullet$ for convenience. The rule rewrites elements specified in the left-hand side $\bullet\mu$ to elements specified in the right-hand side $\mu\bullet$.

A *multiset rewriting system* (MRS) over \mathcal{S} is a pair $\mathcal{M} = (M_0, \xi)$, where M_0 is the initial multiset (*state*) and ξ is a finite set of multiset rewriting rules, both over \mathcal{S} .

We denote by \mathbb{MRS} the class of (non-regulated) multiset rewriting systems. An example of an MRS is given in [Figure 1](#).

$$\mathcal{M} = \left(\begin{array}{l} M_0 = \{A, A\}, \\ \xi = \left\{ \begin{array}{l} \mu_1 : \{A, A\} \rightarrow \{B, A\}, \mu_2 : \{A\} \rightarrow \{X\}, \\ \mu_3 : \{B\} \rightarrow \{C\}, \mu_4 : \{C\} \rightarrow \{B\} \end{array} \right\} \end{array} \right)$$

Figure 1: Example of an MRS over elements $\mathcal{S} = \{A, B, C, X\}$.

Let M be a multiset and $\mathcal{M} = (M_0, \xi)$ an MRS, both over \mathcal{S} . A rule $\mu \in \xi$ is *enabled* at M if $\bullet\mu \subseteq M$. The *application* of an enabled rule $\mu \in \xi$ to M , written $M \rightarrow_\mu M'$, creates a multiset $M' = (M \setminus \bullet\mu) \cup \mu\bullet$. A *run* π of \mathcal{M} is an infinite sequence of multisets $\pi = M_0 M_1 M_2 \dots$ such that for any *step* $i > 0$ holds that $M_{i-1} \rightarrow_\mu M_i$ for some $\mu \in \xi$. We denote by $\pi[i]$ the multiset created in step i . A *run label* $\vec{\pi}$ of a run π is an infinite sequence of rules $\vec{\pi} = \mu_1 \mu_2 \mu_3 \dots$ such that for any *step* $i > 0$ holds that $\pi[i-1] \rightarrow_{\mu_i} \pi[i]$. We denote by $\vec{\pi}[i]$ the rule applied in step i .

In order to ensure the infiniteness of runs, we implicitly assume the presence of a special *empty* rule $\varepsilon = (\emptyset, \emptyset)$. This rule is always enabled, but once applied, no other rule can be used except ε . It is also the only rule where both sides are equal. Otherwise, we require

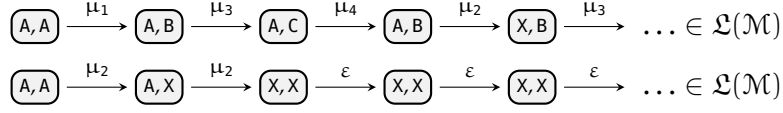


Figure 2: Example of runs of the MRS from Figure 1, including their run labels.

both sides to be different. It ensures that the set of rules ξ is always non-empty.

The *semantics* of the system \mathcal{M} is a (potentially infinite) set of runs starting in the initial multiset. We recognise two types of semantics – *maximal* semantics, denoted $\mathfrak{L}(\mathcal{M})$, assuming only maximal runs (i.e. rule ε can be applied only when no other rule of the system is enabled) and *general* semantics, denoted $\mathfrak{L}_{\text{ALL}}(\mathcal{M})$, assuming all runs (i.e. rule ε can be applied at any time). In Figure 2, we give an example of maximal runs. If not stated otherwise, by run, we always mean maximal run.

1.2 RULE-BASED MODELLING IN GENERAL

This chapter defines a simplified and general version of the rule-based system. The goal is to gain an intuition in using the rule-based approach. The defined rules are based on multiset rewriting rules (referred to as *reactions*). Then, the rule is defined as a set of reactions, encoding multiple possibilities of how it can be used, depending on a particular context.

In addition to multiset rewriting rules as defined in the previous section, we also associate them with rational functions over a specified domain of elements (also called *species*). With Γ , we denote the set of all arbitrary rational functions. The notation $v(\mathbb{M})$ with $v \in \Gamma$ and $\mathbb{M} \in \mathbb{M}$ denotes the evaluation of function v to a rational number by substituting each species a by its corresponding value $\mathbb{M}(a)$.

A reaction ω is a triple $(\text{LHS}, \text{RHS}, v) \in \mathbb{M} \times \mathbb{M} \times \Gamma$ where LHS is left-hand side, RHS is right-hand side, and v is a rational rate function. By Ω , we denote the set of all reactions. Informally, a reaction describes how species interact and change their number of copies in the state of the system.

The application of a reaction $\omega = (\text{LHS}, \text{RHS}, v)$ to a multiset \mathbb{M} , written $\omega(\mathbb{M})$, is a pair $(\mathbb{M} \setminus \text{LHS} \cup \text{RHS}, v(\mathbb{M}))$ if $\mathbb{M} \setminus \text{LHS} \geq 0$; $(\mathbb{M}, 0)$ otherwise.

Applying a reaction to a multiset means removing from the state the number of copies of elements that appear on the left-hand side of the reaction and adding the elements that are present on the right-hand side of the reaction. Additionally, the *rate function* v is evaluated in the process and provides reaction rate, i.e. speed at which the reaction happens. When the number of occurrences of some species required by LHS is not high enough, the application is not possible,

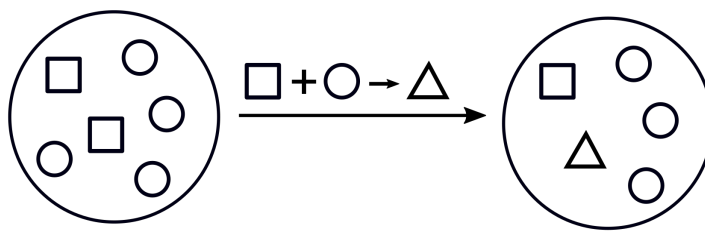


Figure 3: A graphical example of reaction application. In a multiset represented by a large circle we have three kinds of species – \square , \circ , and \triangle . In the reaction $r = (\{\square: 1, \circ: 1\}, \{\triangle: 1\}, 2 \times \square \times \circ)$, the species \square and \circ are consumed and a new specie \triangle is produced. The rate of the reaction is given as a function of molecular counts of both reactants. After substituting the counts to $\square = 2$ and $\circ = 4$, the evaluation of $2 \times 2 \times 4$ gives the rate with value 16.

and the rate is zero (the system remains in the same state). For an example of a reaction application, see [Figure 3](#).

In the reaction-based setting, the elements in the multisets are individual species or molecules. To move to the rule-based world, we need to work with *sets of species* instead. In rule-based terminology, these sets are usually called *agents*. Then, a *rule* ρ is, similarly to the reaction, a triple $(\text{LHS}, \text{RHS}, \nu) \in \mathbb{M}_{2^{\mathcal{S}}} \times \mathbb{M}_{2^{\mathcal{S}}} \times \Gamma$, where \mathcal{S} is set of elements, and the rule sides are multisets over sets of elements.

A rule can also be represented by a set of reactions, since it is a generalisation of multiple reactions. It can be viewed as a high-level compact definition of a set of chemical reactions. Instead of a single possible way of application, as in the case of reaction, there are multiple outcomes of the application to a given multiset. By $\omega \in r$, we denote a reaction from the definition domain of rule r . By \mathbb{R} , we denote the set of all rules.

Rule application of a rule r to a multiset M , written $r(M)$, is a set $\{\omega(M) \mid \omega \in r\}$ of all reaction applications from the definition domain of the rule. Applying a rule to a multiset produces a set of *multiset-rate* pairs. This is caused by the fact that all possible reactions represented by the rule can produce a different outcome (we do not consider reactions with zero rate due to a non-negative number of repetitions requirement in reaction application). An example of a rule application is given in [Figure 4](#).

A rule-based model \mathcal{B} is a pair $(\mathcal{R}, M_0) \in 2^{\mathbb{R}} \times \mathbb{M}$ where \mathcal{R} is a set of rules and M_0 is an initial state (multiset) of the model. The initial state describes the number of occurrences of individual species in the default configuration, and the set of rules defines how the species can interact. The *semantics* of the model is given in terms of rule application, which is transitively applied to the initial state, giving rise to a state transition system.

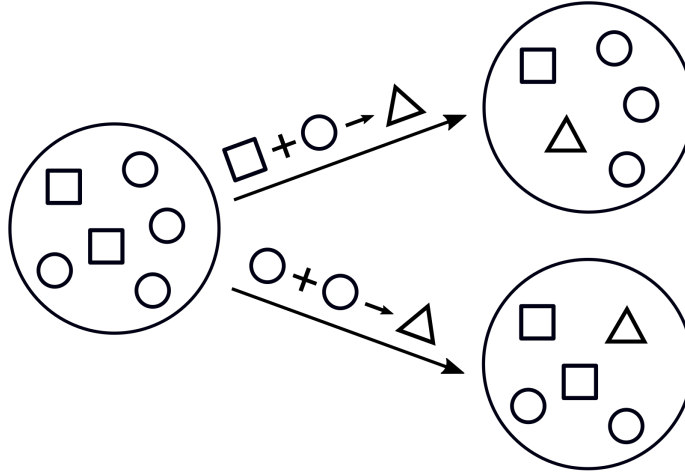


Figure 4: A graphical example of the rule application. In a multiset represented by a big circle we have three kinds of agents – \square , \circ and \triangle . Given rule $\rho = (\{\{\square, \circ\} : 1, \circ : 1\}, \{\triangle : 1\}, 2 \times \{\square, \circ\} \times \circ)$ (we omit the brackets in the case of singletons) represents a set containing reaction $\omega_1 = (\{\square : 1, \circ : 1\}, \{\triangle : 1\}, 2 \times \square \times \circ)$ and reaction $\omega_2 = (\{\circ : 2\}, \{\triangle : 1\}, 2 \times \square \times \circ)$. The rule application can therefore produce a different state for each reaction, each created by particular reaction application. The associated rate is the same for both inferred reactions and it is simplified to reaction-based domain. It can be evaluated as $2 \times 2 \times 4^2 = 32$.

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

*The solution “cost me study that
robbed me of rest for an entire night”.*

JOHANN BERNOULLI

IN this chapter, we provide an insight into the state of the art of rule-based modelling, including its foundations, existing available formalisms, and established analysis techniques.

The conventional modelling of a biochemical system usually involves drawing a reaction-scheme diagram depicting the chemical species and reactions in the system and then manually translating it into a set of equations [Voioo] (e.g. ordinary differential equations). The limit of such an approach is that the models derived in this way are usually based on assumptions and excessive simplifications. Lifting the simplifications causes a combinatorial explosion in the number of possible reactions and species [Bli+o6], which makes manual modelling impractical.

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

The so-called *computational approach* [Caro8; FHo7a] offers an abstraction of the information about individual model components and interactions, providing a way to avoid the combinatorial explosion on the level of model specification. A promising computational approach is provided by *rule-based modelling* [DKo7; Har+16]. Rule-based models form a natural extension of the mechanistic reaction-based models used in biochemistry while avoiding the combinatorial explosion using compact representation.

The formal foundations of rule-based modelling have roots in rewriting systems [And+13; RSo2; BKVo3; DJ90] and process-algebraic frame-

works [GBHo9; CHo8; Caro8; CHo9; Bri+96; Bri+01]. Both approaches have been proposed for describing features of distributed and concurrent systems, and as it was shown [Bis+03b], they are related. Therefore, it is not surprising that the origin of various rule-based formalisms can be traced to one of them.

The formal study of rewriting systems and their properties was established by Axel Thue in the early 20th century, later summarised in [Boo87]. The earliest models of computation were based on notions of rewriting strings or terms, leading to the study of rewriting in the context of programming language semantics. A pioneering formalism was λ -calculus [Alo41], playing a crucial role in mathematical logic with respect to formalising the notion of computability.

The attractiveness of rewriting systems and process algebras rests in the simplicity of their syntax and semantics, facilitating suitable mathematical analysis. Moreover, they provide a natural medium for implementing parallel computations in general. These are the principal reasons why they were also adopted in systems biology to capture concurrency in biology and enable computation in cell [DKo7; RSo2; FB96; GHBo8].

One of the first used algebraic approaches was π -calculus [Mil99], a general and minimalist model-specification language originally designed to capture essential features of concurrent and distributed systems in computer science. Use of π -calculus to model biological interactions was suggested in [RSSoo]. Another example is multiset rewriting systems (MRSs) [Mes92], also with several applications for describing biological systems [Bar+08; Bis+03a]. MRS is a special case of term rewriting where the molecules can be present in multiple copies with no particular arrangement, therefore represented by *multisets*.

Such low-level formalisms can serve as a base to build higher-level languages such as membrane computing [Pău07], Petri nets [Cer94], or rule-based modelling [Bou+18b; DHW13; Har+16]. However, by themselves, they are quite minimalist languages, and some of their notational features are irrelevant and even inappropriate for biological applications. For example, in π -calculus, communication has directionality, but physical association does not. Therefore, researchers have sought to develop more congruent higher-level languages for modelling biochemical systems. Among them, a suitable candidate is a rule-based approach.

With the rule-based approach, instead of operating with concrete *molecules*, it operates with *types* that allow avoiding the combinatorial explosion that occurs when underlying molecules are specified explicitly. The semantics of the model is given in terms of *rules* defined on given types. The rules express a high degree of modularity which helps to avoid the explicit enumeration of all possible molecular species or all the states of a system. They also enable a more

natural starting point for model development than making *ad hoc* assumptions to decide the model scope. An essential advantage of rule-based models is that mathematical models can be automatically generated from them. In particular, instead of relying on a single mathematical formalism, different mathematical models can thus be obtained for a given model (e.g., ODEs [CFL17], PDEs [And16], chemical master equation or continuous-time Markov chains [Pau+10; SFE11], reaction-diffusion systems [Sor+13], etc.).

2.1 RULE-BASED LANGUAGES

There are several rule-based languages dedicated to the modelling of biological systems. Each of them uses different features and abstractions. In this section, we will highlight the key features of the most popular representatives and describe the analysis methods established for them.

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

2.1.1 Kappa language

The Kappa language [DK07] was primarily developed for the modelling of protein–protein interactions in a graph-rewriting setting. The key components used in the language are *agents* formally representing the protein as a node with a fixed number of *binding sites*, allowing it to form a complex as a connected graph built over such nodes. Each binding site must be unique, with at most one bond. Additionally, each site can occur in one of several pre-defined states.

The Kappa *rules* are changing properties of the agents. Particularly, the rule might add, delete, or change a bond or a state of one or multiple agents at once. The rules are patterns with two sides where each of them is a sequence of agents delimited by a comma. The introduction of sites allows to group multiple reactions to a single rule by omitting the context which is not relevant. This is a typical way of how rule-based languages encode reactions to rules, and it ensures conciseness of description. An example of a Kappa model is in Figure 5.

2.1.2 BioNetGen language

The BioNetGen language (BNGL) [Har+16] is similar to the Kappa language, but there are several differences. The basic idea of the language is to use molecules to describe the building blocks of a biolog-

```

// Signatures
%agent: A(x,c)           // Declaration of agent A
%agent: B(x)             // Declaration of B
%agent: C(x1{u p},x2{u p}) // Declaration of C

// Variables
%var: 'on_rate' 1.0E-3    // per molecule per second
%var: 'off_rate' 0.1      // per second
%var: 'mod_rate' 1        // per second

// Rules
// A and B bind and dissociate:
'rule 1' A(x[.]), B(x[.]) ->
    A(x[1]), B(x[1]) @ 'on_rate', 'off_rate'

// AB binds unphosphorylated C:
'rule 2' A(x[.],c[.]), C(x1{u}[.]) ->
    A(x[.],c[2]), C(x1{u}[2]) @ 'on_rate'

// site x1 is modified:
'rule 3' C(x1{u}[1]), A(c[1]) ->
    C(x1{p}[.]), A(c[.]) @ 'mod_rate'

// A conditionally binds C:
'rule 4' A(x[.],c[.]), C(x1{p}[.],x2{u}[.]) ->
    A(x[.],c[1]), C(x1{p}[.],x2{u}[1]) @ 'on_rate'

// site x2 is modified:
'rule 5' A(x[.],c[1]), C(x1{p}[.],x2{u}[1]) ->
    A(x[.],c[.]), C(x1{p}[.],x2{p}[.]) @ 'mod_rate'

// Observables
%obs: 'AB' |A(x[x.B])|
%obs: 'Cuu' |C(x1{u},x2{u})|
%obs: 'Cpu' |C(x1{p},x2{u})|
%obs: 'Cpp' |C(x1{p},x2{p})|

// Initial condition
%init: 1000 A(), B()
%init: 10000 C(x1{u},x2{u})

```

Figure 5: An example of a Kappa model. It contains the declaration of agents (signature), definitions of rules and constants, as well as observable variables summing up molecules of interest. These can be displayed in the simulation plot in the support tool KaSim [Bou+18b].

ical system, such as proteins, genes, and metabolites. Each molecule has a number of sites with associated internal states that are used to represent the status of post-translational modifications or bindings with other molecules. The rules describe the interactions among molecules, including associations, dissociations, modifications to the internal state of a site, as well as the production or consumption of molecular species. Rules also provide rate laws for transformations resulting from molecular interactions. Patterns are used to identify a set of molecules that share the same internal states. The rule can be specified using patterns such that it defines not a single reaction but a potentially large class of reactions, all involving a common transformation parametrised by the same rate law. BNGL rule typically represents molecular interactions and the consequences of these interactions.

An example of a model is given in [Figure 6](#). Software tools have been developed to construct, visualise, simulate, and analyse BNGL models [[Har+16](#); [SFE11](#); [Wen+14](#); [Xu+11](#)].

2.1.3 PySB

The PySB language [[Lop+13](#)] is an open-source programming framework written in Python that allows concepts and methodologies from contemporary software engineering to be applied to the construction of transparent, extensible, and reusable biological models. It is implemented as a package in Python programming language.

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

The core of the language is defined by translating to BNGL. PySB is closely integrated with Python numerical tools for simulation and graphical tools that enable plotting model trajectories and topologies. The main advantage of the language is that it can be used to decompose models into reusable macros that can be independently tested and then used to generate composite models. An example of a model is given in [Figure 7](#), denoted in Python syntax with the usage of some special PySB classes.

2.1.4 Chromar

The Chromar language [[HZ+17](#)] allows defining attributes for agents and range them over pre-defined domains. The qualitative seman-

```

begin model
  begin parameters
    A0 100 # Initial number of A molecules
    B0 100 # Initial number of B molecules
    ka 0.01 # A-B association rate constant (1/molecule 1/s)
    kd 1    # A-B dissociation rate constant (1/s)
  end parameters
  begin molecule types
    A(b,b)
    B(a)
  end molecule types
  begin seed species
    A(b,b) A0
    B(a) B0
  end seed species
  begin observables
    Molecules FreeAsites A(b)
    Molecules FreeB B(a)
    Molecules ABbonds A(b!1).B(a!1)
    Species BAB B(a!1).A(b!1,b!2).B(a!2)
  end observables
  begin reaction rules
    A(b) + B(a) <-> A(b!1).B(a!1) ka, kd
  end reaction rules
end model

```

Figure 6: An example of a BNGL model. The simple binding model with a bivalent A molecule that has two identical sites for binding of B. Binding at each site is not affected by the status of the other site (noncooperative binding).

tics is given by rule match on multisets composed of these agents producing a reaction. It is followed by standard application of the reaction (in the manner of multiset operations). The language is handy when creating a model where we need to create new distinct objects and control the population of these objects.

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

We would like to highlight a feature which allows the assignment of stochastic semantics to the models written in this language. Compared to the other rule-based languages, it is capable of specifying the rates for individual reactions inherited from the rule. It is allowed by variable value bindings and type-determination between the left

```

# import the pysb module and all its methods and functions
from pysb import *

# instantiate a model
Model()

# declare monomers
Monomer('C8', ['b'])
Monomer('Bid', ['b', 'S'], {'S': ['u', 't']})

# the parameter values
Parameter('kf', 1.0e-07)
Parameter('kr', 1.0e-03)
Parameter('kc', 1.0)

# the rules
Rule('C8_Bid_bind',
     C8(b=None) +Bid(b=None, S='u') | C8(b=1) % Bid(b=1, S='u'), kf, kr)
Rule('tBid_from_C8Bid',
     C8(b=1) % Bid(b=1, S='u') >>C8(b=None) +Bid(b=None, S='t'), kc)

```

Figure 7: An example of a PySB model. The rule name can be any string. The species are instances of monomers in a specific state. In this case, we require that C8 and Bid are both unbound, as we would not want any binding to occur with species that are previously bound. The complexation or addition operator $+$ tells the program that the two species are being added, that is, undergoing a transition, to form a new species as specified on the right side of the rule. The forward/backward operator $|$ states that the reaction is reversible. Finally, the binding operator $\%$ indicates that there is a bond formed between two or more species. This is indicated by the matching integer (in this case 1) in the binding site of both species along with the binding operator.

and right-hand sides of the rule. An example of a Chromar model is available in [Figure 8](#), written in Haskell syntax.

2.1.5 Language for Biochemical Systems

Language for Biochemical Systems (LBS) [[PP10](#)] combines rule-based approaches to modelling with modularity. The main features of the language are species expressions for manipulating large complexes in a concise manner, parameterised modules with a notion of subtyping for writing reusable modules, and non-determinism for handling combinatorial explosion.

A simple example of an LBS program is shown in [Figure 9](#). It shows the mechanism involving the receptor activation and G-protein cycle modules and demonstrates how modules can return patterns. The enzymatic reactions are represented using the tilde operator (\sim) with an enzyme (a pattern) on the left and a reaction on the right. Reversible

```

import Chromar

-- Agent declarations
data Agent = A { x :: Int }
              deriving (Eq, Show)

-- Rules
r1 = [rule | A{x=x}, A{x=y} -->
        --> A{x='x+1'}, A{x='y-1'} @'1.0' ['y > 0'] []]
r2 = [rule | A{x=x} -->
        --> A{x='x'}, A{x='0'} @'1.0' []]

na = [er | select A{x=x};
        aggregate (count . 'count + 1') '0' []]
nx = [er | select A{x=x};
        aggregate (count . 'count + x') '0' []]

s = ms [A{x=5}, A{x=5}]

model = Model { rules = [r1, r2], initState = s}

```

Figure 8: An example of a Chromar model. Arithmetic operations can be used in order to change the properties of individual agents. Moreover, rate and conditional functions increase the applicability and practical use of rules.

reactions are represented using a double-arrow (\leftrightarrow) followed by rates for the forward and backward directions. The rate v_{46} in the G-protein cycle module is defined explicitly because it does not follow mass-action kinetics, and this is indicated by the use of square brackets around the forward rate of the reaction.

The definition of modules and separated identifiers within them do not in themselves hold any identity of a species, and one may bind the same identifier to an entirely different species in another part of the program. This allows different modules, possibly developed by different people, to use the same species identifier for molecules which are semantically and biologically different and subsequently combine the modules into a single program without unintended cross-talk.

In [PP10], authors provide a demonstration of multiple concrete semantics defined for the rule-based description, namely basic Petri nets, coloured Petri nets, ODEs, and CTMCs. For that, they usually use a reaction-based (or ground) representation of the system (except coloured Petri nets, which allow most of the rule-based abstract features to be encoded in colours).


```

spec Fus3{p:bool};
module ReceptorAct(comp cyto, pat degrador, patout rl) {
  spec Alpha, Ste2{p: bool};
  // pheromone and receptor degradation:
  degrador ~ Alpha -> {k1} | cyto[Ste2{p=ff}] -> {k5} |
  // Receptor-ligand binding and degradation:
  Alpha + cyto[Ste2{p=ff}] <->
    {k2,k3} cyto[Alpha-Ste2{p=tt}] as rl;
  cyto[rl] -> {k4}
};

module GProtCycle(pat act, patout gbg) {
  spec Ga, Gbg, Sst2{p:bool};
  pat Gbga = Gbg-Ga-GDP;
  // disassociation of G-protein complex:
  act ~ Gbga -> {k6} Gbg + Ga-GTP |
  // ... and the G-protein cycle:
  Ga-GTP -> {k7} Ga-GDP |
  Sst2{p=tt} ~ Ga-GTP -> {k8} Ga-GDP |
  rate v46 = k46 * (Fus3{p=tt}^2 / (4^2 + Fus3{p=tt}^2));
  Fus3{p=tt} ~ Sst2{p=ff} <-> [v46]{k47} Sst2{p=tt} |
  Ga-GDP + Gbg -> {k9} Gbga |
  pat gbg = Gbg; Nil
};

spec Bar1;
comp cytosol inside T vol 1.0;
ReceptorAct(cytosol, Bar1, pat link);
GProtCycle(link, pat link2);

```

Figure 9: An example of LBS program with receptor activation and G-protein cycle modules. The receptor activation module takes parameters for the cytosol compartment and a pattern which degrades the pheromone. The G-protein cycle module has an empty type, indicating that the module does not care about the contents of this pattern.

Moreover, LBS-Kappa [PPP15] is a modular extension of the Kappa language based on LBS, providing a language for writing high-level and modular rule-based models. Since LBS is a general framework, it was only needed to define the necessary instantiation to Kappa and refer to the general semantics of LBS.

2.1.6 SBML-multi package

Systems Biology Markup Language (SBML) [Huc+03] is a standard established for systems biology based on XML. It is a popular standardised format for the electronic storage, exchange, and reuse of math-

ematical models of biochemical systems. However, it is based on the assumption that a model can be specified adequately in terms of a reaction scheme, which is not suitable for a rule-based model.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3" level="3" 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 10: An example of a simple SBML-multi model. The model requires specification of language level (version). In the definition, there are included SpeciesTypes, Species which belong to these types, and Reactions where those species are interacting. Additionally, all processes are encapsulated in the compartments.

The newest version, SBML Level 3 [Kea+20], introduced modular language extension capability, which allows different language features to be added to a common language core. An SBML-multi package [ZMS18] is able to describe all the necessary rule-based features, and therefore, it is possible to export each model in a rule-based language in this format. It is the most suitable format for the exchange and storage of the models but less for analysis and direct presentation to the users. It serves as an intermediate format for model sharing. The presence of a feature tag informs a software tool reading the model that the model uses that particular feature and permits the tool to quit if it does not have the necessary interpretive capabilities. For example, see Figure 10.

2.2 ANALYSIS METHODS

An important advantage of the rule-based approach is that mathematical models can be automatically generated from it. In particular, instead of relying on a single mathematical formalism, different mathematical models can thus be obtained for a given model (e.g., ODEs [CFL17], chemical master equation or continuous-time Markov chains [Pau+10; SFE11]). This fact results in multiple possible ways of how to analyse and explore the models.

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

Model checking [CJ+18] is an automated way of checking whether the model meets a given specification. For that, we need a specification language which allows us to express a property, typically dependent on a notion of time. A suitable tool to use is a *temporal logic*. Examples of logics which are often used in systems biology are LTL [Pnu77], CTL [CES86], STL [MNo4], or PCTL [HJ94] and their usage depends on particular mathematical representation we use. Generally, given a rule-based model \mathcal{B} and a property formula ϕ , the *problem of model checking* is to decide whether the model \mathcal{B} satisfies the property ϕ .

In the case of unknown parameter values present in the model, we can be interested in finding those parametrisations from the parameter space which satisfy the given property. For that, a method called *parameter synthesis* is used. The basic principle is to search parameter space and identify regions where satisfaction is (not) guaranteed. Furthermore, some tools may only produce incomplete results – regions where no guarantees can be produced by the chosen search method. More specifically, given a parametrised rule-based model \mathcal{B}

and a formula ϕ , the *problem of parameter synthesis* is to compute a partitioning of the given parameter space into three disjoint subsets: TRUE – the parameter values satisfying the property, FALSE – the parameter values violating the property, and UNKNOWN – the result is not known.

Additionally, instead of the boolean answer to the satisfaction of a property, a measure of how much the property is preserved (resp. violated) in the given parametrisation can be useful. This measure is called *local robustness* $D_{\phi}^{\mathcal{B}}(p)$ and states how much the property ϕ is preserved in parametrisation p of model \mathcal{B} . The *problem of global robustness* of a model \mathcal{B} is defined as

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

where ϕ is the property of the system under scrutiny, P is the parameter space, $\psi(p)$ is the probability of the parametrisation p , and $D_{\phi}^{\mathcal{B}}(p)$ is the local robustness. The *global robustness* [Kito7] defines a measure of how the given property is globally preserved in the model with respect to the parameter space.

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

Rule-based models can be used to generate other computational models using different semantics. Therefore, many developed analysis techniques for the other computational models can be indirectly used to analyse the rule-based models. A disadvantage of such a procedure is that the computational models are often explicit in enumerating the system reactions, and thus the implicit representation of the rule-based approach is lost during the translation.

2.2.1 Simulation

The most usual way how to exploit the behaviour of a rule-based model is a simulation. While qualitative simulation is also possible (generating a successor of the current state while abstracting the notion of time), a more usual and closer to the reality is a quantitative simulation. Using the generated reaction network from the model rules, one can perform stochastic simulation, for example, using Gillespie algorithm [Gil76], or deterministic simulation [P00] by consequent translation to ordinary differential equations (ODEs) [Higo8]. However, the cost of simulating the kinetics of a reaction network depends on the size of the network. Thus, if a network is large, the

simulation cost can be expensive in terms of computation time or computer memory. To address this problem, network-free simulators were developed that take advantage of rules. The method generalises an agent-based kinetic Monte Carlo method that has been shown to circumvent the combinatorial bottleneck in simulations [Yan+08].

An example of a network-free simulator is NFsim [SFE11]. The advantage of NFsim compared to reaction-based approaches is that during simulation, rules operate directly on molecular objects to produce exact stochastic results with a performance that scales independently of the reaction network size. Reaction rates can be defined as arbitrary functions of molecular states to provide powerful coarse-graining capabilities. NFsim enables researchers to simulate many biological systems that were previously inaccessible to general-purpose software.

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

2.2.2 Model checking

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

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

- *Reachability*. The reachability problem is to check whether there exists a path from a given state to a particular state. The problem can be reformulated to finding a sub-state, meaning we are only interested in the presence of particular species. It has been shown that the reachability problem is decidable [Kos82] (for the family of vector addition-based systems) although it takes at least exponential space (and time) to verify in the general case [Lip76].

- *Boundedness*. A system is bounded if the number of molecular counts is finite for any state reachable from the given initial state. This is useful for the detection of undesirable inconsistency in the flows in the model. It is also generally required by model checking.
- *Persistence*. A system is persistent if, for any two reaction events, the firing of one reaction will not disable the other. This analysis is suitable for detecting concurrent reactions in the models, which are often responsible for some critical decision-making in biological systems.

Generally, checking a property of a model is called *model checking*. Model checking is an automatic formal verification technique able to perform a clever exhaustive search of the state space of a model. It typically operates over a discrete-time model with a finite number of states (usually a Kripke structure [Kri63]), determining the truth value of a temporal logic formula which specifies desired property of the modelled system. Examples of logics which are often used in systems biology are LTL, CTL, STL, or PCTL, depending on the particular mathematical formalism.

A rule-based model can be translated to such a formalism and analysed by an efficient model checker (e.g. [Cim+02]). An important issue with this technique is that the number of states of a model usually grows exponentially in the number of its variables, giving rise to the state explosion problem. Additionally, for various rule-based languages, the state space of the rule-based model is infinite, making the analysis even harder. For this reason, there are often some limitations in the application of the rules which bound possible state space (e.g. global bound on molecular counts in states), abstracting some information and providing a way how to limit the state space to a finite size.

For these reasons, most of the model checkers usually represent the states symbolically [Bur+92], for example, by using binary decision diagrams [Bry01]. Relevant methods for the analysis of continuous semantics are based on the abstraction of the behaviour to a finite number of discrete states. For example, one can be interested in parallel semi-symbolic coloured model checking [Ben+16], bifurcation analysis [Ben+17], and attractor analysis [Ben+18].

Model checking has been extended to many other computational models, such as continuous and discrete-time Markov chains (CTMC and DTMC, respectively), by adding probabilities [KNP10]. Powerful and efficient tools for analysing the Markov chains [KNP11; Deh+17] allow calculating the probability that the model will satisfy the property of interest. They can either provide an exact solution [Azi+00] or an approximated solution [Jha+09] (using a set of samples generated using a Monte Carlo simulation of the model). Moreover, it is possible

to compute a numerical value expressing the measure of satisfaction instead of a binary decision, often referred to as *quantitative model checking* [HK97].

Some model checking algorithms rely on statistical hypothesis testing, thus using *statistical model checking*. The idea is to check the property on a sample set of simulations and to decide whether the system satisfies the property based on the number of executions for which it holds compared to the total number of executions in the sample set. With such an approach, it is not necessary to consider all the executions of the system. Authors in [Cla+08] applied this technique to BioNetGen for probabilistic bounded linear temporal logic.

On-the-fly algorithms employ a top-down approach to model checking, which does not require global knowledge of the complete state space [BCG95; Cou+92; GM11; Holo4; LLM14]. Instead, the algorithms construct step by step local knowledge of the state space until it is possible to decide whether the given state satisfies the formula.

2.2.3 Parameter synthesis

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

In [Bar+11], the authors present an algorithm for parameter synthesis based on parallel model checking, which is conceptually universal with respect to the modelling approach employed. In [Bat+10], an efficient method to analyse large and possibly incomplete parametrised piecewise-affine differential equation models is proposed. The usage of a symbolic encoding of the model structure enables the methods to take full advantage of symbolic model checkers for testing CTL dynamic properties. An algorithm for computing parameter synthesis in non-linear dynamical systems [DCL10] extends formal verification techniques that were first introduced in the context of continuous and hybrid non-linear dynamical systems [DMo7].

In [JL11], algorithms for studying the parameter space of stochastic biochemical models were presented. Methods for parameter synthesis of parametric Markov chains have been introduced with symbolic computation of reachability properties through state elimination [Dawo4; HHZ11; Jan+14], improved by parameter lifting [Qua+16] and fraction-free Gaussian elimination [Bai+20].

Similarly to model checking, an alternative approach to the analysis of complex stochastic models under parameter uncertainty is based on statistical methods [BBW18; BMS16; BS18; LW16]. There are only a few works that bridge the rule-based framework to such tech-

niques. In [LF16; Hla+19], a statistical parameter sampling method is employed to analyse unknown parameters in BNGL models represented by means of CTMCs where the rate function is limited to mass action kinetics. The recent work [KJ18] combines statistical model checking with machine learning techniques to calibrate parameters in order to maximise the probability of satisfying a given specification. In [Boc+15], the authors adapt simulation-based and moment-based methods. In general, statistical techniques do not give an exact representation of satisfying parameter sets.

2.2.4 *Monitoring*

Another way to tackle the state explosion problem is the analysis of a single execution trace instead of performing an exhaustive verification. *Monitoring* aims to check whether the current execution of a program satisfies or violates a property of interest. It is particularly suitable for observing the “live” behaviour of a system. For example, in BIOCHAM [CFS06], authors implemented monitoring of numerical simulations of biological models with LTL used to specify properties of real-valued variables. Additionally, in [Don10], they implemented a tool where the evaluation of the STL [MNo4] formula robustness is used for a particular trajectory through monitoring.

2.2.5 *Robustness analysis*

According to [Kit04], robustness is a property that allows a system to maintain its functions against internal and external perturbations. The concept of robustness is well established for deterministic systems [Don+11; Riz+09]. The evolution of a stochastic system is given by a set of paths compared to a single trajectory of a deterministic system. The stochastic system at any given time is described by a probability distribution over states of the corresponding CTMC in contrast to the single state representation of a deterministic system. Therefore, the definition of robustness for stochastic systems requires a more sophisticated approach. In [Čes+14], authors developed an adaptation of the concept of robustness to stochastic systems, particularly robustness analysis in CTMCs. The method is based on a numerical approximation of the evaluation function [Bri+13], which provides accuracy guarantees in contrast to existing methods employing parameter sampling and statistical techniques.

2.2.6 *Static analysis*

Static analysis is performed on the specification of the model without the need to actually execute it [NNH15]. While model checking generally needs to explore all the states originated by executing the se-

mantics of the model, static analysis operates on the syntactic level of the specification or by using abstract interpretation [CC77] over finite approximations of the possible model executions [PMR12]. It is a powerful tool used to detect potential issues in the model.

There are several well-defined static analysis methods for rule-based systems, which can be useful in finding inconsistency in the models [Dan+09]. For example, we mention *detection of dead rules* — a rule is called dead if there is no trace starting from the initial state in which this rule is applied; *detection of dead agents* — an agent is called dead if there is no trace starting from the initial state with at least one state in which this agent occurs; *detection of redundant rules* — a rule is redundant if after removing it from the model, the particular semantics do not change. These and many other methods can be run statically, providing an efficient way how to verify special types of properties before executing the model. However, it is worth mentioning that such methods are often with limited guarantees, providing only an incomplete solution.

To the best of our knowledge, the only tool dedicated to static analysis of rule-based models is KaSa [Bou+18a]. This static analyser abstracts the set of reachable states of models and then uses this information to collect insightful properties. In particular, KaSa may warn about rules that may never be applied, about potential definitive transformations of proteins, and about the potential formation of unbounded molecular compounds.

2.2.7 Regulations

Although it is not directly an analysis technique, some variant of *control mechanism* belongs to the category of resolving the uncertainty in biological models. Instead of finding parameters that ensure the correct behaviour of the model, we enforce the target modelling intentions by an intervention on the qualitative level.

While control theory [BYG17; Shm+17; Ram+14; KF09; SSVS13] can have various forms and implementations, we focus on methods that can be integrated directly into the model specification and thus serve as a part of system description on an abstract level. This is motivated by biological cases, where a mechanism is needed to influence that a particular reaction is executed (resp. it is not executed) under certain conditions (e.g. cooling the system, knock-out of a gene).

A suitable instrument for that is *regulation* mechanisms. The regulations can have many different forms, but generally, they provide an additional mechanism for controlling the semantics of the rewriting system by restricting the conditions when a rewriting rule can be executed. The regulations can be beneficial in multiple applications, such as simplifying the system description, substituting the missing details yet to be discovered, or even sketching systems properties in

synthetic biology. It becomes particularly useful when newly discovered biological processes need to be introduced into the system. To incorporate the novel processes into the model, detailed knowledge of them is usually necessary. However, in biology, these processes are not always known and discovering their mechanistic components is a tedious and challenging task. Nevertheless, the effects of such processes are crucial for capturing the dynamic behaviour of the modelled system.

Regulated mechanisms are well-established in the area of rewriting systems [Das04] (string grammars in particular) and have been extended to several modelling formalisms such as Petri nets [IH88] or membrane computing [FMVPO4]. In [Del02], authors introduced an approach to regulate MRS using first-order logical formulae over variables, allowing to express constraints to rules application. While this approach significantly increases the expressive power of MRS, it does not provide a suitable solution for applications such as reasoning about the type of behaviour based on relationships among rewriting rules, modelling of the unexplored processes in the early phase of their research, and also does not meet the usability requirements set by the target audience (biologically-oriented community) for its high demands on technical details.

Above all, to the best of our knowledge, regulations have not yet been introduced to rule-based modelling.

2.3 SUMMARY

This chapter summarised the existing rule-based languages and established them in the context of more general formal methods. It is worth commenting on the disadvantages of existing rule-based approaches in the context of their usability as well as supported analysis methods.

The general problem present in the Kappa and BNGL languages is a too detailed description on the level of bonding. It is often difficult for biological systems to manage bonds between sites, especially when such details are not needed. While PySB and Chromar have their unique features and advantages, they both require at least basic knowledge of a programming language (Python and Haskell, respectively). Moreover, in the case of Chromar, all the biologically relevant terms, such as states, have to be encoded in natural numbers. Finally, while the modularity of LBS can be beneficial, it can also be a downside for its complicated structure and abstract approach.

Additionally, as we will see in the following sections, compared to languages presented here, BCSL offers some unique features that can be beneficial in the modelling of biological processes. For example, it does not employ explicit bonding of molecules, relaxing the notation and complicated structure of macromolecules. Complexes represent

only a coexistence of molecules without enforcing any particular biochemical meaning. Also, the usage of complexes in reaction rules is rather unique. Rules are always applied to the whole complexes, in contrast to the usual context-free approach allowing to modify subparts of complexes in other languages. While this feature is limiting in some sense, it also provides a safety policy for unintentional side effects of rules.

Analysis support by software tools of existing languages is very individual but generally does not cover the standard analysis base established in systems biology [Eng+09], especially when dealing with uncertainty in models such as unknown parameters and employment of exact formal methods. Finally, expressing the uncertainty is also limited, with, for example, no support for regulation mechanisms allowing to quickly sketch complicated details of the modelled system.

*Although to penetrate into the
intimate mysteries of nature and
thence to learn the true causes of
phenomena is not allowed to us,
nevertheless it can happen that a
certain fictive hypothesis may suffice
for explaining many phenomena.*

LEONHARD EULER

BIOCHEMICAL Space Language (BCSL) is a language for modelling biological systems employing rule-based features. This chapter introduces BCSL on an informal level, followed by its formal definition. Then, we show how the language is related to multiset rewriting systems (MRS). In particular, every BCSL model corresponds to an MRS that preserves its behaviour, and we show how such MRS can be constructed. Finally, we show how BCSL can be directly implemented by embedding it in MRS.

3.1 INTRODUCTION TO BIOCHEMICAL SPACE LANGUAGE

In this section, we first introduce the language on an informal level. Since this thesis is centralised around BCSL, it is crucial for the reader to gain intuition and biological context behind the language.

The general goal of the language is to deal with the combinatorial explosion of numerous interactions by providing a concise and understandable notation. To capture fundamentally different features of biological objects, we introduce several types of the *agents*. We show how we understand and use *rules* to describe the transformations of agents. Finally, we provide intuition behind the semantics of the rules.

3.1.1 Agents

In existing biochemical ontologies, objects residing in several different conditions (oxidised, reduced, etc.) are usually treated as separate objects, thus causing the total number of objects to be enormous. To reduce this complexity, the concept of *states* is used in BCSL.

To hold information about the internal state, *atomic agent* is used. It describes the most basic type of biochemical objects, usually small

molecules. It carries information about the name of the agent and the associated state.

The atomic agent is denoted with its name followed by the state in curly brackets. For example, in the respective line of Table 1, there is a carbonate in a 2- state and a serine amino acid residue in a methylated state.

Structure agent represents a biochemical object composed of several known atomic agents, while we know that composition is abstract and not necessarily complete. To incorporate this kind of abstraction into our language, a structure agent is labelled with a unique name. The key construct of a structure agent is *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.

BCSL object	Example(s)
Atomic agent	$\text{CO}_3\{2-\}, \text{Ser}\{\text{met}\}$
Structure agent	$\text{Cas}(\text{Ser}\{\text{p}\}, \text{Tyr}\{\text{u}\})$
Complex agent	$\text{Cas}(\text{Ser}\{\text{p}\}, \text{Tyr}\{\text{p}\}) . \text{Cas}(\text{Ser}\{\text{p}\}, \text{Tyr}\{\text{p}\})$
Compartment	$\text{Cas}(\text{Ser}\{\text{p}\}, \text{Tyr}\{\text{p}\}) . \text{Cas}(\text{Ser}\{\text{p}\}, \text{Tyr}\{\text{p}\}) :: \text{loc}$

Table 1: Examples of fundamental BCSL objects. The usage of atomic, structure, and complex agents, including compartment, is demonstrated on simple biochemical molecules.

The structure agent is denoted with its name, followed by an enumeration of its atomic agents in round brackets. For example, in casein protein (Cas), two out of a few hundred amino acid residues (e.g. serine and tyrosine) can undergo some post-translational modifications, such as phosphorylation or methylation. It is suitable to model only these two acids instead of the entire primary structure of the protein. In the respective line of Table 1, there is the Cas protein with phosphorylated serine and unphosphorylated tyrosine subparts.

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

The key element of a complex agent is a *sequence* describing inductively constructed coexistence expressions from existing (atomic and structure) agents. In contrast to composition in structure agent, the employed order of the sequence plays an important role in the language, and we allow replication at the level of sequence (an agent of a certain name can appear more than once in a sequence). The structure agent is denoted by the sequence of agents separated by

a period. An example of a complex agent is a dimer of casein, where all its amino acid residues are phosphorylated (see Table 1).

The agents in BCSL reside in *compartments*. Typical representatives on the level of molecular modelling are cell, cytosol, or nucleus. Each complex agent belongs to one or several compartments. An atomic or a structure agent cannot belong directly to a compartment, but they are always part of a complex agent (the case when only one agent is in its sequence can occur). This guarantees each atomic and structure agent has an indirectly given spatial location – the compartment. The example for a complex agent is extended by `loc` defining the compartment (see Table 1).

Agents are associated with *signatures*, which describe allowed attributes for individual agents. There are three types of signatures. *Atomic signature* defines an allowed set of possible internal states for an atomic agent. For example, for an atomic agent serine Ser, it allows states methylated met, phosphorylated p, and neutral n. *Structure signature* defines an allowed set of atomic agents for a structure agent. For example, for the casein, we define a set of allowed atomic agents tyrosine Tyr and serine Ser. *Complex signature* allows replacing a complex expression with an alias, which increases readability.

3.1.2 Rules

Rule is specified by a *rule equation*, containing identifiers of *substrates* and *products*. Every agent appearing in a rule equation has to be followed by the *localisation* operator associating it with a particular compartment. This is, for example, important for rules that act on both sides of a membrane. That way, a rule is always precisely localised in or between the compartments. A natural *stoichiometric* coefficient can be placed before any agent in a rule equation. Irreversible and reversible rules are distinguished by the operators ' \Rightarrow ' and ' \rightleftharpoons '. The '+' symbol is used as a separator between individual substrates and individual products. Additionally, a rule can be associated with a custom expression representing a kinetic rate law.

The rule does not precisely specify *which* agents can undergo the transformation; it only specifies restrictions on the agents. That is the general feature of rule-based languages.

An example describing the transformation of casein from the methylated tyrosine residue to the neutral one can be expressed by rule **a** from Table 2. It is important to notice that above we defined casein with two subparts Ser and Tyr, but serine Ser is not mentioned in the rule. It basically means its context has no influence on the underlying process, and therefore, Ser can be completely omitted, and the rule can happen regardless of its state.

Another example, rule **b** from Table 2, describes complex formation. While both fibroblast growth factor FGF and its receptor R can have a

a $\sim \text{Cas}(\text{Tyr}\{\text{met}\})::\text{cell} \Rightarrow \text{Cas}(\text{Tyr}\{\text{n}\})::\text{cell}$
b $\sim \text{FGF}()::\text{ext} + \text{R}()::\text{ext} \Rightarrow \text{FGF}().\text{R}()::\text{membrane}$
 $\quad @ \text{ k1} \times [\text{FGF}()::\text{ext}] \times [\text{R}()::\text{ext}]$
c $\sim \text{Cas}(\text{Ser}\{\text{u}\}).\text{Cas}().\text{Cas}().\text{Cas}()::\text{cyt} \Rightarrow$
 $\quad \Rightarrow \text{Cas}(\text{Ser}\{\text{p}\}).\text{Cas}().\text{Cas}().\text{Cas}()::\text{cyt}$
d $\sim \text{Ser}\{\text{u}\}:\text{Cas}():\text{Cas4}::\text{cyt} \Rightarrow \text{Ser}\{\text{p}\}:\text{Cas}():\text{Cas4}::\text{cyt}$

Table 2: Examples of BCSL rules. Examples demonstrate typical representatives of rules such as state change (**a**), complex formation and transport (**b**) additionally associated with a rate expression representing mass action kinetic law, less compact (**c**) and more compact (**d**) variants of a rule with long repetitive complex.

variety of internal states, the association described by the rule can happen regardless of the particular states of both reactants. The complex formation happens in extracellular space (ext), and the resulting complex is attached to the cell membrane, thus the change of compartment. This rule is also associated with a rate expression. Typically, such expressions contain agents in square brackets, representing the number of occurrences of such agents in solution. Any mathematical expression representing a rational function can be then formed using agents, numbers, and basic operations such as addition or multiplication.

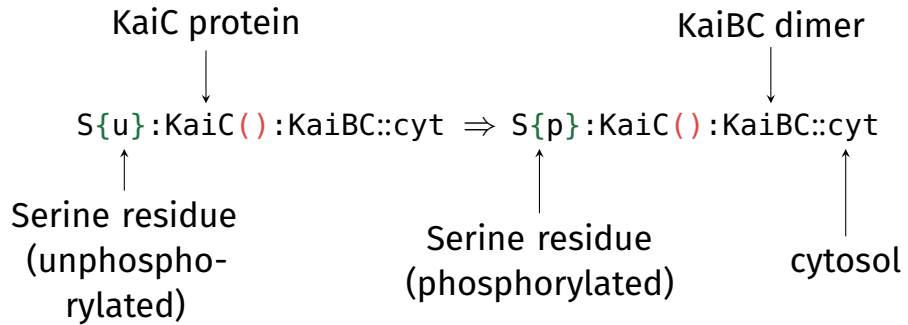


Figure 11: An example of a rule employing localisation operator. The rule describes the change of serine (S) amino acid residue from an unphosphorylated to a phosphorylated state. Additionally, such phosphorylation can happen only when the serine is part of a KaiC protein, which occurs inside a protein complex of KaiC and KaiB proteins. The entire process is allowed only inside of the cytosol (cyt) compartment.

We provide one more example dedicated to complex agents. As mentioned above, complex agents capture the complete enumeration of interacting subparts, which can be atomic or structure agents encapsulated in a compartment. However, it results in one obstacle for the compact syntax of the language – agents in its sequence can be rather numerous. Meanwhile, the process we want to express by

a rule usually involves only a small fraction of the composition. For example, imagine that any protein Cas in the complex of four same proteins (homo-tetramer) can undergo phosphorylation on a serine Ser residue. Normally, we would have to enumerate the whole composition (Table 2 rule **c**), where a single protein changes its state.

This form of description is neither concise nor easily readable. We improve this notation by using information encoded in complex signatures – these associate complex agent names with their sequence. However, we cannot use it directly; we need to introduce the localisation operator ‘.’ between the agents. The main idea is to allow zooming into individual parts of complex and structure expressions.

The rule **d** from Table 2 has the same meaning as the previous one with the assumption we have defined complex signature with name Cas4 (which can be arbitrary). It allows the change of state of one of the Cas proteins inside the complex. An additional and more detailed example is available in Figure 11.

3.1.3 Semantics

In this section, we describe the semantics on an informal level using an artificial example. The precise mathematical definition is provided in Section 3.2.

We represent a BCSL model as a set of rules and an *initial solution* of interacting agents. We understand the solution as a mixture of individual agents with possible repetitions which are randomly distributed (an example is given in Figure 13a). Therefore, we cannot assign them any order, and we represent them as *multisets*. From a biological perspective, this representation of the solution is as close as possible to reality (neglecting spatial information) while preserving conciseness.

$$\begin{aligned}
 \mathbf{a} &\sim A() + B() \Rightarrow A().B() @ k1 \times [A()] \times [B()] \\
 \mathbf{b} &\sim A(S\{u\}).B() \Rightarrow A(S\{p\}).B() @ k2 \times [A(S\{u\}).B()] \\
 \mathbf{c} &\sim A().B(T\{p\}) \Rightarrow A() + B(T\{p\}) @ k3 \times [A().B(T\{p\})]
 \end{aligned}$$

Figure 12: Examples of rules. **a)** An A and a B can form a complex regardless of their internal states. **b)** An A is allowed to change the state of its S from u to p only if it is in a complex with a B, regardless of their other internal states. **c)** The rule can disassemble the complex only if a B has T in state p. All rules are associated with mass action rate laws, parametrised by constants k1, k2, and k3.

The rules are patterns that describe the behaviour of a class of agents. 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 agents, while a rule uses patterns to de-

scribe several agents at once. Therefore, a reaction (e.g. Figure 14c) can be seen as a special case of a rule.

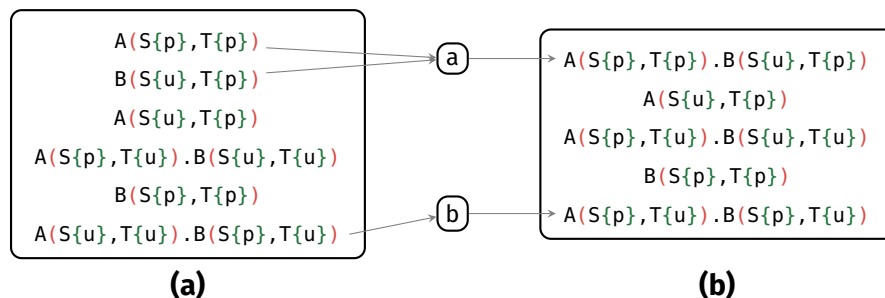


Figure 13: Examples of solutions. The particular order of agents is arbitrary. In solution **a**, there are two separate As, two separate Bs, and two complexes of A and B in various internal states. Solution **b** depicts solution **a** after the rules from Figure 12 were applied to it. The rule **a** from Figure 12 was applied to the top A and B and produced a complex (the top one in solution **b**). Note there are more options for how the rule could be mapped – each combination of free A and B. The rule **b** from Figure 12 was applied to the complex of A and B (the bottom one) where the state of S inside of A was changed from u to p. The rule **c** from Figure 12 could not be applied because there is no such complex with T in state p inside of B.

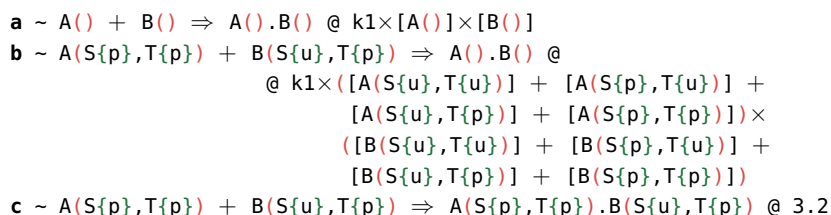


Figure 14: Example of a map-apply action. For this demonstration, we use the solution **a** from Figure 13 and the rule **a** from Figure 12. **a**) The rule can be mapped on any A and B regardless of their particular states and together form a complex. **b**) We randomly choose the first agent A and the second agent B from the solution. The rule was mapped on chosen agents, and they were assigned to the left-hand side of the rule. The abstract agent in square brackets can be represented as a sum of occurrences of all possible corresponding concrete agents (since a single abstract agent corresponds to a set of concrete agents). **c**) The rule was applied, and a new complex of A and B with their particular states was created. We obtained the reaction describing the particular action which has just occurred. The rate expression can be evaluated to a numerical value after substituting molecular counts of agents and assigning a value to the parameter, which is $k1 = 0.8$ for example, thus $k1 \times 2 \times 2 = 3.2$.

The rule is mapped on a solution. Then it can be applied, and a new solution is produced (the so-called *map-apply* action). The mapping can be seen as assigning particular agents from the solution to the respective pattern on the left-hand side of the rule. At the same time, rate expression is evaluated, giving quantitative information about the kinetics of the rule. The mapping is not always successful (Figure 13, application of rule **c**), and there also might be multiple possibilities (caused by the abstraction).

```

#! rules
r1_S ~ P(S{i})::cell ⇒ P(S{a})::cell @ 0.3×[P()::cell]
r1_T ~ P(T{i})::cell ⇒ P(T{a})::cell @ k1×[P()::cell]
r2 ~ P()::cell ⇒ P()::out @ k2×[P()::cell]

#! inits
1 P(S{i},T{i})::cell

#! definitions
k1 = 0.4
k2 = 0.5

```

Figure 15: Example of a simple BCSL model. In this model, a single agent *P* can be modified on its two active sites *S* and *T*. Both sites can be independently activated in the respective rules *r1_S* and *r1_T*. Additionally, the agent can be transported to another compartment outside of the *cell*. All rules have labels placed before the tilde symbol. Initially, there is a single *P* agent present with both sites inactivated (defined in *inits* section). Each rule has defined a rate expression. The used constants in the rate expressions can be named and their values assigned separately in the *definitions* section.

The rule application represents the change of the mapped agents to new agents according to the right-hand side of the rule (i.e., particular agents are assigned to the right-hand side). As a by-product, we obtain an instance of the rule – a reaction (Figure 14). The rate expression can be instantiated as well since, in general, a class of agents can be written as a sum of individual agents.

Starting with the initial solution, we can construct a transition system representing all possible scenarios (*simulations* or *runs*) of how the solution can evolve by applying various rules. This reflects the nondeterministic behaviour of biological systems. Such a transition system can be further used to analyse the behaviour of the BCSL model [Cla97]. In Figure 15, there is an example of a simple BCSL model, and in Figure 16 its transition system.

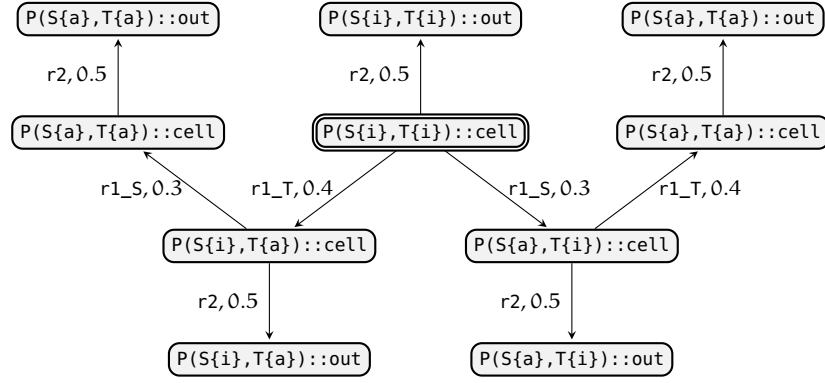


Figure 16: Labelled transition system of the model from Figure 15 in a tree-like representation. The double-circled state is the initial state. The transitions are labelled by a rule which was used to enable the transition and by evaluated rate. Individual model runs correspond to the particular paths starting in the initial state.

3.2 BIOCHEMICAL SPACE LANGUAGE DEFINITION

In this section, we provide a formal definition of BCSL. While the terms established in the previous section are reused here, they are defined on a more formal level. Moreover, since complex agents basically encapsulate atomic and structure agents, we often refer to complexes as the agents, and other types of agents are called atomic and structure *components* (or just simply atomics and structures). We also use the term multiset instead of a solution to highlight its technical properties crucial in the formal definition. This generally applies to the whole thesis if not stated otherwise. This difference in nomenclature is purely technical to highlight the position of complexes in this language in the context of other rule-based languages (where the term *agent* always plays a central role).

```

multiset:  $\emptyset$  | agent | multiset "+" multiset
agent: sequence "::" COMPARTMENT
sequence: component | sequence "." component
component: atomic | structure
structure: S_NAME "(" composition ")"
composition:  $\emptyset$  | atomic | composition "," atomic
atomic: A_NAME "{" FEATURE "}"

```

Table 3: A context-free grammar of core BCSL terms in EBNF [Sco93] notation.

Let $\mathcal{V}_\delta, \mathcal{V}_a, \mathcal{V}_s, \mathcal{V}_c$ be mutually exclusive finite sets of names of features, names of atomic and structure components, and compartments, respectively. In Table 3, we provide a fragment of the complete syntax

of BCSL, capturing agents and multisets, where the terminal (in capitals) $\text{FEATURE} \in \mathcal{V}_\delta$ is from given set of feature names, $\text{A_NAME} \in \mathcal{V}_a$, $\text{S_NAME} \in \mathcal{V}_s$ are from given sets of atomic and structure component names, and $\text{COMPARTMENT} \in \mathcal{V}_c$ is from given set of compartments. We restrict ourselves only to finite expressions and require that an atomic name occurs at most once in a composition. On top of this syntax, several syntactic extensions (see [Section 3.2.3](#) for details) are built, providing more convenient and succinct notation. Details on how BCSL models are technically handled are in [Section 6.1.2](#).

For simplicity, we denote by M a multiset and by \mathbb{M} the set of all multisets. We assume the *structural congruence* \equiv to be the least congruence on terms from [Table 4](#) satisfying respective axioms. That is, two multisets (or any terms) are *equal* if they are structurally congruent.

Term	Satisfying axioms
multisets	$M_1 + M_2 \equiv M_2 + M_1$ $M + \emptyset \equiv M$
sequences	$\text{sequence.component} \equiv \text{component.sequence}$
compositions	$\text{composition, atomic} \equiv \text{atomic, composition}$ $\emptyset, \text{composition} \equiv \text{composition}$

Table 4: Table defining axioms of structural congruence \equiv for particular terms from BCSL grammar.

The structural congruence \equiv allows us to formally define the algebraic multiset operations $\in, \subseteq, \subset, \cup, \cap$ and \setminus on BCSL multisets. For example, $\text{agent} \in M$ corresponds to $\exists M' \in \mathbb{M}. M \equiv \text{agent} + M'$ and $M \subseteq M'$ corresponds to $\exists M'' \in \mathbb{M}. M' \equiv M + M''$. Moreover, by $M(\text{agent})$ we denote the number of occurrences of agent in the multiset M .

To associate atomics with allowed features and structures with allowed atomics, we employ signature functions. *Atomic signature* $\sigma_a : \mathcal{V}_a \rightarrow 2^{\mathcal{V}_\delta}$ is a function from an atomic name to a non-empty set of feature names. The set of possible atomic signatures is denoted as Σ_a . *Structure signature* $\sigma_s : \mathcal{V}_s \rightarrow 2^{\mathcal{V}_a}$ is a function from a structure name to a set of atomic names. The set of possible structure signatures is denoted as Σ_s .

Let $\mathbb{V}_\delta = \mathcal{V}_\delta \cup \{\varepsilon\}$ be a set of feature names extended by a special symbol ε . *Pattern* T is defined according to the same grammar as multiset except FEATURE is from set \mathbb{V}_δ , i.e. it is allowed to use ε . We denote by \mathbb{T} the set of all patterns.

The two patterns are equal if they are structurally equal (the congruence relation defined on multisets does *not* apply). Finally, a pattern is *well-formed* if the atomics are alphanumerically sorted in com-

positions with respect to their names. From now on, we assume only well-formed patterns.

Note that in the following text, there is often a situation when a pattern T is compared to a multiset M . In such a case, we treat the pattern as a multiset, too (i.e. they are equal if they are structurally congruent according to Table 4). Moreover, it holds that $\varepsilon \neq \delta$ for any $\delta \in \mathcal{V}_\delta$.

An *instantiation function* $\mathcal{J} : \mathbb{T} \rightarrow \mathbb{M}$ assigns to every atomic A in \mathbb{T} with feature ε a feature $\delta \in \sigma_\alpha(A)$. By $\Gamma(T)$ we denote a finite set of all possible *instantiations* $\mathcal{J}(T)$ of pattern T .

With $d(T)$ we denote *deatomisation* of pattern T , which corresponds to a sequence of atomics preserving the order of their occurrence in the pattern. Note that this applies to atomics in both sequences and compositions.

Let us have two finite patterns T, T' with their deatomisations:

$$d(T) = A_1, A_2, \dots, A_n \text{ and } d(T') = A'_1, A'_2, \dots, A'_m$$

and two instantiations $\mathcal{J}(T) \in \Gamma(T)$ and $\mathcal{J}(T') \in \Gamma(T')$ with their respective deatomisations:

$$\begin{aligned} d(\mathcal{J}(T)) &= \mathcal{J}(A_1), \mathcal{J}(A_2), \dots, \mathcal{J}(A_n) \text{ and} \\ d(\mathcal{J}(T')) &= \mathcal{J}(A'_1), \mathcal{J}(A'_2), \dots, \mathcal{J}(A'_m). \end{aligned}$$

The instantiations $\mathcal{J}(T), \mathcal{J}(T')$ are *consistent*, written $\mathcal{J}(T) \Delta \mathcal{J}(T')$, if $\forall i < \min(m, n)$ holds that $A_i = A'_i \Rightarrow \mathcal{J}(A_i) = \mathcal{J}(A'_i)$. Consistency of two instantiations ensures that the same features are assigned to both patterns at the same positions.

Pattern expansion is a function $\langle _ \rangle : \mathbb{T} \rightarrow \mathbb{T}$ which extends a given pattern T to a pattern $\langle T \rangle$ such that every occurrence of a composition of a structure is extended by atomics whose names are not yet present in the composition and are defined in the given signature $\sigma_s(\text{structure})$. These newly added atomics have assigned feature ε and are inserted into the composition in such a way that they preserve the alphanumerical order.

Next, we define the grammar for the algebraic rational *rate expression* f . We denote by \mathbb{F}_p the set of all rate expressions. For the sake of readability, we allow additional simplifications (e.g. parentheses), which can always be converted to a form given by the provided grammar:

$$\begin{array}{ll} \text{rate expression} & f ::= \frac{g}{g} \mid g \\ \text{polynomial expression} & g ::= c \mid p \mid [A] \mid g + g \mid g \times g \mid g^n \end{array}$$

where A is an agent (with allowed ε features), $c \in \mathbb{Q}$ is a *constant*, $n \in \mathbb{N}$ is an *exponent*, and $p \in \mathbb{P}$ is a *parameter*. The set \mathbb{P} specifies a set of parameters. For each parameter $p \in \mathbb{P}$, a domain of admissible positive values is assigned, denoted by $\mathcal{D}(p) \in 2^{\mathbb{Q}^+}$.

We need to specify how such expressions are actually evaluated to quantitative value. Typically, the evaluation means substituting constant values and molecular counts in the expression and evaluating it to a rational number. However, since the functions are generally parametrised, the evaluation can produce just a simplified expression that still contains unknown parameters. By \mathbb{F}_p^\diamond , we denote a fragment of the rate expressions where the agents are not present (therefore, it holds that $\mathbb{F}_p^\diamond \subseteq \mathbb{F}_p$).

Rate evaluation $\mathbb{F}_p \times \mathbb{M} \rightarrow \mathbb{F}_p^\diamond$ of a rate expression $f \in \mathbb{F}_p$ on a multiset $M \in \mathbb{M}$, written $f(M)$, is a rate expression $f' \in \mathbb{F}_p^\diamond$ such that each pattern $[A]$ is replaced by an integer equal to $\sum_{J\langle A \rangle \in \Gamma\langle A \rangle} M(J\langle A \rangle)$, expressing the sum of all possible instantiations of the pattern A , in general treated as a pattern since ε feature is allowed.

A BCSL *rule* R is a triple $(T_l, T_r, f) \in \mathbb{T} \times \mathbb{T} \times \mathbb{F}_p$, usually written as $T_l \xrightarrow{f} T_r$. The rule describes a structural change of a multiset defined by the difference between left-hand and right-hand side patterns, additionally associated with the rate expression f . We also often refer to the sides of the rule as $\text{LHS}(R)$, resp. $\text{RHS}(R)$, or simply LHS and RHS when the context of the rule is clear. We use the symbol \mathbb{R} to denote the universe of all possible rules.

A BCSL *model* \mathcal{B} is a tuple $(\mathcal{R}, \sigma_s, \sigma_a, M_0)$ such that \mathcal{R} is a finite set of rewrite rules, $\sigma_s \in \Sigma_s$ is a structure signature, $\sigma_a \in \Sigma_a$ is an atomic signature, and $M_0 \in \mathbb{M}$ is an initial multiset. We denote by \mathbb{B}_p the set of all parametrised (because the parameters can be used in rate expressions) BCSL models.

Let $\mathcal{B} = (\mathcal{R}, \sigma_s, \sigma_a, M_0)$ be a parametrised BCSL model. The *rewriting* is given by labelled transition relation $\rightarrow \subseteq \mathbb{M} \times \mathbb{R} \times \mathbb{M}$, written as $M + M_l \xrightarrow{R, f(M+M_l)} M + M_r$ with $M, M_l, M_r \in \mathbb{M}$, $R : T_l \xrightarrow{f} T_r$ from \mathbb{R} , and $f(M + M_l) \in \mathbb{F}_p^\diamond$, satisfying the following inference rule:

$$\frac{\begin{array}{l} \exists J\langle T_l \rangle \in \Gamma\langle T_l \rangle. J\langle T_l \rangle = M_l \\ \exists J\langle T_r \rangle \in \Gamma\langle T_r \rangle. J\langle T_r \rangle = M_r \\ J\langle T_l \rangle \Delta J\langle T_r \rangle \end{array}}{M + M_l \xrightarrow{R, f(M+M_l)} M + M_r}$$

The rewriting of the multisets gives semantics to the model. Intuitively, for pattern T_l , the corresponding agents from the state are found and consequently replaced according to pattern T_r . Applying such an operation transitively, starting in the initial multiset, yields a labelled transition system $\text{LTS}(\mathcal{B}) = (S, T, L)$, where S is a set of states (a state is a multiset of agents), T is a set of transitions (a transition corresponds to the application of a rule), and L is labelling function assigning to each transition an identifier of applied rule and additionally the evaluated rate expression.

As an example of the parametrised model, we can assume the model from [Figure 15](#) with a modification in its `#!definitions` section. We

omit the definition of constant k_1 , thus leaving the value of constant k_1 undefined, effectively declaring it as a parameter. The respective LTS is available in Figure 17.

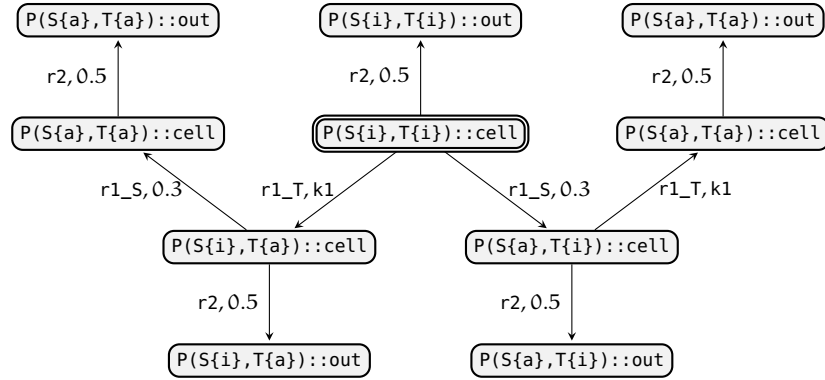


Figure 17: Labelled transition system of the model from Figure 15 with unknown parameter k_1 . While some of the labels contain particular quantitative values, others could not be fully evaluated since they are parametrised by the unknown parameter.

3.2.1 Non-parametrised fragment

There are cases when the rate expressions associated with rules are completely known and do not need to be parametrised. We study such a fragment of the BCSL model separately and denote it by \mathbb{B}_q . This fragment allows us to formally capture systems with fully known dynamics and define typical analysis methods for them (e.g. simulation is generally undefined for parametrised models).

We first define the grammar for the *rate expression* f without parameters. We denote by \mathbb{F} the set of all such rate expressions given by the following grammar:

$$\begin{array}{ll} \text{rate expression} & f ::= \frac{g}{g} \mid g \\ \text{polynomial expression} & g ::= c \mid [A] \mid g + g \mid g \times g \mid g^n \end{array}$$

where all symbols have the same meaning as in the previous case.

The evaluation of the rate needs to be modified to the non-parametrised case. *Rate evaluation* $\mathbb{F} \times \mathbb{M} \rightarrow \mathbb{Q}$ of a rate expression $f \in \mathbb{F}$ on a multiset $M \in \mathbb{M}$, written $f(M)$, is a rational number $n \in \mathbb{Q}$ such that each expression $[A]$ is replaced by an integer equal to $\sum_{J\langle A \rangle \in \Gamma\langle A \rangle} M(J\langle A \rangle)$ expressing the sum of occurrences of all possible instantiations of the agent A .

In the non-parametrised setting, the *rewrite rule* R is defined as $(T_l, T_r, f) \in \mathbb{T} \times \mathbb{T} \times \mathbb{F}$, usually written as $T_l \xrightarrow{f} T_r$.

As a consequence, while the model is defined in the same fashion, the rewriting of a model $\mathcal{B} = (\mathcal{R}, \sigma_s, \sigma_a, M_0)$ produces an LTS with

slightly modified label type. In particular, the *rewriting* of the multisets is given by labelled transition relation $M \xrightarrow{R,n} M'$ where $n \in \mathbb{Q}$. Instead of partially evaluated rate expression, the rates are evaluated to a rational number. An example of such LTS is available in Figure 16, corresponding to an example model from Figure 15.

3.2.2 Qualitative fragment

The parametrised and non-parametrised BCSL models assume at least partial quantitative information about the dynamics of the modelled system. However, this is not always the case. Therefore, we also define a *qualitative* fragment of BCSL, denoted as \mathbb{B} , that does not employ the rate expressions at all.

With no rate expressions used, the whole mechanisms for its evaluation can be omitted. The definition of a rule R simplifies to a pair $(T_l, T_r) \in \mathbb{T} \times \mathbb{T}$, simply written as $T_l \rightarrow T_r$.

```

#! rules
r1_S ~ P(S{i})::cell ⇒ P(S{a})::cell
r1_T ~ P(T{i})::cell ⇒ P(T{a})::cell
r2 ~ P()::cell ⇒ P()::out

#! inits
1 P(S{i},T{i})::cell

```

Figure 18: Example of a qualitative BCSL model. On the qualitative level, it corresponds to the model from Figure 15, but it has defined no rate expression.

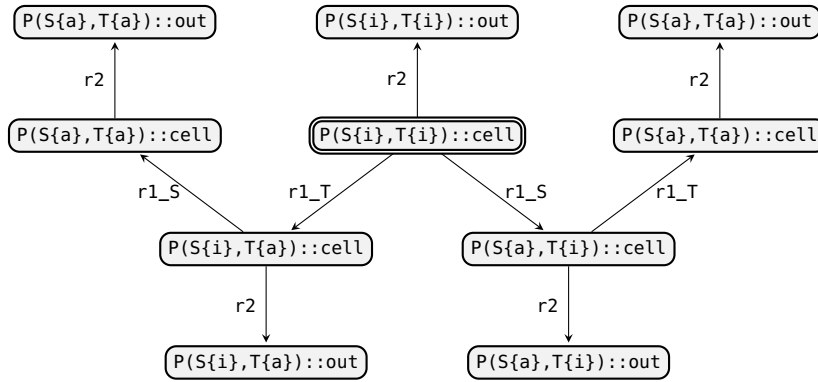


Figure 19: Transition system of the model from Figure 18. The labels of the edges contain only information about the applied rule.

This change influences the rewriting process of a model $\mathcal{B} \in \mathbb{B}$ in a way that the produced LTS does not contain rate information in the

labels of edges. More precisely, let $\mathcal{B} = (\mathcal{R}, \sigma_s, \sigma_a, M_0)$ be a qualitative BCSL model. The *rewriting* is given by labelled transition relation $\rightarrow \subseteq \mathbb{M} \times \mathbb{R} \times \mathbb{M}$, written as $M + M_l \xrightarrow{R} M + M_r$ with $M, M_l, M_r \in \mathbb{M}$ and $R : T_l \rightarrow T_r$ from \mathbb{R} , satisfying the following inference rule:

$$\frac{\begin{array}{l} \exists \mathcal{J}\langle T_l \rangle \in \Gamma\langle T_l \rangle. \mathcal{J}\langle T_l \rangle = M_l \\ \exists \mathcal{J}\langle T_r \rangle \in \Gamma\langle T_r \rangle. \mathcal{J}\langle T_r \rangle = M_r \\ \mathcal{J}\langle T_l \rangle \Delta \mathcal{J}\langle T_r \rangle \end{array}}{M + M_l \xrightarrow{R} M + M_r}$$

An example of a qualitative model is in [Figure 18](#), with its corresponding LTS in [Figure 19](#). While their layout, contents of the states, and used rules for transitions are the same as in the quantitative case, the transition labels do not contain any rate-related information.

3.2.3 Syntactic extensions

In this section, we define several syntactic extensions which increase the readability of the rule expressions. Note that each rule expression in an extended form can always be translated into a basic form. All rule expressions containing the following extensions must be converted to basic form before the semantics can be applied. For better demonstration, we provide a running example, which will go through all syntactic extensions, starting with Running example 1. Please note there is no biological sense of the example model, its only purpose is to demonstrate all defined syntactic extensions effectively.

Running example 1. (No syntactic extensions)

$$\begin{aligned} 1 & \sim \text{KaiC}(S\{u\}, T\{\varepsilon\}) . \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) . \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) :: \text{cyt} \Rightarrow \\ & \Rightarrow \text{KaiC}(S\{p\}, T\{\varepsilon\}) . \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) . K(S\{\varepsilon\}, T\{\varepsilon\}) :: \text{cyt} \\ 2 & \sim \text{KaiC}(S\{u\}, T\{\varepsilon\}) . \text{KaiB}(\emptyset) :: \text{cyt} \Rightarrow \text{KaiC}(S\{p\}, T\{\varepsilon\}) . \text{KaiB}(\emptyset) :: \text{cyt} \\ 3 & \sim \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) :: \text{cyt} + \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) :: \text{cyt} + \\ & \quad + \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) :: \text{cyt} \Rightarrow \\ & \Rightarrow \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) . \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) . \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) :: \text{cyt} \\ 4 & \sim \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) . \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) . \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) :: \text{cyt} \Rightarrow \\ & \Rightarrow \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) :: \text{cyt} + \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) :: \text{cyt} + \\ & \quad + \text{KaiC}(S\{\varepsilon\}, T\{\varepsilon\}) :: \text{cyt} \end{aligned}$$

$$\begin{aligned} \sigma_a &= \{ S \rightarrow \{u, p\}, T \rightarrow \{a, i\} \} \\ \sigma_s &= \{ \text{KaiC} \rightarrow \{S, T\}, \text{KaiB} \rightarrow \emptyset \} \end{aligned}$$

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

3.2.3.1 Composition context elimination

It is possible to omit all atomic expressions with unspecified state ε from compositions of structure agents (Running example 2). Such agent expressions do not give any additional information, and the whole composition can be reconstructed from the given signature.

Running example 2. (Unspecified states)

```

1 ~ KaiC(S{u}).KaiC(∅).KaiC(∅)::cyt ⇒
    ⇒ KaiC(S{p}).KaiC(∅).KaiC(∅)::cyt
2 ~ KaiC(S{u}).KaiB(∅)::cyt ⇒ KaiC(S{p}).KaiB(∅)::cyt
3 ~ KaiC(∅)::cyt + KaiC(∅)::cyt + KaiC(∅)::cyt ⇒
    ⇒ KaiC(∅).KaiC(∅).KaiC(∅)::cyt
4 ~ KaiC(∅).KaiC(∅).KaiC(∅)::cyt ⇒
    ⇒ KaiC(∅)::cyt + KaiC(∅)::cyt + KaiC(∅)::cyt

```

Additionally, this extension can go even further by omitting the \emptyset part from structure agents completely (Running example 3).

Running example 3. (Empty composition)

```

1 ~ KaiC(S{u}).KaiC().KaiC()::cyt ⇒ KaiC(S{p}).KaiC().KaiC()::cyt
2 ~ KaiC(S{u}).KaiB()::cyt ⇒ KaiC(S{p}).KaiB()::cyt
3 ~ KaiC()::cyt + KaiC()::cyt + KaiC()::cyt ⇒
    ⇒ KaiC().KaiC().KaiC()::cyt
4 ~ KaiC().KaiC().KaiC()::cyt ⇒
    ⇒ KaiC()::cyt + KaiC()::cyt + KaiC()::cyt

```

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

3.2.3.2 Complex signature

We extend the model definition by complex signature σ_X (Running example 4). In this signature, there are defined aliases for valid complex expressions. Then, the original complex expressions are substituted by the aliases.

Running example 4. (Complex alias)

Definition of complex signature:

$$\sigma_X = \left\{ \begin{array}{l} \text{KaiC3}::\text{cyt} \rightarrow \text{KaiC}().\text{KaiC}().\text{KaiC}()::\text{cyt}, \\ \text{KaiBC}::\text{cyt} \rightarrow \text{KaiC}().\text{KaiB}()::\text{cyt} \end{array} \right\}$$

```

1 ~ KaiC(S{u}).KaiC().KaiC()::cyt ⇒ KaiC(S{p}).KaiC().KaiC()::cyt
2 ~ KaiC(S{u}).KaiB()::cyt ⇒ KaiC(S{p}).KaiB()::cyt
3 ~ KaiC()::cyt + KaiC()::cyt + KaiC()::cyt ⇒ KaiC3::cyt
4 ~ KaiC3::cyt ⇒ KaiC()::cyt + KaiC()::cyt + KaiC()::cyt

```

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

3.2.3.3 Directions

We allow rule expressions to be bi-directional – it is just a shortcut for two rule expressions, and it can be converted to the basic rule expression form. A rule expression $\rho : \text{LHS} \Leftrightarrow \text{RHS}$ can be written as two rule expressions $\rho_1 : \text{LHS} \Rightarrow \text{RHS}$ and $\rho_2 : \text{RHS} \Rightarrow \text{LHS}$ (Running example 5).

Running example 5. (Bi-directional rules)

```
1 ~ KaiC(S{u}).KaiC().KaiC()::cyt ⇒ KaiC(S{p}).KaiC().KaiC()::cyt
2 ~ KaiC(S{u}).KaiB()::cyt ⇒ KaiC(S{p}).KaiB()::cyt
3 ~ KaiC()::cyt + KaiC()::cyt + KaiC()::cyt ⇔ KaiC3::cyt
```

Definition of rules 3 and 4 from Running example 4 was replaced by one bi-directional rule 3 in Running example 5.

3.2.3.4 Stoichiometry

For a rule expression of form:

$$\beta_1 :: c + \beta_2 :: c + \dots + \beta_n :: c \Rightarrow \beta_1.\beta_2. \dots .\beta_n :: c$$

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

For the left-hand side $\beta_1 :: c + \beta_2 :: c + \dots + \beta_n :: c$ of the reordered rule expression, we can replace all rule expressions $[\beta_i, \dots, \beta_j]$ which belong to the same non-crossing partition B by notation ' $k \beta$ ', where β is a representative from β_i, \dots, β_j (they are all equivalent) and k is the number of the expressions in partition B (Running example 6). Note that this process is fully reversible – we can simply enumerate all expressions for each partition.

Running example 6. (Repetitions)

```
1 ~ KaiC(S{u}).KaiC().KaiC()::cyt ⇒ KaiC(S{p}).KaiC().KaiC()::cyt
2 ~ KaiC(S{u}).KaiB()::cyt ⇒ KaiC(S{p}).KaiB()::cyt
3 ~ 3 KaiC()::cyt ⇔ KaiC3::cyt
```

The definition of rule expression 3 from Running example 5 was replaced by a new rule expression using stoichiometry.

3.2.3.5 Locations

The localisation operator is intended to allow an alternative way of expressing the hierarchically constructed agent expressions (Running example 7). The main idea is to allow zooming into individual parts of complex and structure expressions. For this purpose, we use $a : b$ notation such that a, b are arbitrary agents which satisfy one of the conditions:

1. $A : S \Leftrightarrow \exists A' \in d(S)$ such that $A \triangleleft A'$,
2. $A : X \Leftrightarrow$ there exists a S in X with a $A' \in d(S)$ such that $A \triangleleft A'$,
3. $S : X \Leftrightarrow$ there exists a S' in X such that $S \triangleleft S'$.

such that A, S, X are atomic, structure and complex, respectively, and \triangleleft is a compatibility relation defined in [Section 3.4.1](#), intuitively relating similar agents where one has less specified properties.

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 composition (resp. sequence) of the agent β such that it is *compatible with* the agent α . Then, agent α' is merged with agent α and agent β' is constructed.

Running example 7. (Structure zooming)

- 1 $\sim S\{u\}:KaiC():KaiC3::cyt \Rightarrow S\{p\}:KaiC():KaiC3::cyt$
- 2 $\sim S\{u\}:KaiC():KaiBC::cyt \Rightarrow S\{p\}:KaiC():KaiBC::cyt$
- 3 $\sim 3 KaiC():cyt \Leftrightarrow KaiC3::cyt$

Definition of rule expressions 1 and 2 from Running example 6 was replaced using locations. The localisation operator allowed us to leverage the complex signatures even further.

3.2.3.6 Variables

Rule expressions 1 and 2 from Running example 7 are very similar except for the context of the complex expression they take place in. We can substitute this context with a variable with a given domain.

In a rule expression, one agent expression might be referenced using a variable as a set of rule agent expressions it can be replaced with (Running example 8). Such an agent expression is referenced as question mark. Moreover, in the case when a question mark is used in a location, it must satisfy the defined conditions.

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

Running example 8. (Variables)

- 1 $\sim S\{u\}:KaiC():?::cyt \Rightarrow S\{p\}:KaiC():?::cyt ; ? = \{KaiC3, KaiBC\}$
- 2 $\sim 3 KaiC():cyt \Leftrightarrow KaiC3::cyt$

Definition of rule expressions 1 and 2 from Running example 7 was replaced as a single rule expression with a variable.

This is the final syntactic extension. Compared to the original model (Running example 1), the resulting model is more concise and readable.

3.3 MRS ENCODING

This section shows how an MRS can be constructed for any qualitative BCSL model $\mathcal{B} \in \mathbb{B}$ and that such an MRS exhibits equivalent behaviour to the original BCSL model. This proves that MRS is a suitable foundation for the low-level data structures and operations for BCSL implementation.

First, we show how an MRS $\mathcal{M} = (\xi, M_0)$ can be constructed from a BCSL model $\mathcal{B} = (\mathcal{R}, \sigma_a, \sigma_s, S_0)$. This approach is based on grounding agents and rules (supplement the missing context using grounding function Θ). In particular, we need to do two steps – construct the support set of elements by grounding all possible agents and then construct the set of multiset rewriting rules by grounding each BCSL rule, creating its possible instantiations in terms of multisets.

The abstraction provided by BCSL rules, which allow expressing patterns, needs to be grounded in concrete multisets. Informally, this is accomplished by supplementing the context information from the signature functions to the patterns, obtaining particular realisations of patterns.

We define grounding function Θ for a pattern T as a set of all its possible instantiated multisets $\Theta(T) = \{\mathcal{I}\langle T \rangle \mid \mathcal{I}\langle T \rangle \in \Gamma\langle T \rangle\}$. Applied to a rule R , we obtain a set of all possible *reactions* using consistent instantiations $\Theta(R) = \{\mathcal{I}\langle \text{LHS} \rangle \Rightarrow \mathcal{I}\langle \text{RHS} \rangle \mid \mathcal{I}\langle \text{LHS} \rangle \in \Gamma\langle \text{LHS} \rangle \wedge \mathcal{I}\langle \text{RHS} \rangle \in \Gamma\langle \text{RHS} \rangle \wedge \mathcal{I}\langle \text{LHS} \rangle \Delta \mathcal{I}\langle \text{RHS} \rangle\}$, where $\mathcal{I}\langle \text{LHS} \rangle$ and $\mathcal{I}\langle \text{RHS} \rangle$ are treated as multisets.

Next, we show how to create a set of all possible unique agents present in the model, which can be considered as the set of elements. It is constructed from initial state M_0 and a set of rules \mathcal{R} with the information provided in signature functions. We assume that the initial state M_0 contains agents which are already grounded.

Let \mathcal{S}_0 be a set of unique elements from initial state M_0 and $\mathcal{S}_{\mathcal{R}}$ be a set of all possible grounded agents present in the rules \mathcal{R} defined as $\mathcal{S}_{\mathcal{R}} = \{A \in \Theta(A') \mid A' \in \mathcal{A}(R) \wedge R \in \mathcal{R}\}$, where $\mathcal{A}(R) = \text{LHS}(R) \cup \text{RHS}(R)$ is a set of all agents used in rule R . Then, the set of all possible unique agents present in the model is $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_{\mathcal{R}}$.

Next, we show how to construct a set of MRS rewriting rules from a BCSL rule. The approach is straightforward since the grounding function Θ creates the set of all possible grounded rules (reactions). Then, we need to create a pair of multisets from both sides of each grounded rule. The obtained pair of multisets can be directly considered as a multiset rewriting rule over support set \mathcal{S} because all the possible agents are already present in the set \mathcal{S} (follows from its construction). We call such rule *MRS instantiation* of the BCSL rule.

Let R be a BCSL rule. We define *MRS instantiation* $\mu(R)$ of rule R as a multiset rewriting rule $\mu(R) = (\mathfrak{l}, r)$ where $\mathfrak{l} \Rightarrow r \in \Theta(R)$. Then, for

a set of BCSL rules \mathcal{R} , the corresponding set of MRS rules ξ is defined as a set of all possible MRS instantiations $\xi = \{\mu(R) \mid R \in \mathcal{R}\}$.

We obtain the MRS $\mathcal{M} = (\xi, M_0)$ over the set of elements \mathcal{S} by taking constructed set of multiset rewriting rules ξ (MRS instantiations) and the initial state M_0 . In [Figure 20](#), there is an example of the MRS constructed from the BCSL model available in [Figure 18](#) using this approach.

$$\mathcal{S} = \left\{ \begin{array}{l} P(S\{i\}, T\{i\}) : \text{cell}, P(S\{a\}, T\{i\}) : \text{cell}, P(S\{i\}, T\{a\}) : \text{cell}, \\ P(S\{a\}, T\{a\}) : \text{cell}, P(S\{i\}, T\{i\}) : \text{out}, P(S\{a\}, T\{i\}) : \text{out}, \\ P(S\{i\}, T\{a\}) : \text{out}, P(S\{a\}, T\{a\}) : \text{out} \end{array} \right\}$$

(a) Set of all unique objects.

$$\mathcal{M} = \left(\begin{array}{l} M_0 = \{ P(S\{i\}, T\{i\}) : \text{cell} \}, \\ \xi = \left\{ \begin{array}{l} \mu_{r1_S} : \{P(S\{i\}, T\{i\}) : \text{cell}\} \rightarrow \{P(S\{a\}, T\{i\}) : \text{cell}\}, \\ \mu_{r1_S} : \{P(S\{i\}, T\{a\}) : \text{cell}\} \rightarrow \{P(S\{a\}, T\{a\}) : \text{cell}\}, \\ \mu_{r1_T} : \{P(S\{i\}, T\{i\}) : \text{cell}\} \rightarrow \{P(S\{i\}, T\{a\}) : \text{cell}\}, \\ \mu_{r1_T} : \{P(S\{a\}, T\{i\}) : \text{cell}\} \rightarrow \{P(S\{a\}, T\{a\}) : \text{cell}\}, \\ \mu_{r2} : \{P(S\{i\}, T\{i\}) : \text{cell}\} \rightarrow \{P(S\{i\}, T\{i\}) : \text{out}\}, \\ \mu_{r2} : \{P(S\{a\}, T\{i\}) : \text{cell}\} \rightarrow \{P(S\{a\}, T\{i\}) : \text{out}\}, \\ \mu_{r2} : \{P(S\{i\}, T\{a\}) : \text{cell}\} \rightarrow \{P(S\{i\}, T\{a\}) : \text{out}\}, \\ \mu_{r2} : \{P(S\{a\}, T\{a\}) : \text{cell}\} \rightarrow \{P(S\{a\}, T\{a\}) : \text{out}\} \end{array} \right\} \end{array} \right)$$

(b) Instantiated rules.

Figure 20: MRS representation of a BCSL model from [Figure 18](#). All the objects in set \mathcal{S} are unique strings representing possible forms of original BCSL agents. These are used in multiset rewriting rules and the initial multiset. For convenience, to allow identification of source rule, we label each constructed multiset rewriting rule by μ_R where R is the label of source BCSL rule.

Next, we show that the behaviour of such a constructed MRS is equivalent to the behaviour of the original BCSL model. This is shown in [Theorem 1](#) by considering that the type of states in both systems is the same, both the BCSL rule and its MRS instantiation can always be applied to the same state ([Lemma 1](#)), and they can always be rewritten to the same states ([Lemma 2](#)).

The semantics of MRS are given in terms of a set of infinite runs, while the semantics of BCSL are given in terms of LTS. First, we need to relate these two constructs. We define how a set of runs corresponds to an LTS. To ensure that the LTS represents only infinite runs, we extend it to LTS_ε such that we add self-loops on states with no successors labelled by an empty rule ε .

Let $LTS_\varepsilon = (S, T, L)$ be a labelled transition system. LTS_ε generates a set of infinite runs $\mathcal{L}(LTS_\varepsilon)$ such that the infinite run π of form

$\pi = s_0, s_1, \dots, s_n, \dots$ belongs to $\mathfrak{L}(\text{LTS}_\varepsilon)$ if (i) $s_0 \in S$ and (ii) for all $i \geq 1 : (s_{i-1}, s_i) \in T$. Moreover, the run π has a run label $\vec{\pi} = l_1, l_2, \dots, l_n, \dots$ such that for all $i \geq 1 : L((s_{i-1}, s_i)) = l_i$.

Please note that the multisets in constructed MRS use as elements grounded BCSL agents, and therefore the type of MRS multisets and BCSL multisets can be considered to be the same and they can be freely interchanged and checked for equality.

From the construction of the set of rules ξ follows that for any rule $\text{lhs} \Rightarrow \text{rhs} \in R$, the function Θ creates grounded rules, which represent all possible instantiations. Then, for any instantiation $L \Rightarrow R$, L and R are used to form a multiset rewriting rule, obtaining an MRS instantiation.

Lemma 1. *Let M be a grounded multiset, R a BCSL rule, and $\mu = (\bullet\mu, \mu\bullet)$ its MRS instantiation. Then, R can be applied to M iff μ can be applied to M .*

Proof. From construction of \mathcal{M} we know that to BCSL rules correspond their MRS instantiations.

\Rightarrow : if R

- a) *can be applied to M , then there exists $L \in \Theta(\text{LHS})$ such that $L \subseteq M$ (follows from the definition of BCSL rewriting). That means there has to exist an MRS instantiation $\mu = (\bullet\mu, \mu\bullet)$ in ξ of rule R such that it can be applied to M because $\bullet\mu \equiv L$ and therefore $\bullet\mu \subseteq M$ and μ is enabled.*
- b) *can not be applied to M , then for all $L \in \Theta(\text{LHS})$ holds that $L \not\subseteq M$ (follows from the definition of BCSL rewriting). That means that any MRS instantiation $\mu = (\bullet\mu, \mu\bullet)$ in ξ of rule R can not be applied to M because $\bullet\mu \equiv L$ and therefore $\bullet\mu \not\subseteq M$ and μ is not enabled.*

\Leftarrow : Symmetrically, if $\mu = (\bullet\mu, \mu\bullet)$

- a) *can be applied to M , then μ is enabled and therefore $\bullet\mu \subseteq M$. That means there has to exist a rule R such that μ is its MRS instantiation with $L \in \Theta(\text{LHS})$ where $\bullet\mu \equiv L$. Therefore, also $L \subseteq M$ and R can be applied to M .*
- b) *can not be applied to M , then μ is not enabled and therefore $\bullet\mu \not\subseteq M$. That means that any rule R such that μ is its MRS instantiation with $L \in \Theta(\text{LHS})$ where $\bullet\mu \equiv L$, holds that $L \not\subseteq M$ and R can not be applied to M .*

□

Lemma 2. *Let M be a grounded multiset, R a BCSL rule of form $\text{lhs} \Rightarrow \text{rhs}$, and $\mu = (\bullet\mu, \mu\bullet)$ its MRS instantiation. Then, by applying R to M , we get a set of possible multisets. Among them, there is a multiset M' , which can be obtained by applying μ to M .*

Proof. Follows from the definition of BCSL rewriting where instantiations of both LHS and RHS are created, which corresponds to the MRS instantiation μ . Then, instantiated agents from LHS are subtracted from, and RHS agents are added to the current state, which is in parallel with the MRS approach. Finally, we know that states in BCSL directly correspond to multisets in MRS, which forms the same basis for both formalisms. \square

Having such constructed MRS \mathcal{M} , we need to show that its behaviour (set of runs) corresponds to the behaviour (transition system) of the BCSL model.

Theorem 1. *For any BCSL model $\mathcal{B} = (\mathcal{R}, \sigma_a, \sigma_s, M_0)$ there exists an MRS $\mathcal{M} = (\xi, M_0)$ with $\mathfrak{L}(\text{LTS}_\varepsilon(\mathcal{B})) = \mathfrak{L}(\mathcal{M})$.*

When we construct the MRS \mathcal{M} using the approach described in the previous section, the proof of the theorem boils down to proving that for any grounded multiset M , the following two implications hold:

- \Rightarrow : for any BCSL rule $R \in \mathcal{R}$, if \mathcal{B} can apply R to M then there exists MRS rule $\mu \in \xi$ such that \mathcal{M} can apply μ to M
- \Leftarrow : for any MRS rule $\mu \in \xi$, if \mathcal{M} can apply μ to M then there exists $R \in \mathcal{R}$ such that \mathcal{M} can apply R to M

and in both cases we obtain the same multiset M' .

Proof. Follows from the construction of \mathcal{M} (construction of the corresponding MRS instantiations of rules), [Lemma 1](#) (either both rules are enabled or neither of them is), and [Lemma 2](#) (both rules create identical results). \square

3.4 BCSL CONSTRUCTION

In the previous section, it was proven that MRS can be used for low-level qualitative implementation of BCSL. In this section, we investigate details of constructive embedding of BCSL rules to the domain of MRS. We focus on the process of *matching* as a key step in the application of a rule. It can be seen as a detailed insight into how instantiations can be handled. While in the previous section, we showed it could be done indirectly via encoding to MRS, in this section, we show that by employing suitable data structures, this process can be done with a more direct approach. We show how MRS can be used as a foundation for the low-level data structures and operations for BCSL implementation, and for comprehensiveness, we recall the main BCSL objects and put them in context with detailed data structures.

3.4.1 Objects construction

In the whole section, let $\mathcal{V}_a, \mathcal{V}_s, \mathcal{V}_\delta, \mathcal{V}_c$ 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 syntax examples for the most important objects with their definitions. The formal relation of syntactic form to the objects is given below (Section 3.4.2).

Let us recall the signatures that define a set of allowed states for an atomic name and an allowed set of atomic names for a structure name. *Atomic signature* is a function $\sigma_a : \mathcal{V}_a \rightarrow 2^{\mathcal{V}_\delta}$ that associates each atomic name to a set of state names. Similarly, *structure signature* is a function $\sigma_s : \mathcal{V}_s \rightarrow 2^{\mathcal{V}_a}$ that associates each structure name to a set of atomic names.

For example, $\{ S \rightarrow \{u, p\}, Q \rightarrow \{a, i\} \}$ is a set of atomic signatures and $\{ \text{KaiC} \rightarrow \{S, Q\}, \text{KaiB} \rightarrow \emptyset \}$ is a set of structure signatures.

3.4.1.1 Atomic agent

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

Atomic agents are the simplest objects used for describing biological entities. Each atomic agent has its name and state. The allowed set of admissible states for the atomic agent (with additional empty ε state) is given by signature $\sigma_a(\eta)$. We use the symbol \mathbb{A} to denote the universe of all possible atomic agents.

Let A, A' be atomic agents. A is *equal* to A' , written $A = A'$, iff $\eta(A) = \eta(A') \wedge \delta(A) = \delta(A')$. Intuitively, the defined equality relation of atomic agents is an equivalence.

Atomic agents are usually used to express small biological entities which can change their state, for example, amino acids or small inorganic molecules. Examples of atomic agents are $A_1 = (S, u)$, written as $S\{u\}$, and $A_2 = (Q, \varepsilon)$, written as $S\{\varepsilon\}$. Note the meaning of ε is the state is unknown or not important to be considered in a given context.

The compatibility of atomic agents is a key property defined between agents. The agent A is *compatible with* agent A' , written $A \triangleleft A'$, if either $A = A'$ or $\eta(A) = \eta(A') \wedge \delta(A) = \varepsilon$. An agent is compatible with another agent if they have the same name and are in the same state or if the first agent is in the unknown ε state. It provides a formal way to determine which agent is more detailed, i.e. its state is more specified.

Finally, let $A \in \mathbb{A}$ be an atomic agent. We say the agent A is *fully specified*, written $\triangle A$, iff $\forall A' \in \mathbb{A}$ such that $A' \neq A : \neg(A' \triangleleft A)$.

3.4.1.2 Structure agent

Structure agent S is a pair (η, γ) where $\eta \in \mathcal{V}_s$ is a name and $\gamma \subseteq \mathbb{A}$ is a set of atomic agents called composition such that $\forall A, A' \in \gamma : \eta(A) \neq \eta(A')$. The name and the composition of the agent S are usually denoted by $\eta(S)$ and $\gamma(S)$, respectively.

A structure agent represents a biochemical object that is composed of a set of known atomic agents, while we know that composition is abstract and not necessarily complete. This set is restricted according to the given structure signature with the same name as the structure agent. We use the symbol \mathbb{S} to denote the universe of all possible structure agents.

Let S, S' be structure agents. S is *equal* to S' , written $S = S'$, iff $\eta(S) = \eta(S') \wedge \gamma(S) = \gamma(S')$. Intuitively, the defined equality on structure agents is an equivalence relation.

The key construct of a structure agent is *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.

A typical example of a structure agent is a protein, where the atomic agents are amino acids that are of interest in the particular setting. Imagine that in our modelled system, only three out of a few hundred amino acids are able to undergo some post-translational modifications, such as phosphorylation or methylation. It is suitable to model only these three amino acids instead of the entire primary structure of the protein. As examples of structure agent, we provide agent $S_1 = (K, \{(S, p), (Q, i)\})$, written as $K(S\{p\}, Q\{i\})$, and agent $S_2 = (K, \{(Q, a)\})$, written as $K(Q\{a\})$.

Similarly to atomic agents, we define the compatibility of structure agents. The agent S is *compatible with* agent S' , written $S \triangleleft S'$, iff either $S = S'$ or $\eta(S) = \eta(S') \wedge \forall A \in \gamma(S) \exists A' \in \gamma(S') : A \triangleleft A'$.

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

Additionally, we define the difference at the level of compositions of structure agents, which is necessary for the construction of semantics below. Let γ, γ' be two arbitrary compositions. We define *difference of compositions* $\gamma \ominus \gamma' = \{A \mid A \in \gamma \wedge A \notin \gamma \cap \gamma'\}$ where $\gamma \cap \gamma' = \{A \mid A \in \gamma \wedge \exists A' \in \gamma' : \eta(A') = \eta(A)\}$.

Finally, we say the agent S is *fully specified*, written $\triangle S$, iff $\forall S' \in \mathbb{S}$ such that $S' \neq S : \neg(S' \triangleleft S)$.

3.4.1.3 Complex agent

A complex agent represents a non-trivial composite biochemical object that is inductively constructed from already-known biological objects. In rule-based languages, this is usually defined by introducing bonds between individual biochemical objects [DK07]. In BCSL, we abstract from the detailed specification of bonds, and we rather assume a complex as a coexistence of certain objects in a particular group. Moreover, a complex agent resides in a compartment which gives it a spatial position.

Formally, a *complex agent* X is a pair (λ, com) where $\lambda \in (\mathbb{A} \cup \mathbb{S})^n$ is a sequence of agents, $\text{com} \in \mathcal{V}_c$ is a compartment for some $n \in \mathbb{N}$. The sequence and the compartment of the agent X is usually denoted by $\lambda(X)$ and $\text{com}(X)$, respectively. We use the symbol \mathbb{X} to denote the universe of all possible complex agents.

The key element of a complex agent is *sequence* inductively constructed from existing agents. In contrast to 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). Example of a complex agent is $X = (((K, \{(S, p), (Q, i)\}), (S, p)), \text{cell})$, written as $K(S\{p\}, Q\{i\}) . S\{p\} :: \text{cell}$.

The order in the sequence is necessary to uniquely identify agents which are equal. On the other hand, when comparing two sequences, we do it regardless of the order.

Let X, X' be two complex agents. X is *equal* to X' , written $X = X'$, iff $\text{com}(X) = \text{com}(X')$ and $\mathbb{M}(\lambda(X)) = \mathbb{M}(\lambda(X'))$ (multiset-based comparison of sequences). Intuitively, the defined equality on complex agents is an equivalence relation.

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

The compatibility of complex agents benefits from already defined properties of atomic and structure agents. The complex agent X is *compatible with* complex agent X' , written $X \triangleleft X'$, iff either $X = X'$ or $\text{com}(X) = \text{com}(X') \wedge \exists \lambda' \in \text{PERM}(\lambda(X'))$ such that $\forall i \in [1, n] : \lambda_i(X) \triangleleft \lambda'_i$, where n is length of sequence which is the same for both sequences and PERM set of all possible permutations of length n .

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

Finally, we say the agent X is *fully specified*, written $\triangle X$, iff $\forall X' \in \mathbb{X}$ such that $X' \neq X : \neg(X' \triangleleft X)$.

It is worth noting that the complexes have no binding topology. While it provides many advantages, specifically when it comes to combinatorial explosion, it also has several drawbacks. The most impor-

tant one is that we are not able to express structural modifications on the level of complexes. These have to be encoded using states.

3.4.1.4 Rules construction

Let us have a simple example of a rule:

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

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

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

- $\chi \in \mathbb{X}^n$ is a sequence of complex agents,
- $\omega \in (\mathbb{A} \cup \mathbb{S})^m$ is a sequence of atomic and structure agents,
- $\iota \in \{0, \dots, n\}$ is an index determining the end of LHS of χ ,
- $\varphi \in \{1, \dots, m\}^k$ is an index map from ω to χ ,
- $\psi \in ((\{1, \dots, l, -\} \times \{l+1, \dots, m, -\})^{|\varphi|})$ is an index map from LHS to RHS

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

The reason for this particular construction is that it is necessary to capture the relationship between the left-hand side and the right-hand side of the rule. This is done by enumerating all atomic and structure agents ω from the sequence of complex agents χ . The index map ψ between the agents in ω determines pairs of agents from the left-hand side and the right-hand side, which correspond to each other. It is possible that there are agents which do not have a pair (denoted by $-$) in the situation when the rule is modelling *inflow* from (resp. *outflow* to) the system. Another index map φ serves for relating agents from ω back to the original sequence of complexes χ . Finally, by the index ι , we determine the end of the left-hand side of the rule. Note the index is zero in the situation when there are no agents on the left-hand side. We use the symbol \mathbb{R} to denote the universe of all possible rules. An example of a rule is available in [Figure 21](#).

Not every rule is meaningful. For example, a rule where not a single agent is changed or a rule where the relation between the left-hand and the right-hand side would not be clear. In order to avoid such

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

$$\begin{aligned}
&K(S\{u\}) \cdot B(\emptyset)::\text{cyt} + C(\emptyset) \cdot D\{i\}::\text{cyt} + A\{\varepsilon\}::\text{cyt} \Rightarrow \\
&\Rightarrow K(S\{p\}) \cdot B(\emptyset) \cdot C(\emptyset)::\text{cyt} + D\{a\} \cdot A\{\varepsilon\}::\text{cyt} + H\{u\}::\text{cyt}
\end{aligned}
\tag{b}$$

Figure 21: (a) Example of the representation of a rule $R = (\chi, \omega, \iota, \varphi, \psi)$ and (b) its written form.

cases, we need to specify when a rule is *well-formed*, i.e. it makes sense semantically.

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

1. at least one of conditions is satisfied:
 - a) $\exists (i, j) \in \psi : \omega_i \neq \omega_j$,
 - b) $|\text{LHS}(R)| \neq |\text{RHS}(R)|$,
 - c) $\exists i \in [1, \iota] : \text{com}(\chi_i) \neq \text{com}(\chi_{\iota+i})$;
2. $\forall (i, j) \in \psi : \eta(\omega_i) = \eta(\omega_j)$;
3. $\forall (-, i) \in \psi : \Delta\omega_i$.

The conditions claim that an agent has to change during the rule application. This is ensured by condition (1), where there are three options: (1a) at least one pair of agents from LHS and RHS of the rule is different; (1b) the lengths of the LHS and RHS are different, i.e. either a new agent is created or complex is formed/dissociated; (1c) a compartment is changed. Any combination of these sub-conditions is allowed. The second condition (2) guarantees that the pairs of structure and atomic agents in ω of the rule have the same name. Please note the conditions (1) and (2) do not apply to those agents in ω , which do not have a pair on the other side of the rule. Finally, the condition (3) claims that if there is an agent which does not have defined a pair via index map ψ (denoted by $-$), it is required to be a fully

specified agent (but only in case of agent creation, it is not necessary for agent degradation).

3.4.2 Translation function

Once we defined BCSL agents, rules, and syntax for the language, we need to connect them in order to give semantic meaning to a model written in the syntax. For this purpose, we define the translation function F . It is defined recursively according to the expression given as an argument in double square brackets $\llbracket \dots \rrbracket$ as follows:

- $F\llbracket \eta\{\varepsilon\} \rrbracket = (\eta, \varepsilon) \in \mathbb{A}$
- $F\llbracket \eta\{\delta\} \rrbracket = (\eta, \delta) \in \mathbb{A}$
- $F\llbracket \eta(\emptyset) \rrbracket = (\eta, \emptyset) \in \mathbb{S}$
- $F\llbracket \eta(a_1, \dots, a_k) \rrbracket = (\eta, \{F\llbracket a_1 \rrbracket, \dots, F\llbracket a_k \rrbracket\}) \in \mathbb{S}$
- $F\llbracket \alpha_1 \dots \alpha_k :: c \rrbracket = ((F\llbracket \alpha_1 \rrbracket, \dots, F\llbracket \alpha_k \rrbracket), c) \in \mathbb{X}$
- $F\llbracket \Gamma_1 + \dots + \Gamma_n \Rightarrow \Gamma_{n+1} + \dots + \Gamma_m \rrbracket = (\chi, \omega, \iota, \varphi, \psi) \in \mathbb{R}$ with:
 - $\chi = (F\llbracket \Gamma_1 \rrbracket, \dots, F\llbracket \Gamma_n \rrbracket, F\llbracket \Gamma_{n+1} \rrbracket, \dots, F\llbracket \Gamma_m \rrbracket),$
 - $\omega = \text{++}_{i=1}^{|\chi|} \lambda(\chi_i),$
 - $\iota = n,$
 - $\varphi = (J_1, \dots, J_m)$ where $J_k = \sum_{i=1}^k |\lambda(\chi_i)|,$
$$\{(i, j) \mid i \in [1, \varphi_\iota] \wedge j \in [\varphi_\iota + 1, |\omega|] \wedge |i - j| = \varphi_\iota\} \cup$$
- $\psi = \{(i, -) \mid i \in [k, \varphi_\iota] \wedge k = |\omega| - \varphi_\iota + 1\} \cup$

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

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

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

3.4.3 Matching and replacement

Let us proceed with the mechanism of matching and replacement. Intuitively, matching chooses agents that are about to be modified

by the rule, while the replacement actually applies the rule to the selected agents.

Let $R = (\chi, \omega, \iota, \varphi, \psi)$, $R' = (\chi', \omega', \iota', \varphi', \psi')$ be two arbitrary rules, and $\mathcal{S} \in \mathbb{M}^{\mathbb{X}}$ be a state such that $\forall X \in \mathcal{S} : \Delta X$. Indeed, all complexes in the state must be fully specified. This corresponds with the fact that state contains concrete biological molecules. Then, $\models \subseteq \mathbb{R} \times \mathbb{M}^{\mathbb{X}} \times \mathbb{R}$ is the *matching* relation such that a tuple $(R, \mathcal{S}, R') \in \models$, written $R \models_R' \mathcal{S}$, iff

1. $\iota = \iota' \wedge \varphi = \varphi' \wedge \psi = \psi'$,
2. $|\chi| = |\chi'| \wedge |\omega| = |\omega'|$,
3. $\forall i \in [1, |\chi|] : \chi'_i \triangleleft \chi_i$,
4. $\mathbb{M}(\text{LHS}(R')) = \mathcal{S}$,
5. $\forall (i, j) \in \psi :$
 - a) $\omega_i \in \mathbb{A} \Rightarrow \begin{cases} \omega'_i = \omega'_j & \text{if } \omega_i = \omega_j \\ \omega_i = \omega'_i \wedge \omega_j = \omega'_j & \text{if } \omega_i \neq \omega_j \end{cases}$
 - b) $\omega_i \in \mathbb{S} \Rightarrow \gamma(\omega'_i) \ominus \gamma(\omega_i) = \gamma(\omega'_j) \ominus \gamma(\omega_j)$.

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

Finally, we define $\rightarrow \subseteq \mathbb{M}^{\mathbb{X}} \times \mathbb{R} \times \mathbb{M}^{\mathbb{X}}$ as the *replacement* relation such that a tuple $(\mathcal{S}, R, \mathcal{S}') \in \rightarrow$, written $\mathcal{S} \rightarrow_R \mathcal{S}'$, iff $\exists R' \in \mathbb{R} \exists x \subseteq \mathcal{S}$ such that $R \models_{R'} x \wedge \mathcal{S}' \setminus (\mathcal{S} \setminus x) = \mathbb{M}(\text{RHS}(R'))$. Replacement relation defines how a state is transformed according to a given rule.

With match and replace relations in place, we have shown how a state is transformed according to a given rule on the constructive level. From this point, it is quite straightforward to construct the whole BCSL model and implement its semantics, i.e. an LTS.

3.5 SUMMARY

This chapter provided a description of BCSL from several perspectives. It starts with an informal explanation of fundamental features to introduce the language to the reader to gain confidence and intuition behind the employed abstraction.

A formal definition follows that establishes the language on a mathematical basis. Three classes of BCSL models are established, namely *parametrised* models \mathbb{B}_p employing parameters in quantitative rate laws, *quantitative* models \mathbb{B}_q that do now allow parameters in the rates, and *qualitative* models \mathbb{B} with no rates at all. The individual

classes allow performing different analysis techniques, as we will see in the following chapter.

In the subsequent two sections, we focus on qualitative aspects of the language, relate the language to MRS, and explain in detail the key and most abstract language operations. In particular, we show that for any BCSL model, an equivalent MRS can be constructed. Therefore, we utilise this fact and use MRS as a low-level basis for the construction of BCSL. We provide a detailed description of the key operations, such as matching and replacement, on the level of multisets.

The quantitative variants of BCSL models are not discussed any further in this chapter since the extension compared to the qualitative variant is rather straightforward. In general, a rule can be additionally associated with an optional rate function just like defined in [Section 3.2](#), i.e. rational function over molecular counts (occurrences of agents in a state). The rate does not influence the explained rewriting mechanisms of the rule directly, it just adds some extra information about its kinetics.

Compared to the rather complicated structure of the rule due to matching requirements, the evaluation of the rate function is relatively straightforward. The critical step is to identify concrete agents corresponding to those specified in the rate function and then sum their occurrences in the current state. Then, known constants are assigned to the expression, and the function is evaluated to a rational number or, in the parametrised case, simplified to a function dependent on the unknown parameters. More details on this topic are provided in the implementation ([Section 6.1.4](#)).

*I do not know.
(summarising his life's work)*

JOSEPH-LOUIS LAGRANGE

DESCRIBING biochemical processes often requires incorporating knowledge on an abstract level to simplify the system description or substitute the missing details. For this purpose, in this chapter, we present *regulation* mechanisms, an extension of rule-based formalism with additional controls on the rewriting process.

Since MRS often serve as a basis for rule-based formalisms, and in [Section 3.3](#), it was shown that it is the case also for BCSL, we define the regulations on the level of MRS. By utilising the relationship between BCSL and MRS, the regulations can be easily used in the context of BCSL.

To summarise, we first define regulations on the level of MRS. In particular, we introduce several distinct regulation classes exhibiting different properties and mechanisms. Then we show a simple example of how the regulations can be used in BCSL. Finally, in [Section 7.4](#), we demonstrate the usage of such regulations on several case studies from the biochemical domain.

4.1 CLASSES OF REGULATED SYSTEMS

We start by introducing *regulated MRS* (rMRS), which restricts the applicability of rules, leading to a reduced set of possible runs. An rMRS is a triple $\overline{\mathcal{M}} = (M_0, \zeta, \xi)$ such that $\mathcal{M} = (M_0, \xi)$ is an MRS and ζ is a *regulation*, which defines further restrictions on runs and is used to reduce the set of possible runs $\mathfrak{L}(\mathcal{M})$ to $\mathfrak{L}(\overline{\mathcal{M}})$. The exact definition of ζ depends on the particular regulation mechanism and is specified for individual classes below.

The effect of regulation is that it can eliminate a run completely or snip the suffix of a run. Formally, the regulation is evaluated on the general semantics $\mathfrak{L}_{\text{ALL}}(\mathcal{M})$ of MRS \mathcal{M} (to assume all prefixes of runs), reducing the set of runs only to those satisfying the regulation ζ , and selecting only maximal runs (denoted by MAX in the sections below). To clarify, the infinite run π is maximal if (1) it does *not* contain an infinite suffix S composed of rules ε or (2) it has form $\overrightarrow{\pi} = FS$ where F

is a finite prefix and there is no other run $\pi' \in \mathcal{L}_{\text{ALL}}(\mathcal{M})$ with $\vec{\pi}' = F\mu S$ for some $\mu \in \xi \setminus \{\varepsilon\}$ such that π' satisfies regulation ζ .

4.1.1 Regular rewriting

Informally, the idea is to define an ω -regular [Th90] language ζ over rules, that is, a regular language over infinite words. Then, only runs with the run label from this language are allowed. The set of words ζ , due to its infiniteness, is almost always (though not necessarily) described via some more convenient mechanism (such as ω -regular expression).

In a *regular* multiset rewriting system, the regulation ζ is defined as an ω -regular language over set of rules ξ . The set of possible runs $\mathcal{L}(\mathcal{M})$ is reduced to a set of runs $\mathcal{L}(\overline{\mathcal{M}}) = \max\{\pi \in \mathcal{L}_{\text{ALL}}(\mathcal{M}) \mid \vec{\pi} \in \zeta\}$.

We denote by $\mathbb{R}\mathbb{R}$ the class of regular multiset rewriting systems. An example of an $\mathbb{R}\mathbb{R}$ system is available in Figure 22 with an example of valid and invalid runs in Figure 23.

$$\overline{\mathcal{M}} = \left(\begin{array}{l} M_0 = \emptyset, \quad \zeta = (\mu_1 \cdot \mu_2)^* \cdot \mu_3^* \cdot \varepsilon^\omega, \\ \xi = \left\{ \begin{array}{l} \mu_1 : \emptyset \rightarrow \{A\}, \mu_2 : \{A\} \rightarrow \{B\}, \mu_3 : \{B\} \rightarrow \emptyset \end{array} \right\} \end{array} \right)$$

Figure 22: Example of a regular multiset rewriting system over a set of elements $\mathcal{S} = \{A, B\}$. The regulation ζ allows a particular finite sequence of rules followed by infinite application of rule ε .

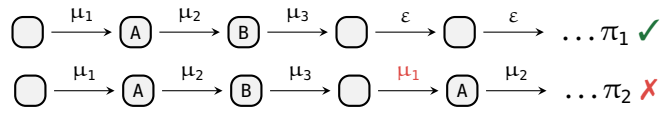


Figure 23: Example of valid and invalid runs of the $\mathbb{R}\mathbb{R}$ system from Figure 22. Run $\pi_1 \in \mathcal{L}(\overline{\mathcal{M}})$ because $\vec{\pi}_1 = \mu_1 \cdot \mu_2 \cdot \mu_3 \cdot \varepsilon^\omega \in \zeta$ and run $\pi_2 \notin \mathcal{L}(\overline{\mathcal{M}})$ because $\vec{\pi}_2 = \mu_1 \cdot \mu_2 \cdot \mu_3 \cdot \mu_1 \dots \notin \zeta$.

4.1.2 Ordered rewriting

The ordered rewriting regulation is based on a strict partial order of rules. Only runs with the run label, which does not break the ordering in the immediate successors, are allowed.

In an *ordered* multiset rewriting system, the regulation ζ is a strict partial order over a set of rules. For any two rules $\mu, \mu' \in \xi$ we write $\mu < \mu'$ iff $(\mu, \mu') \in \zeta$. The set of possible runs $\mathcal{L}(\mathcal{M})$ is reduced to a set of runs $\mathcal{L}(\overline{\mathcal{M}}) = \max\{\pi \in \mathcal{L}_{\text{ALL}}(\mathcal{M}) \mid \forall i > 0. \vec{\pi}[i+1] \not\prec \vec{\pi}[i]\}$.

We denote by \mathbb{OR} the class of ordered multiset rewriting systems. An example of an \mathbb{OR} system is available in Figure 24 with an example of valid and invalid runs in Figure 25.

$$\overline{\mathcal{M}} = \left(\begin{array}{l} M_0 = \emptyset, \quad \zeta = \left\{ (\mu_1, \mu_2) \right\}, \\ \xi = \left\{ \mu_1 : \emptyset \rightarrow \{A\}, \mu_2 : \{A\} \rightarrow \emptyset \right\} \end{array} \right)$$

Figure 24: Example of an ordered multiset rewriting system over a set of elements $\mathcal{S} = \{A\}$. The regulation ζ defines order $\mu_1 < \mu_2$ on rules μ_1, μ_2 , which does not allow μ_1 to be immediately used after μ_2 .

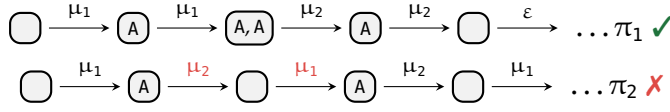


Figure 25: Example of valid and invalid runs of the \mathbb{OR} system from Figure 24. Run $\pi_1 \in \mathcal{L}(\overline{\mathcal{M}})$ because the rule μ_1 is never used immediately after rule μ_2 . Run $\pi_2 \notin \mathcal{L}(\overline{\mathcal{M}})$ because the rule μ_1 was used immediately after rule μ_2 : $\vec{\pi}_2[3] < \vec{\pi}_2[2]$.

4.1.3 Programmed rewriting

Informally, for every rule, there exists a set of successor rules. Then, only runs with the run label where each rule is followed by its successor are allowed.

In a *programmed* multiset rewriting system, the regulation $\zeta : \xi \rightarrow 2^\xi$ is a *successor* function, assigning to each rule a set of successor rules. The set of possible runs $\mathcal{L}(\mathcal{M})$ is reduced to a set of runs $\mathcal{L}(\overline{\mathcal{M}}) = \max\{ \pi \in \mathcal{L}_{\text{ALL}}(\mathcal{M}) \mid \forall i > 0. \vec{\pi}[i+1] \in \zeta(\vec{\pi}[i]) \}$.

We denote by \mathbb{PIR} the class of programmed multiset rewriting systems. An example of a \mathbb{PIR} system is available in Figure 26 with an example of valid and invalid runs in Figure 27.

$$\overline{\mathcal{M}} = \left(\begin{array}{l} M_0 = \emptyset, \quad \zeta = \left\{ \mu_1 \rightarrow \{\mu_2\}, \mu_2 \rightarrow \{\mu_1\} \right\}, \\ \xi = \left\{ \mu_1 : \emptyset \rightarrow \{A\}, \mu_2 : \emptyset \rightarrow \{B\} \right\} \end{array} \right)$$

Figure 26: Example of a programmed multiset rewriting system over a set of elements $\mathcal{S} = \{A, B\}$. The regulation ζ allows only alternate application of rules μ_1 and μ_2 .

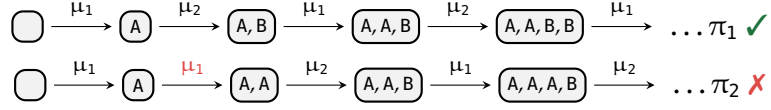


Figure 27: Example of valid and invalid runs of the IIR system from Figure 26.

Run $\pi_1 \in \mathcal{L}(\overline{\mathcal{M}})$ because rules μ_1 and μ_2 are alternating and run $\pi_2 \notin \mathcal{L}(\overline{\mathcal{M}})$ because the run label $\vec{\pi}_2$ violates the successor function, e.g. $\vec{\pi}_2[2] \notin \zeta(\vec{\pi}_2[1])$ (i.e. $\mu_1 \notin \zeta(\mu_1)$).

4.1.4 Conditional rewriting

The idea is to define a multiset of elements (or multiple multisets) for each rule, which represents a *prohibited* context where the rule cannot be used. Then, a run is valid if, in every step, the prohibited context of the used rule is not a subset of the current multiset.

In a *conditional* multiset rewriting system, the regulation ζ is defined as a *conditional* function, assigning a set of multisets over elements \mathcal{S} to each rule. Each of these multisets represents the prohibited context in which the rule cannot be applied, i.e. it is not a subset of the current state. The set of possible runs $\mathcal{L}(\overline{\mathcal{M}})$ is reduced to a set of runs $\mathcal{L}(\overline{\mathcal{M}}) = \max\{\pi \in \mathcal{L}_{\text{ALL}}(\overline{\mathcal{M}}) \mid \forall i > 0. \forall A \in \zeta(\vec{\pi}[i]). A \not\subseteq \pi[i-1]\}$.

We denote by CIR the class of conditional multiset rewriting systems. An example of a CIR system is available in Figure 28 with an example of valid and invalid runs in Figure 29. Note that in the case when there is only one prohibited context defined for a rule, we simplify the notation by omitting the parent set.

$$\overline{\mathcal{M}} = \left(\begin{array}{l} \mathcal{M}_0 = \emptyset, \quad \zeta = \left\{ \mu_1 \rightarrow \{B\}, \mu_2 \rightarrow \emptyset \right\}, \\ \xi = \left\{ \mu_1 : \emptyset \rightarrow \{A\}, \mu_2 : \{A\} \rightarrow \{B\} \right\} \end{array} \right)$$

Figure 28: Example of a conditional multiset rewriting system over a set of elements $\mathcal{S} = \{A, B\}$. The regulation ζ limits the application of rule μ_1 to states where an element B is not present.

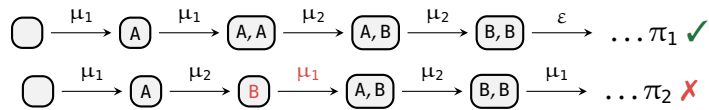


Figure 29: Example of valid and invalid runs of the CIR system from Figure 28.

Run $\pi_1 \in \mathcal{L}(\overline{\mathcal{M}})$ because rule μ_1 is never used in the context of element B. Run $\pi_2 \notin \mathcal{L}(\overline{\mathcal{M}})$ because rule μ_1 was applied in prohibited context: $\{B\} \in \zeta(\vec{\pi}_2[3])$ with $\{B\} \subseteq \pi_2[2]$.

4.1.5 Concurrent-free rewriting

The idea is to detect concurrent rules and assign a priority to one of them. Then, only runs where a prioritised rule was used in place of non-prioritised ones are valid. We say two rules μ, μ' are *concurrent* iff $\bullet\mu \cap \bullet\mu' \neq \emptyset$ (i.e. rules rewrite common elements).

In a *concurrent-free* multiset rewriting system, the regulation $\zeta \subseteq \xi \times \xi$ is a binary relation over rules, for which holds that (1) the relation is irreflexive, (2) any two rules μ, μ' with $(\mu, \mu') \in \zeta$ are concurrent, and (3) if $(\mu, \mu') \in \zeta$, then $(\mu', \mu) \notin \zeta^+$, where ζ^+ is the transitive closure of relation ζ . The third condition ensures acyclicity, which disables ambiguous priority resolving. The set of possible runs $\mathcal{L}(\mathcal{M})$ is reduced to a set of runs $\mathcal{L}(\overline{\mathcal{M}}) = \max\{\pi \in \mathcal{L}_{\text{ALL}}(\mathcal{M}) \mid \forall i > 0 \forall \mu \in \xi \text{ enabled at } \pi[i], (\pi[i+1], \mu) \notin \zeta\}$.

We denote by **CFR** the class of concurrent-free multiset rewriting systems. An example of a **CFR** system is available in Figure 30 with an example of valid and invalid runs in Figure 31.

$$\overline{\mathcal{M}} = \left(\begin{array}{l} M_0 = \{A\}, \quad \zeta = \left\{ (\mu_3, \mu_2) \right\}, \\ \xi = \left\{ \mu_1 : \{A\} \rightarrow \{A, B\}, \mu_2 : \{A, B\} \rightarrow \{A\}, \mu_3 : \{A\} \rightarrow \emptyset \right\} \end{array} \right)$$

Figure 30: Example of a concurrent-free multiset rewriting system over a set of elements $\mathcal{S} = \{A, B\}$. The regulation ζ gives priority to rule μ_2 over the concurrent rule μ_3 in the case they are both enabled, which makes sure element A cannot be removed until any Bs are present. Please note that the priority does not need to be resolved for every concurrent pair of rules (e.g. μ_1 and μ_2).

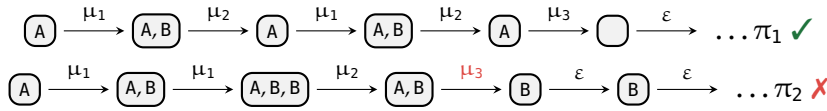


Figure 31: Example of valid and invalid runs of the **CFR** system from Figure 30. Run $\pi_1 \in \mathcal{L}(\overline{\mathcal{M}})$ because rule μ_2 was used with priority when rule μ_3 was also enabled. Run $\pi_2 \notin \mathcal{L}(\overline{\mathcal{M}})$ the rule μ_3 was used when the prioritised rule μ_2 was also enabled.

4.2 PROPERTIES OF REGULATED MULTISET REWRITING SYSTEMS

In this section, we state some general properties of the classes of rewriting systems defined in the previous section. First, we compare the classes on the level of generative power, i.e. runs they can generate, and then we discuss their expressive power, i.e. the functions they can compute (both summarised in Figure 32).

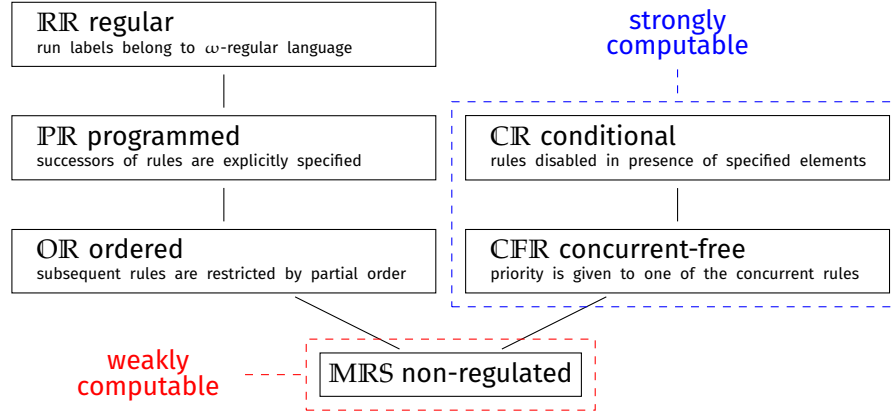


Figure 32: Summary of generative power comparison among classes of regulated systems. Additionally, identified expressive power classes are highlighted in the scheme.

We say two runs π, π' are *equivalent* iff $\forall i \in \mathbb{N}. \pi[i] = \pi'[i]$ (regardless of the run labels). We say two systems $\mathcal{M}, \mathcal{M}'$ are *equivalent* iff their corresponding sets of runs $\mathcal{L}(\mathcal{M}), \mathcal{L}(\mathcal{M}')$ are equal.

Let \mathcal{C} be an arbitrary class of systems. We define the *generative power* of a class \mathcal{C} as a set $g(\mathcal{C}) = \{\mathcal{L}(\mathcal{M}) \mid \mathcal{M} \in \mathcal{C}\}$ of all possible sets of runs. We can compare classes on their level of generative power defining the following operators for any two classes $\mathcal{C}_1, \mathcal{C}_2$:

- $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2$ iff $g(\mathcal{C}_1) \subseteq g(\mathcal{C}_2)$
- $\mathcal{C}_1 = \mathcal{C}_2$ iff $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2 \wedge \mathcal{C}_2 \sqsubseteq \mathcal{C}_1$
- $\mathcal{C}_1 \subset \mathcal{C}_2$ iff $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2 \wedge \mathcal{C}_1 \neq \mathcal{C}_2$
- $\mathcal{C}_1 \not\sqsubseteq \mathcal{C}_2$ iff $\mathcal{C}_1 \not\subseteq \mathcal{C}_2 \wedge \mathcal{C}_2 \not\subseteq \mathcal{C}_1$

We use these properties to analyse and compare individual classes of (regulated) systems in the following section.

4.2.1 Generative power comparison

All regulated classes are strictly more generative than non-regulated class, which follows from the existence of neutral regulation for each type of regulation.

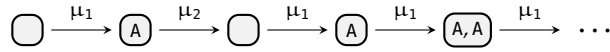
Theorem 2. $\forall \mathcal{C} \in \{\text{RR}, \text{OR}, \text{PR}, \text{CR}, \text{CFR}\}. \text{MRS} \subset \mathcal{C}$

Proof. To show that $\text{MRS} \sqsubseteq \mathcal{C}$, we show that there exists a *neutral* regulation ζ_0 for any class \mathcal{C} such that $\forall \mathcal{M} \in \text{MRS} \exists \overline{\mathcal{M}} \in \mathcal{C}$ with $\overline{\zeta} = \zeta_0$ such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\overline{\mathcal{M}})$. In other words, the neutral regulation places no restriction on the rules application. The neutral regulations for individual classes are defined as follows:

- $\mathbb{RR} : \zeta_\emptyset = \xi^\omega$
- $\mathbb{OR} : \zeta_\emptyset = \emptyset$
- $\mathbb{PR} : \zeta_\emptyset = \{\mu \rightarrow \xi \mid \mu \in \xi\}$
- $\mathbb{CR} : \zeta_\emptyset = \{\mu \rightarrow \emptyset \mid \mu \in \xi\}$
- $\mathbb{CFR} : \zeta_\emptyset = \emptyset$

To show the strictness of the relations, we find an $\overline{\mathcal{M}} \in \mathbb{C}$ such that $\forall \mathcal{M} \in \mathbb{MRS}$ holds $\mathcal{L}(\overline{\mathcal{M}}) \neq \mathcal{L}(\mathcal{M})$. We only show that $\mathbb{MRS} \subsetneq \mathbb{OR}$, and we skip other proofs as they are based on the same idea of showing that their respective example (used above in the class definition) does not have a representative in \mathbb{MRS} with an equal set of runs.

We use $\overline{\mathcal{M}} \in \mathbb{OR}$ from Figure 24 and the example of valid run π_1 from Figure 25. The rule μ_1 applied in step one and μ_2 applied in step four enforce that $\mu_1, \mu_2 \in \xi$ for the $\mathcal{M} \in \mathbb{MRS}$ (because there is no other way these transitions could happen). However, then a run $\pi' \in \mathcal{L}(\mathcal{M})$:



is possible, but clearly run $\pi' \notin \mathcal{L}(\overline{\mathcal{M}})$. From this follows that $\mathcal{L}(\overline{\mathcal{M}}) \neq \mathcal{L}(\mathcal{M})$ and thus the strictness of inclusion. \square

From Theorem 2 also follows that all regulated classes have a common intersection in \mathbb{MRS} . Now we show some relations among regulated classes. There are several subset relations among the classes, namely $\mathbb{OR} \subsetneq \mathbb{PR} \subsetneq \mathbb{RR}$ and $\mathbb{CFR} \subsetneq \mathbb{CR}$. All the other pairs of classes are incomparable, as we will show below.

Theorem 3. $\mathbb{OR} \subsetneq \mathbb{PR}$

Proof. To show that $\mathbb{OR} \subseteq \mathbb{PR}$, we describe how for any $\mathcal{M} \in \mathbb{OR}$ an equivalent system $\mathcal{M}' \in \mathbb{PR}$ can be constructed. The partial order ζ defined on the rules does not allow a rule lower in the order to be used immediately after a rule higher in the order. In other words, rules higher in the order or those incomparable *can* be applied. Using this fact we construct successor function $\zeta'(\mu) = \{\mu' \in \xi \mid (\mu', \mu) \notin \zeta\}$ for all $\mu \in \xi$ and $\xi' = \xi, M'_0 = M_0$.

To prove that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$, we must show that for any state M and rule $\mu \in \xi$ it holds that \mathcal{M} can apply μ to $M \Leftrightarrow \mathcal{M}'$ can apply μ to M . This is trivial for the initial state M_0 since neither of the systems set any restrictions on rule application in the first step (from the definition). For any other general case $\dots \xrightarrow{\mu_{\text{pre}}} \textcircled{M} \xrightarrow{\mu} \dots$ with an enabled rule μ :

\Rightarrow : If \mathcal{M} can apply μ to M , that means that $(\mu, \mu_{\text{pre}}) \notin \zeta$. From that and definition of ζ' follows that $\mu \in \zeta'(\mu_{\text{pre}})$. Therefore, \mathcal{M}' also can apply μ to M .

If \mathcal{M} can *not* apply μ to M , that means that $(\mu, \mu_{\text{pre}}) \in \zeta$. From that and definition of ζ' follows that $\mu \notin \zeta'(\mu_{\text{pre}})$. Therefore, \mathcal{M}' also can *not* apply μ to M .

\Leftarrow : If \mathcal{M}' can apply μ to M , that means that $\mu \in \zeta'(\mu_{pre})$. From the definition of ζ' follows that $(\mu, \mu_{pre}) \notin \zeta$. Therefore, \mathcal{M} also can apply μ to M .

If \mathcal{M}' can *not* apply μ to M , that means that $\mu \notin \zeta'(\mu_{pre})$. From the definition of ζ' follows that $(\mu, \mu_{pre}) \in \zeta$. Therefore, \mathcal{M} also can *not* apply μ to M .

To show the strictness of the inclusion, we find an $\mathcal{M} \in \mathbb{PR}$ such that $\forall \mathcal{M}' \in \mathbb{OR}$ holds $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. We use $\mathcal{M} \in \mathbb{PR}$ from Figure 26. The example of a valid run π_1 from Figure 27 enforces that $\mu_1 \in \xi'$ due to its first step. The symmetric run π_1' :

$$\bigcirc \xrightarrow{\mu_2} \boxed{B} \xrightarrow{\mu_1} \boxed{A, B} \xrightarrow{\mu_2} \boxed{A, B, B} \xrightarrow{\mu_1} \boxed{A, A, B, B} \xrightarrow{\mu_2} \dots \in \mathcal{L}(\mathcal{M})$$

enforces that $\mu_2 \in \xi'$ also due to its first step. Then, for the ordered system \mathcal{M}' , there are three options w.r.t. ζ' how these two rules can be related (regardless of other rules in ξ'):

$$\begin{array}{ll} \mu_1, \mu_2 \text{ are incomparable} & \bigcirc \xrightarrow{\mu_1} \boxed{A} \xrightarrow{\mu_1} \boxed{A, A} \xrightarrow{\mu_1} \boxed{A, A, A} \xrightarrow{\mu_1} \dots \notin \mathcal{L}(\mathcal{M}) \\ \mu_1 < \mu_2 & \bigcirc \xrightarrow{\mu_1} \boxed{A} \xrightarrow{\mu_2} \boxed{A, B} \xrightarrow{\mu_2} \boxed{A, B, B} \xrightarrow{\mu_2} \dots \notin \mathcal{L}(\mathcal{M}) \\ \mu_2 < \mu_1 & \bigcirc \xrightarrow{\mu_2} \boxed{B} \xrightarrow{\mu_2} \boxed{B, B} \xrightarrow{\mu_1} \boxed{A, B, B} \xrightarrow{\mu_1} \dots \notin \mathcal{L}(\mathcal{M}) \end{array}$$

for each case, there is a run valid in $\mathcal{L}(\mathcal{M}')$ which is invalid in $\mathcal{L}(\mathcal{M})$, i.e. $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. \square

Theorem 4. $\mathbb{PR} \sqsubset \mathbb{RR}$

Proof. To show that $\mathbb{PR} \sqsubseteq \mathbb{RR}$, it is enough to take the regular language where in every run label, each pair of neighbouring rules satisfies the condition of the programmed rewriting system. This is accomplished by the unlimited memory provided by regular rewriting systems. Such language can be easily described by an automaton.

To show the strictness of the inclusion, we find an $\mathcal{M} \in \mathbb{RR}$ such that $\forall \mathcal{M}' \in \mathbb{PR}$ holds $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. We use the following $\mathcal{M} \in \mathbb{RR}$:

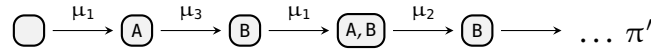
$$\mathcal{M} = \left(\begin{array}{l} M_0 = \emptyset, \quad \zeta = \left\{ \mu_1 \cdot \mu_2 \cdot \mu_1 \cdot \mu_3 \cdot \varepsilon^\omega \right\}, \\ \xi = \left\{ \mu_1 : \emptyset \rightarrow \{A\}, \mu_2 : \{A\} \rightarrow \emptyset, \mu_3 : \{A\} \rightarrow \{B\} \right\} \end{array} \right)$$

with $\mathcal{S} = \{A, B\}$. In this system, rules can be used only in a particular finite sequence, followed by infinite application of empty rule ε . Therefore, the set of runs $\mathcal{L}(\mathcal{M})$ contains only one possible run π :

$$\bigcirc \xrightarrow{\mu_1} \boxed{A} \xrightarrow{\mu_2} \bigcirc \xrightarrow{\mu_1} \boxed{A} \xrightarrow{\mu_3} \boxed{B} \xrightarrow{\varepsilon} \dots$$

The run π actually enforces all the rules μ_1, μ_2, μ_3 to be present in ξ' due to steps one, two, and four, respectively. Let us focus on the successor function ζ' for rule μ_1 w.r.t. rules μ_2, μ_3 . There are four options:

$\mu_2 \notin \zeta'(\mu_1) \wedge \mu_3 \notin \zeta'(\mu_1)$... run $\pi \notin \mathcal{L}(\mathcal{M}')$ because μ_2 cannot be used after μ_1
$\mu_2 \notin \zeta'(\mu_1) \wedge \mu_3 \in \zeta'(\mu_1)$... run $\pi \notin \mathcal{L}(\mathcal{M}')$ because μ_2 cannot be used after μ_1
$\mu_2 \in \zeta'(\mu_1) \wedge \mu_3 \notin \zeta'(\mu_1)$... run $\pi \notin \mathcal{L}(\mathcal{M}')$ because μ_3 cannot be used after μ_1
$\mu_2 \in \zeta'(\mu_1) \wedge \mu_3 \in \zeta'(\mu_1)$... run $\pi' \in \mathcal{L}(\mathcal{M}')$ while $\pi' \notin \mathcal{L}(\mathcal{M})$ specified as:



In each case holds $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. □

Corollary 1. $\text{OR} \sqsubset \text{RR}$

Proof. The property follows directly from $\text{OR} \sqsubset \text{PR}$ (Theorem 3) and $\text{PR} \sqsubset \text{RR}$ (Theorem 4). □

Theorem 5. $\text{CFR} \sqsubset \text{CR}$

Proof. To show that $\text{CFR} \sqsubseteq \text{CR}$, we describe how for any $\mathcal{M} \in \text{CFR}$ an equivalent system $\mathcal{M}' \in \text{CR}$ can be constructed. First, we set $\mathcal{M}'_0 = \mathcal{M}_0$, $\xi' = \xi$, and $\zeta'(\mu) = \emptyset$ for all $\mu \in \xi'$. Then, we investigate each pair (μ, μ') in the relation ζ (i.e. μ, μ' are concurrent) individually:

1. if $\bullet\mu \supseteq \bullet\mu'$
 - rule μ can never be used \Rightarrow remove it from ξ'
2. if $\bullet\mu \subset \bullet\mu'$ or $\bullet\mu, \bullet\mu'$ are incomparable
 - rule μ can never be used when μ' is enabled, then extend the forbidden context $\zeta'(\mu) = \{\bullet\mu'\} \cup \zeta'(\mu)$

To prove that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$, we must show that for any state M and enabled rule $\mu \in \xi$ it holds that \mathcal{M} can apply μ to $M \Leftrightarrow \mathcal{M}'$ can apply μ to M :

\Rightarrow : If \mathcal{M} can apply μ to M , that means that $\forall \mu' \in \xi$ enabled at M such that $(\mu, \mu') \notin \zeta$. No forbidden context was introduced for rule μ , therefore also \mathcal{M}' can apply μ to M .

If \mathcal{M} can *not* apply μ to M , that means that $\exists \mu' \in \xi$ enabled at M such that $(\mu, \mu') \in \zeta$. In case (1), rule μ is removed from ξ' and therefore also \mathcal{M}' can *not* apply μ to M . In case (2), the new forbidden context $\bullet\mu'$ is introduced for rule μ ; since μ' is obviously enabled, it means that also $\bullet\mu' \subset M$ and therefore \mathcal{M}' also can *not* apply μ to M .

\Leftarrow : If \mathcal{M}' can apply μ to M , that means that $\forall \mathcal{A} \in \zeta'(\mu)$ holds that $\mathcal{A} \not\subseteq M$. Since for case (1) obviously $\mu \in \xi'$ and there was no forbidden context introduced in case (2), there is no concurrent rule $\mu' \in \xi$ with $(\mu, \mu') \in \zeta$. Therefore also \mathcal{M} can apply μ to M .

If \mathcal{M}' can *not* apply μ to M , that means that $\exists \mathcal{A} \in \zeta'(\mu)$ such that $\mathcal{A} \subseteq M$. Since for case (1) obviously $\mu \in \xi'$, there was some forbidden context introduced in case (2). From that follows there exists a concurrent rule $\mu' \in \xi$ enabled at M with $(\mu, \mu') \in \zeta$. Therefore \mathcal{M} also can *not* apply μ to M .

To show the strictness of the inclusion, we find an $\mathcal{M} \in \text{CIR}$ such that $\forall \mathcal{M}' \in \text{CFIR}$ holds $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. We use the following $\mathcal{M} \in \text{CIR}$:

$$\mathcal{M} = \left(\begin{array}{l} M_0 = \emptyset, \quad \zeta = \left\{ \mu_1 \rightarrow \{C\}, \mu_2 \rightarrow \{B\} \right\}, \\ \xi = \left\{ \mu_1 : \emptyset \rightarrow \{B\}, \mu_2 : \emptyset \rightarrow \{C\} \right\} \end{array} \right)$$

with $\mathcal{S} = \{B, C\}$ where using one of the rules disables the other one. Therefore, the set of runs $\mathcal{L}(\mathcal{M})$ contains only two possible runs π_1 and π_2 :

$$\begin{array}{l} \emptyset \xrightarrow{\mu_1} (B) \xrightarrow{\mu_1} (B, B) \xrightarrow{\mu_1} (B, B, B) \xrightarrow{\mu_1} \dots \pi_1 \in \mathcal{L}(\mathcal{M}) \\ \emptyset \xrightarrow{\mu_2} (C) \xrightarrow{\mu_2} (C, C) \xrightarrow{\mu_2} (C, C, C) \xrightarrow{\mu_2} \dots \pi_2 \in \mathcal{L}(\mathcal{M}) \end{array}$$

To ensure at least the first step in both runs, it has to hold that both rules $\mu_1, \mu_2 \in \xi'$ for $\mathcal{M}' \in \text{CFIR}$. Since these two rules are not concurrent, we cannot restrict their application at all using ζ' . From that follows, there will be some additional runs present in $\mathcal{L}(\mathcal{M}')$ and therefore $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. \square

All the other classes are incomparable. We now investigate individual relationships. The proofs are given by showing counterexamples and using already proven relations.

Theorem 6. $\text{OR} \not\subseteq \text{CFIR}$

Proof. To show that $\text{OR} \not\subseteq \text{CFIR}$, we find an $\mathcal{M} \in \text{OR}$ such that $\forall \mathcal{M}' \in \text{CFIR}$ holds that $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. We use $\mathcal{M} \in \text{OR}$ from [Figure 24](#). Based on proof of [Theorem 2](#), both rules μ_1, μ_2 belong to ξ' . However, these rules are not concurrent, so they can be used in an arbitrary fashion in any $\mathcal{M}' \in \text{CFIR}$. For example, a run:

$$\emptyset \xrightarrow{\mu_1} (A) \xrightarrow{\mu_2} \emptyset \xrightarrow{\mu_1} (A) \xrightarrow{\mu_1} (A, A) \longrightarrow \dots$$

is possible in $\mathcal{L}(\mathcal{M}')$ but not in $\mathcal{L}(\mathcal{M})$, from that $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$.

To show that $\text{CFIR} \not\subseteq \text{OR}$, we find an $\mathcal{M} \in \text{CFIR}$ such that $\forall \mathcal{M}' \in \text{OR}$ holds that $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. We use the following $\mathcal{M} \in \text{CFIR}$:

$$\mathcal{M} = \left(\begin{array}{l} M_0 = \{A, B\}, \quad \zeta = \left\{ (\mu_2, \mu_1) \right\}, \\ \xi = \left\{ \mu_1 : \{A, B\} \rightarrow \{A, C\}, \mu_2 : \{A\} \rightarrow \{A, B\} \right\} \end{array} \right)$$

with $\mathcal{S} = \{A, B, C\}$. In this system, rules are alternating due to the defined priority on rule μ_1 . The set of runs $\mathcal{L}(\mathcal{M})$ contains only one possible run π :

$$\boxed{A, B} \xrightarrow{\mu_1} \boxed{A, C} \xrightarrow{\mu_2} \boxed{A, B, C} \xrightarrow{\mu_1} \boxed{A, C, C} \xrightarrow{\mu_2} \boxed{A, B, C, C} \xrightarrow{\mu_1} \dots$$

which implies that at least two rules $\mu_I, \mu_{II} \in \xi'$ such that $B \in \bullet\mu_I \wedge C \in \mu_I^\bullet$ (from the first step) and $B \in \mu_{II}^\bullet$ (from the second step). Then, analogously to proof of [Theorem 3](#), there are three options (incomparable and ordered in either direction) and for each option, there exists a run possible in $\mathcal{L}(\mathcal{M}')$ but not possible in $\mathcal{L}(\mathcal{M})$, i.e. $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$.

Another option would be to create a unique rule $\mu_i : \pi[i-1] \rightarrow \pi[i]$ for each step i and placing this rule above each rule μ_j with $j < i$ in the order ζ' . However, this would require that ξ' is infinite, which is not allowed by definition. \square

Theorem 7. $\text{CFR} \not\sqsubseteq \text{PR}$

Proof. The property $\text{PR} \not\sqsubseteq \text{CFR}$ follows directly from $\text{OR} \sqsubseteq \text{CFR}$ ([Theorem 6](#)) and $\text{OR} \sqsubset \text{PR}$ ([Theorem 3](#)).

To show that $\text{CFR} \not\sqsubseteq \text{PR}$, we find an $\mathcal{M} \in \text{CFR}$ such that $\forall \mathcal{M}' \in \text{PR}$ holds that $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. We use $\mathcal{M} \in \text{CFR}$ from [Figure 30](#).

First, we make a general observation about any $\mathcal{M}' \in \text{PR}$: due to the presence of the successor function, the rules of a PR system can be seen as a directed *successor graph*, where vertices are the rules and edges are the possible successors. Such a graph of the \mathcal{M}' system must contain at least one cycle, which (in summation) increases the number of elements B in the state of the system. Let c be the maximal number of B s produced by any such cycle in this graph.

Next, let $\pi_I \in \mathcal{L}(\mathcal{M})$ be a run which produces $2c$ of B s, then immediately discards them and removes the element A with rule μ_3 . Clearly, \mathcal{M}' must contain rule μ_3 (e.g. fifth step of run π_1 in [Figure 31](#)), and this rule is applied at the end of this run (neglecting the empty rule ε). Additionally, since the number of B s produced is $2c$, the run must contain at least one aforementioned cycle from the successor graph.

Finally, let us consider another run π_{II} which extends the run π_I by adding one repetition of this cycle. The run π_{II} is still valid in \mathcal{M}' , by definition of the successor graph, and the last transition of this run is still μ_3 . However, the number of produced B s is higher than the number of B s consumed. Such a run is not possible in \mathcal{M} because μ_3 can be fired only when all B s have been consumed. Therefore $\pi_{II} \notin \mathcal{L}(\mathcal{M})$ and from that $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. \square

Theorem 8. $\text{OR} \sqsubseteq \text{CR}$

Proof. The property $\text{CR} \not\sqsubseteq \text{OR}$ follows directly from $\text{OR} \sqsubseteq \text{CFR}$ ([Theorem 6](#)) and $\text{CFR} \sqsubset \text{CR}$ ([Theorem 5](#)).

To show that $\text{OR} \sqsubseteq \text{CR}$, we find an $\mathcal{M} \in \text{OR}$ such that $\forall \mathcal{M}' \in \text{CR}$ holds that $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. We use $\mathcal{M} \in \text{OR}$ from [Figure 24](#). As

it was stated in the proof of [Theorem 2](#), both rules $\mu_1, \mu_2 \in \xi'$ of the $\mathcal{M}' \in \mathbb{CR}$. Due to the minimalistic nature of this system, it can be shown for any case of ζ' definition (assuming adding prohibited context with increased number of A's has no effect in this case), that $\mathcal{L}(\mathcal{M}')$ contains more possible runs:

$$\begin{array}{l} \zeta'(\mu_1) = \emptyset \\ \wedge \\ \zeta'(\mu_2) = \emptyset \end{array} \quad \square \xrightarrow{\mu_1} \textcircled{A} \xrightarrow{\mu_2} \square \xrightarrow{\mu_1} \textcircled{A} \xrightarrow{\mu_2} \dots \notin \mathcal{L}(\mathcal{M})$$

$$\begin{array}{l} \zeta'(\mu_1) = \{A\} \\ \wedge \\ \zeta'(\mu_2) = \emptyset \end{array} \quad \square \xrightarrow{\mu_1} \textcircled{A} \xrightarrow{\mu_2} \square \xrightarrow{\mu_1} \textcircled{A} \xrightarrow{\mu_2} \dots \notin \mathcal{L}(\mathcal{M})$$

$$\begin{array}{l} \zeta'(\mu_1) = \emptyset \\ \wedge \\ \zeta'(\mu_2) = \{A\} \end{array} \quad \square \xrightarrow{\mu_1} \textcircled{A} \xrightarrow{\mu_1} \textcircled{A,A} \xrightarrow{\mu_1} \textcircled{A,A,A} \xrightarrow{\mu_1} \dots \notin \mathcal{L}(\mathcal{M})$$

$$\begin{array}{l} \zeta'(\mu_1) = \{A\} \\ \wedge \\ \zeta'(\mu_2) = \{A\} \end{array} \quad \square \xrightarrow{\mu_1} \textcircled{A} \xrightarrow{\varepsilon} \textcircled{A} \xrightarrow{\varepsilon} \textcircled{A} \xrightarrow{\varepsilon} \dots \notin \mathcal{L}(\mathcal{M})$$

from that follows $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. □

Corollary 2. $\mathbb{PR} \not\sqsubseteq \mathbb{CR}$

Proof. The property $\mathbb{PR} \not\sqsubseteq \mathbb{CR}$ follows directly from $\mathbb{OR} \sqsubseteq \mathbb{CR}$ ([Theorem 8](#)) and $\mathbb{OR} \sqsubset \mathbb{PR}$ ([Theorem 3](#)). The property $\mathbb{CR} \not\sqsubseteq \mathbb{PR}$ follows directly from $\mathbb{CFR} \sqsubseteq \mathbb{PR}$ ([Theorem 7](#)) and $\mathbb{CFR} \sqsubset \mathbb{CR}$ ([Theorem 5](#)). □

Theorem 9. $\mathbb{RR} \not\sqsubseteq \mathbb{CFR}$

Proof. To show that $\mathbb{RR} \not\sqsubseteq \mathbb{CFR}$, we find an $\mathcal{M} \in \mathbb{RR}$ such that $\forall \mathcal{M}' \in \mathbb{CFR}$ holds that $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. We use the $\mathcal{M} \in \mathbb{RR}$ from the proof of [Theorem 4](#). As stated in the proof, all rules μ_1, μ_2, μ_3 belong to ξ' . Rules μ_2, μ_3 are concurrent and there are three options how to resolve the concurrency:

$$\begin{array}{ll} \text{concurrency is} & \square \xrightarrow{\mu_1} \textcircled{A} \xrightarrow{\mu_2} \square \xrightarrow{\mu_1} \textcircled{A} \xrightarrow{\mu_2} \square \longrightarrow \\ \text{not resolved} & \\ (\mu_2, \mu_3) \in \zeta' & \square \xrightarrow{\mu_1} \textcircled{A} \xrightarrow{\mu_2} \square \xrightarrow{\mu_1} \textcircled{A} \xrightarrow{\mu_2} \square \longrightarrow \\ (\mu_3, \mu_2) \in \zeta' & \square \xrightarrow{\mu_1} \textcircled{A} \xrightarrow{\mu_3} \textcircled{B} \xrightarrow{\mu_1} \textcircled{A,B} \xrightarrow{\mu_3} \textcircled{B,B} \longrightarrow \end{array}$$

where in each case the run does not belong to $\mathcal{L}(\mathcal{M})$ and therefore holds that $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$.

To show that $\text{CFR} \not\sqsubseteq \text{RR}$, we find an $\mathcal{M} \in \text{CFR}$ such that $\forall \mathcal{M}' \in \text{RR}$ holds that $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. We use the following $\mathcal{M} \in \text{CFR}$:

$$\mathcal{M} = \left(\begin{array}{l} M_0 = \{C\}, \quad \zeta = \left\{ (\mu_4, \mu_3) \right\}, \\ \xi = \left\{ \begin{array}{l} \mu_1 : \{C\} \rightarrow \{A, C\}, \mu_2 : \{C\} \rightarrow \{B\} \\ \mu_3 : \{A, B\} \rightarrow \{B\}, \mu_4 : \{B\} \rightarrow \emptyset \end{array} \right\} \end{array} \right)$$

with $\mathcal{S} = \{A, B, C\}$. In this system, a number $n \in \mathbb{N}$ of elements A is generated, followed by the change of C to a B and finally, all elements A are discarded, including symbol B . This is ensured by giving priority to rule μ_3 over rule μ_4 . Please note the concurrency between rules μ_1 and μ_2 is not resolved (which is allowed by definition). Since runs:

$$\begin{array}{ccccccc} \textcircled{C} & \xrightarrow{\mu_1} & \textcircled{A, C} & \xrightarrow{\mu_2} & \textcircled{A, B} & \xrightarrow{\mu_3} & \textcircled{B} \xrightarrow{\mu_4} \textcircled{} \xrightarrow{\varepsilon} \dots \pi_1 \\ \textcircled{C} & \xrightarrow{\mu_2} & \textcircled{B} & \xrightarrow{\mu_4} & \textcircled{} & \xrightarrow{\varepsilon} & \textcircled{} \xrightarrow{\varepsilon} \dots \pi_2 \end{array}$$

are both allowed, both $\mu_2, \mu_4 \in \xi'$ and also $\mu_I, \mu_{III} \in \xi'$ such that $A \in \mu_I^\bullet$ (from the first step) and $A \in \bullet \mu_{III}$ (from the third step).

Let $\mathcal{A} = \{\mu_I^n \cdot \mu_2 \cdot \mu_{III}^n \cdot \mu_4 \cdot \varepsilon^\omega \mid n \in \mathbb{N}\}$ be an ω -language over rules $\mu_I, \mu_2, \mu_{III}, \mu_4, \varepsilon$. For the system \mathcal{M} , it holds that $\pi \in \mathcal{L}(\mathcal{M}) \Leftrightarrow \vec{\pi} \in \mathcal{A}$ (such that instead of rules μ_I, μ_{III} their possible instances μ_1, μ_3 are used, respectively). However, language \mathcal{A} is obviously not ω -regular. The *smallest* ω -regular language \mathcal{R} that contains language \mathcal{A} (i.e. $\mathcal{A} \subset \mathcal{R}$) is given by ω -regular expression $\mu_I^* \cdot \mu_2 \cdot \mu_{III}^* \cdot \mu_4 \cdot \varepsilon^\omega$. But this language contains also runs where μ_{III} is applied fewer times than μ_I – there can be some spare A s when no other rule except ε can be applied. This does not correspond to $\mathcal{L}(\mathcal{M})$. We would observe the same issue in any language which contains \mathcal{R} , and therefore such an $\mathcal{M}' \in \text{RR}$ does not exist. \square

Theorem 10. $\text{RR} \not\sqsubseteq \text{CFR}$

Proof. To show that $\text{RR} \not\sqsubseteq \text{CFR}$, we find an $\mathcal{M} \in \text{RR}$ such that $\forall \mathcal{M}' \in \text{CFR}$ holds that $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$. We use the $\mathcal{M} \in \text{RR}$ from the proof of Theorem 4. From the only possible run π it follows that rules μ_2, μ_3 need to be applicable in the context of element A while it is only possible candidate to be in their prohibited context. Since that cannot be done, there will be some additional runs in $\mathcal{L}(\mathcal{M}')$. From that $\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{M}')$.

The property $\text{CFR} \not\sqsubseteq \text{RR}$ follows directly from $\text{RR} \not\sqsubseteq \text{CFR}$ (Theorem 9) and $\text{CFR} \sqsubseteq \text{CFR}$ (Theorem 5). \square

4.2.2 Expressive power

In this section, we discuss *expressive power* for some of the defined classes. While the generative power discussed above concerns the

set of runs that can be generated, the expressive power studies *functions*, which the formalism can compute.

A multiset rewriting system (with or without regulation) *strongly computes* a numerical function $f : \mathbb{N} \rightarrow \mathbb{N}$ by starting in an initial multiset M_0 containing an input element I with $M_0(I) = n$ and always reaches a final multiset M_f (all subsequent multisets are equal) containing an output element O with $M_f(O) = f(n)$. In order for the system to be correct, there cannot exist a run where for the final multiset M_f holds that $M_f(O) \neq f(n)$.

Since this definition does not accommodate nondeterminism nicely (which is natural for multiset rewriting), we say a multiset rewriting system *weakly computes* a numerical function $f : \mathbb{N} \rightarrow \mathbb{N}$ if there exists a run with $M_f(O) = f(n)$ and for any other run holds that $M_f(O) \leq f(n)$. This definition is the adoption of weak computability for Petri nets [LS14].

Theorem 11. *MRS weakly compute all computable numerical functions.*

For the proof, we refer to [Cer94; Cer95] where it is shown that the expressive power of MRS is equivalent to the expressive power of Petri Nets, because they are both based on simple operations on positive integer counters – decrements and increments. More precisely, they cannot compute all recursive functions because they lack zero-tests which makes them less expressive than Turing machine – in particular, they weakly compute numerical functions [LS14].

Theorem 12. *CR and CFR strongly compute all computable numerical functions.*

Proof. Assuming the Church-Turing thesis is correct, the simplest way how to prove this theorem is in terms of the Minsky register machine [Min67]. Minsky machine has a number of registers storing arbitrary large numbers. A program is a sequence of instructions manipulating the registers. It was shown that two register machines with the following instruction set can compute all computable numerical functions (shown by equivalence to Turing machine [SS63]):

l_1 : Com₁;
 l_2 : Com₂;
 \vdots
 l_m : Com_m;
 l_{m+1} : halt;

where $m \geq 0$ and every Com_i is one of two types:

- Type I: $c_j := c_j + 1$; goto l_p
- Type II: if $c_j = 0$ then goto l_p else $c_j := c_j - 1$; goto l_q

with counter index $j \in \{1, 2\}$ and $p, q \in \{1, \dots, m+1\}$.

The computation of the machine with the counters set to initial values (with $c_1 = n$) starts at l_1 and proceeds consistently with the intuitive semantics of the instructions. We say that the machine *halts* if the computation eventually reaches the `halt` instruction. Finally, the result of the computation is the value of counter c_2 .

Thus, if a register machine can be converted into an equivalent conditional (resp. concurrent-free) regulated MRS, we see that $\mathbb{C}\mathbb{R}$ (resp. $\mathbb{C}\mathbb{F}\mathbb{R}$) can compute all computable numerical functions. First, we show this for $\mathbb{C}\mathbb{R}$. The proof for $\mathbb{C}\mathbb{F}\mathbb{R}$ is identical except for a minor modification specific to its regulatory approach.

To represent a register machine as an $\mathcal{M} \in \mathbb{C}\mathbb{R}$, we represent the two registers by two elements c_1, c_2 (such that initially c_1 has n repetitions and c_2 represents the output). We use elements l_1, \dots, l_m, l_{m+1} to represent the position of the program. The instructions are represented as follows:

- Type I: introduce rule $\mu_i : \{l_i\} \rightarrow \{l_p, c_j\}$
- Type II: introduce rules $\mu_i : \{l_i\} \rightarrow \{l_p\}$ and $\bar{\mu}_i : \{l_i, c_j\} \rightarrow \{l_q\}$ and set $\zeta(\mu_i) = \{c_j\}$

Defining prohibited context to the rule μ_i in Type II instruction effectively prioritises rule $\bar{\mu}_i$ until the counter c_j is not zero. Finally, the instruction `halt` is represented by the application of rule ε .

This shows that a register machine can be converted into $\mathbb{C}\mathbb{R}$. In the case of $\mathbb{C}\mathbb{F}\mathbb{R}$, the only modification is in the definition of regulation ζ in the Type II instruction – we require that $(\mu_i, \bar{\mu}_i) \in \zeta$, which ensures that $\bar{\mu}_i$ has priority until the counter c_j is not zero. \square

4.3 REGULATED BCSL

Since in [Section 3.3](#) we related BCSL to MRS, this allows us to use regulations on the level of BCSL models. It also results in propagation of theoretical findings of expressivity to the context of BCSL. In the following subsections, we show the elemental usage of regulations on a simple BCSL model, and we show how regulation changes the behaviour of the model.

4.3.1 Regular rewriting

An ω -regular expression (RE) is a pattern that defines sequences of rules. Such an approach can define sequences of rules that are allowed explicitly. These, in practice, describe particular executions of the model. For simplicity, we omit the ε^ω suffix in the following expressions.

The most general RE has the form $\{r_1, r_2, r_3, \dots\}^*$, which allows any possible sequence of rules corresponding to unregulated behaviour.

On the other hand, the least permissive REs have the form $r_1 r_2 r_3$, which allows only this particular sequence of rules to be used. Then, some more variable options can include, for example, choice from a set of candidates on some fixed positions $r_1 \{r_2, r_4\}^* r_3$ or specification of several alternatives using boolean “or” $r_1 r_2 | r_3 r_4$.

For example, let us use an RE which allows either a sequence of three rules $r1_S$, $r1_T$, and $r2$ or a sequence of two rules $r1_T$ and $r1_S$. Employing such RE, written $r1_S r1_T r2 | r1_T r1_S$, as a regulation for the model from Figure 18 ensures that first both activation rules are used and then the molecule is exported out of the cell, depending on the order of activation. The effect of regulation on the transition system is depicted in Figure 33.

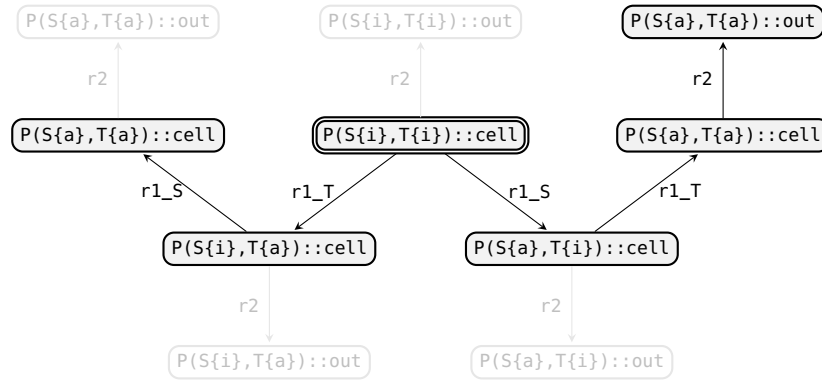


Figure 33: Transition system of the model from Figure 18 with applied regular regulation given by RE $r1_S r1_T r2 | r1_T r1_S$. The faded states and transitions are absent due to the effects of the regulation.

4.3.2 Ordered rewriting

We use ordered rewriting to define an order of rules and then require that any rule sequence has to respect it in a pair-wise fashion. In other words, the latter rule is greater than the former in any pair of subsequent rules. Moreover, we assume the so-called partial order, which allows some rules to be incomparable. Incomparable rules can be applied after each other in an arbitrary fashion.

For example, in a partial order given by $r1_S > r2$ and $r1_T > r2$ both rules $r1_S$ and $r1_T$ are greater than $r2$ while rules $r1_S$ and $r1_T$ are incomparable. Applied as a regulation to the model from Figure 18, this order ensures that rule $r2$ is never used immediately after neither $r1_S$ nor $r1_T$. The effect of regulation on the transition system is depicted in Figure 34.

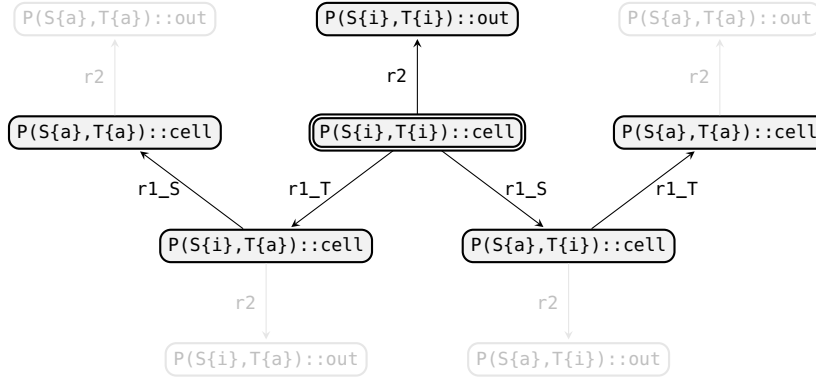


Figure 34: Transition system of the model from Figure 18 with applied ordered regulation given by $r1_S > r2$ and $r1_T > r2$. The faded states and transitions are absent due to the effects of the regulation.

4.3.3 Programmed rewriting

A successor function of a rule defines the rules which can be used in the next step. This allows limiting the set of possible successors to an admissible subset. When a particular rule is used, only its successors are allowed to be used in the next step. For a particular rule, it is possible to define as successors the complete set of rules (without restrictions), a proper subset (some of the rules cannot be used immediately after), or an empty set, which makes the rule terminal (no other action can be done after).

For example, let us have a successor function given by $r1_S \rightarrow \{r1_T, r2\}$ and $r1_T \rightarrow \{r1_S\}$. For rule $r2$, there are no successors defined, which is implicitly interpreted as an empty set. Using this function as a regulation of the model from Figure 18 ensures that rule $r2$ is used only after rule $r1_S$, never after rule $r1_T$. The effect of regulation on the transition system is depicted in Figure 35.

4.3.4 Conditional rewriting

Generally, a rule can be used if there are enough agents in a state where it is being applied. A natural extension to this mechanism is to define agents, which, when present in the state, prohibit the usage of the rule. Conditional rewriting defines a *prohibited* context to each rule, that is, a set of agents which cannot be present in the current state. If present, the rule cannot be used. If the defined context is an empty set, rule applicability has no limitations.

For example, we define prohibited context $\{P(S\{a\}, T\{i\})::cell\}$ for rule $r2$ and leave it empty for rules $r1_T$ and $r1_S$ (i.e. no restrictions). As a regulation for the model from Figure 18, it ensures that rule $r2$ is never used when agent $P(S\{a\}, T\{i\})::cell$ is present in the current

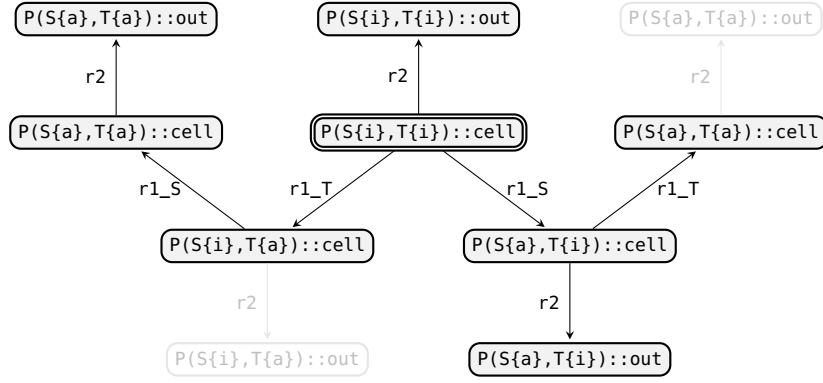


Figure 35: Transition system of the model from Figure 18 with applied programmed regulation given by $r1_S \rightarrow \{r1_T, r2\}$ and $r1_T \rightarrow \{r1_S\}$. The faded states and transitions are absent due to the effects of the regulation.

state. The effect of regulation on the transition system is depicted in Figure 36.

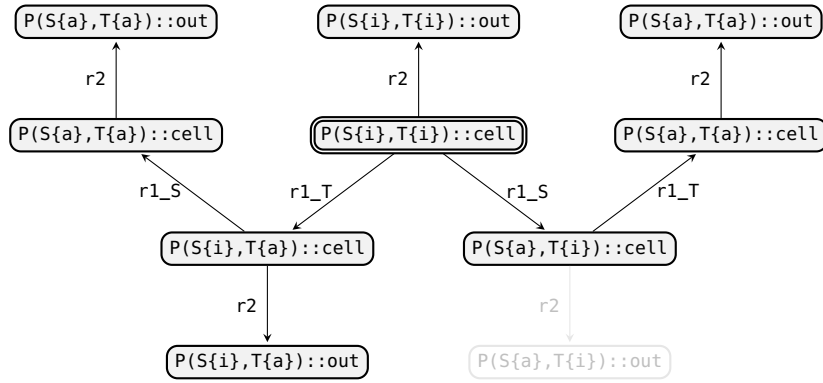


Figure 36: Transition system of the model from Figure 18 with applied conditional regulation given by $r2 \rightarrow \{P(S\{a\}, T\{i\})::cell\}$. The faded states and transitions are absent due to the effects of the regulation.

4.3.5 Concurrent-free rewriting

Concurrency is a natural process in biology, and controlling concurrent processes may be desirable. In the context of rules, such processes are expressed by rules which consume mutual agents. We allow assigning a priority to one of them for such concurrent rules. Only the prioritised one can be used whenever multiple concurrent rules are applicable in a state.

For example, we give priority to both rules $r1_S$ and $r1_T$ over rule $r2$ (i.e. $r1_S > r2$ and $r1_T > r2$). Using it as a regulation for the

model from [Figure 18](#), the rules $r1_S$ and $r1_T$ are always used instead of rule $r2$ when both are enabled. The effect of regulation on the transition system is depicted in [Figure 37](#).

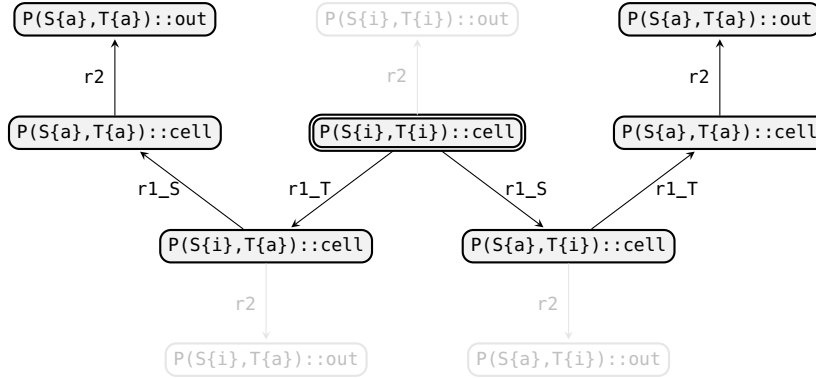


Figure 37: Transition system of the model from [Figure 18](#) with applied concurrent-free regulation given by $r1_S > r2$ and $r1_T > r2$. The faded states and transitions are absent due to the effects of the regulation.

4.4 SUMMARY

This chapter introduced five regulation classes for MRS that can be used in the context of BCSL thanks to its established relationship to MRS. While formally, the regulations restrict the applicability of rules, leading to a reduced set of possible runs, on the practical side, they allow simplifying the system description or substituting the missing details about the modelled system.

Besides introducing the regulated classes, we also provided and proved some theoretical results about relationships among them regarding the runs they can generate. We also investigated the increased expressive power of selected classes. Then we briefly applied individual regulations to a minimal BCSL model and demonstrated their effects on its qualitative behaviour. For some more elaborated examples, we recommend [Section 7.4](#), where we demonstrate the usage of regulations on several case studies from the biochemical domain.

*The profound study of nature is the
most fertile source of mathematical
discovery.*

JEAN-BAPTISTE JOSEPH FOURIER

THIS chapter establishes an analysis base for BCSL. Since there are multiple variants of the models, we also reflect this fact by appropriate analysis types. For the qualitative variant, we propose CTL model checking and several static analysis techniques. The quantitative variant can be examined using two types of simulations and PCTL model checking. Finally, for the parametrised variant, methods for parameter synthesis and robustness analysis are proposed.

5.1 SIMULATION

The most usual way how to observe the quantitative behaviour of a rule-based model is a simulation. This section focuses on the technical aspects of simulating rule-based models and the possible solutions we employ. In particular, we explain two approaches available for BCSL, that is, stochastic simulation using Gillespie's algorithm [Gil76], and deterministic simulation [PReo] by translation of reaction network to ODEs [Higo8].

From the previous chapters, we already know that the state of a BCSL model is a multiset of molecules (instantiated agents). To be accurate, we should start with a position and a velocity for each molecule and let the system evolve under appropriate laws of physics, keeping track of collisions between molecules and the resulting interactions. Since such an approach is typically computationally too expensive, we assume *well-stirred* system (i.e. species of each type are spread uniformly throughout the spatial domain), allowing us, in general, to ignore spatial properties and just keep track of the number of species. For the same reason, it is also good practice to assume that the system is in thermal equilibrium and that the volume of the spatial domain is constant.

When we know the amounts of molecules at time $t = 0$, i.e. in the initial state, the aim of the simulation is to describe how these amounts evolve as time increases. In this setting, we assume a *state vector* $\mathbf{X}(t) = [X_1(t), X_2(t), \dots, X_n(t)]$ where $X_i(t)$ is an integer repre-

senting the number of molecules at position i present in the state at time t .

The state vector $\mathbf{X}(t)$ can change with the application of a rule. Since typically multiple rules can be enabled in a state, we are thinking in terms of the probability of a rule taking place. In other words, the outcome is treated as a random variable. It is, therefore, natural to discuss the probability of the system being in a particular state at time t and to describe the evolution of these probabilities.

This leads to the *chemical master equation* (CME), i.e. a set of ODEs, with one ODE for each possible state of the system. At the time t , the k -th equation gives the probability of the system being in the k -th state. Note that the dimension of the ODEs is given by the number of possible states, not the number of possible molecules. Therefore, although it depends on the particular system, the number of equations is usually enormous and cannot be handled analytically or computationally.

A possible workaround is computing the CME indirectly, for example, using a stochastic simulation algorithm (SSA), also called Gillespie's algorithm [Gil76]. Instead of solving the set of ODEs to get a probability distribution over all possible states for each time t , we focus on computing samples from these distributions. In other words, we compute the realisation of the state vector $\{t, \mathbf{X}(t)\}$ such that the chance of a particular realisation being computed reflects the actual probability given by the CME.

We adopted this simulation approach for BCSL. Two possible approaches are available – with or without reaction network generation. The reaction network generating (or indirect approach) first computes all possible instantiations of rules. This approach allows leveraging the computationally-fast vector operations on the reaction level (details in [Section 6.2.1](#)), thus potentially speeding up the simulation of systems with rather small reaction network. Indeed, the cost of simulating the kinetics of a reaction network depends on the size of the network. Thus, if a network is large, the simulation cost can be expensive in terms of computation time or memory.

This problem is addressed using network-free simulators, where rules are used directly in the simulation algorithm instead of generating the whole reaction network. The method generalises an agent-based kinetic Monte Carlo method that has been shown to circumvent the combinatorial bottleneck in simulations [Yan+08]. The adoption of the algorithm is relatively straightforward, with an additional matching step, which needs to select particular molecules in the current state (generally, one rule can be applied in multiple ways). The advantage of network-free compared to reaction-based approaches is that during simulation, rules operate directly on molecular objects to produce exact stochastic results with a performance that scales independently of the reaction network size. However, for systems with

large molecules (e.g. a complex composed of many structure components with multiple feature domains), even a single simulation step can become computationally infeasible. An overview of the SSA algorithm adopted for BCSL is available in [algorithm 1](#).

Algorithm 1: Adopted Gillespie’s algorithm for BCSL. The input for this algorithm is the initial state vector \mathbf{X}_0 and a set of rules \mathcal{R} . In this case, with $\text{LHS}(r)$ (resp. $\text{RHS}(r)$), we denote a state vector corresponding to the left-hand side (resp. right-hand side) of a rule r .

```

1  $t \leftarrow 0, \mathbf{X} \leftarrow \mathbf{X}_0, \text{RXNS} \leftarrow \emptyset$ 
  /* find enabled rules and their possible matches */
2 for  $R \in \mathcal{R}$  do
3    $\text{RXNS} \leftarrow \text{RXNS} \cup \{r \mid R \models_r \mathbf{X}\}$ 
  /* evaluate rate functions and sort them in descending order */
4  $\text{RATES} \leftarrow \{f_r(\mathbf{X}) \mid r \in \text{RXNS}\}$ 
  /* select reaction  $J$  to fire with uniformly distributed random
     number  $\rho \in (0,1)$  */
5  $\text{ARG MIN}_J(\sum_{j=1}^J r_j > \rho \times \text{SUM}(\text{RATES}))$ 
  /* apply the reaction  $r_J$  */
6  $\mathbf{X} \leftarrow \mathbf{X} - \text{LHS}(r_J) + \text{RHS}(r_J)$ 
  /* sample the time of the next reaction event from exponential
     distribution with random number  $\rho'$  */
7  $t' \leftarrow -(1/\text{SUM}(\text{RATES})) \times \ln(\rho')$ 
  /* update time and go to line 2 or terminate */
8  $t \leftarrow t + t'$ 
```

While SSA is rather straightforward to implement, it can be impractically slow when rules occur frequently. A possible speed-up can be achieved by *lumping* the rules together and only updating the state vector after many rules were applied. This so-called τ -leaping approximation introduces error to the observed simulations. The error is, however, small enough as long as the state vector updates are relatively small.

By pushing the approximation even further, we obtain the *chemical Langevin equation* (CLE) [Giloo], represented by a set of stochastic differential equations. The number of equations is equal to the number of all possible molecules. The state vector becomes a continuous time, real-valued stochastic process. At each time t , $X_i(t)$ is also a real-valued random variable. To sum up the transition from CME to CLE, we have (1) relaxed from integers to real values in numbers of molecules, and (2) shifted from the probability distribution over the discrete set of states to a continuous probability distribution for each of the molecules (i.e. reduced the dimension of the system). Simulating the set of the stochastic differential equations, i.e. computing approximate trajectories $\{t, \mathbf{X}(t)\}$, is relatively cheap from

the computational point of view, especially compared to the original set of ODEs for every state.

A possible additional simplification is to ignore the fluctuations in the CLE and consider our model to be deterministic. This produces so-called *reaction rate equations*, that is, a set containing an ODE for each possible molecule. The state vector now describes concentrations of molecules at a certain time. Compared to the CME and CLE, simulating with this approach is an effortless task that can be handled by any stiff ODE solver.

```

#! rules
X()::rep ⇒ @ k1×[X()::rep]
Y()::rep ⇒ @ k2×[Y()::rep]
Z()::rep ⇒ @ k1×[Z()::rep]
⇒ X()::rep @ 5**4/(5**4 + [Z()::rep]**4)
⇒ Y()::rep @ 5**4/(5**4 + [X()::rep]**4)
⇒ Z()::rep @ 5**4/(5**4 + [Y()::rep]**4)

#! inits
1 X()::rep
1 Y()::rep

#! definitions
k1 = 0.05
k2 = 0.12

```

Figure 38: Example of generalised repressilator model in BCSL language. The simulation results for both stochastic and deterministic approaches are visualised in [Figure 53](#). The stochastic simulation is an average of two runs.

However, to construct the set of ODEs, the whole reaction network of the model is required. The reason is that an ODE for the particular molecule is composed of rate fluxes from the reactions it occurs in. More specifically, the rate of a reaction has a positive influence on the concentration of its products and a negative influence on its substrates. Therefore, in order to gather this information, we need first to enumerate the reactions.

To construct the ODEs, we need to specify for each reaction a *stoichiometric vector* $v_j = \text{RHS}(r_j) - \text{LHS}(r_j)$ where i th element represents the change of i th molecule after the reaction is applied. The effect of the reaction changes $\mathbf{X}(t)$ to $\mathbf{X}(t) + v_j$.

Then, assuming N possible molecules, and for every reaction r_j we have its stoichiometric vector v_j and rate function f_j , we arrive at a set of ODEs:

$$\frac{d\mathbf{X}(t)}{dt} = \sum_{j=1}^N v_j f_j(\mathbf{X}(t))$$

In this chemical kinetics, the state vector $\mathbf{X}(t)$ represents nonnegative real number concentrations of molecules at time t . Concentrations are usually measured in moles per litre. With the Avagadro's constant, giving the number of molecules in a mole, we can determine the number of molecules from the concentrations, assuming a fixed volume.

Finally, to be more precise, instead of directly using rate functions, a propensity function should be specified reflecting the *order* of the reaction. These underlying propensity functions can be derived from first principle physical arguments [Gil77], and therefore typically apply just to the law of mass action used as the rate functions. However, we loosen these restrictions and allow specifying arbitrary (yet still rational) rate functions, although possibly disrupting the low-level physical interpretation of the rate laws.

In Figure 38, we provide a generalised repressilator model [ELoo; Mül+06] as an example of a model suitable for both stochastic and deterministic types of simulation. Visualised results of both simulations are then available in Figure 53.

5.2 MODEL CHECKING

Model checking allows analysing the behaviour of a model with respect to temporal properties. We assume two approaches corresponding to the two fragments of BCSL. We investigate branching-time logics, namely computational tree logic CTL [CES86] for the qualitative fragment and *probabilistic* computational tree logic PCTL [HJ94] for the quantitative fragment. For both of these approaches, we focus on the properties of individual logics, their evaluation, and particular data structures used for model checking.

First, let us start with the qualitative fragment. For that, we need to define the CTL logic and how it can be interpreted over the labelled transition system of the BCSL model.

We define a Kripke structure [Kri63] $\mathcal{K} = (S, s_0, R, l)$ where S is a finite set of states, $s_0 \in S$ is an initial state, $R \subseteq S \times S$ is a transition relation such that it is left-total, i.e., $\forall s \in S. \exists s' \in S$ such that $(s, s') \in R$, and l is a labelling function $l : S \rightarrow 2^{AP}$, where AP is given set of atomic propositions.

Assuming we have an LTS(\mathcal{B}) = (S', T, L) of a qualitative BCSL model $\mathcal{B} \in \mathbb{B}$, we can construct the corresponding Kripke structure \mathcal{K} . The set of states S is given by a bijective function $h : S' \rightarrow S$ assigning to each state from S' a state from S . The initial state $s_0 = h(s)$ is given by the state $s \in S'$ corresponding to the initial state of the model \mathcal{B} .

The transition relation $R = \{(h(s), h(s')) \mid (s, r, s') \in T\}$ corresponds to the transition relation T of the LTS, for any rule r .

Finally, we need to tackle the labelling function l . For that, we need to construct the set of atomic propositions AP . This can be created using the information present in states S , which contain details about copies of individual molecules present in the state. Then the particular atomic propositions can be created generally, and their evaluation in the respective logic needs to be adjusted.

For the set of atomic propositions AP , we can consider the expressions of type $[X] \bowtie n$ where $X \in \mathbb{X}$ is a complex agent, $n \in \mathbb{N}$ is a threshold, and $\bowtie \in \{\leq, <, >, \geq\}$. These formulae allow reasoning about patterns which is very natural in the rule-based setting. Then, the evaluation of such an expression in a state $s \in S$ is given by:

$$s \models [X] \bowtie n \Leftrightarrow \sum_{J \in \Gamma} s(J(X)) \bowtie n$$

i.e., we sum all possible instantiations of the agent X and compare it with the given threshold. Alternatively, a more straightforward approach is to use only atomic propositions provided by the property we want to evaluate; then, the labelling function l checks whether they hold in each individual state and assigns them to such state.

With a proper data structure with all the necessary details suitable for model checking, we can proceed with the definition of CTL logic and its evaluation on this structure. The regular language of well-formed formulae for CTL is given by the following (context-free) grammar:

$$\begin{aligned} \phi &::= \text{True} \mid \alpha \in AP \mid \neg\phi \mid (\phi \wedge \phi) \mid A\psi \mid E\psi \\ \psi &::= X\phi \mid (\phi \mathbf{U} \phi) \end{aligned}$$

where ϕ is also referred to as a state formula and ψ as a path formula. Additionally, there are additional operators commonly used in CTL formulae that can be inferred from the existing ones:

- $EF\phi \equiv E[\text{True} \mathbf{U} \phi]$
- $AF\phi \equiv A[\text{True} \mathbf{U} \phi]$
- $EG\phi \equiv \neg AF\neg\phi$
- $EG\phi \equiv \neg EF\neg\phi$

We evaluate a CTL formula in a Kripke structure \mathcal{K} . For the two types of formulae, we also have semantics defined separately for both of them. A path π in the Kripke structure is a sequence of states $s_0 s_1 s_2 \dots$ with $s_i \in S$ and $(s_i, s_{i+1}) \in R$ for all $i \geq 0$. By π_i , we denote the state s on position i and by $\pi[i]$ the subsequence of path π starting by the state at position i . Then the semantics of the state formulae is given by relation $(\mathcal{K}, s) \models \phi$ defined recursively:

- $(\mathcal{K}, s) \models \text{True}$
- $(\mathcal{K}, s) \models a \Leftrightarrow a \in l(s)$
- $(\mathcal{K}, s) \models \neg\phi \Leftrightarrow (\mathcal{K}, s) \not\models \phi$
- $(\mathcal{K}, s) \models (\phi_1 \wedge \phi_2) \Leftrightarrow (\mathcal{K}, s) \models \phi_1 \wedge (\mathcal{K}, s) \models \phi_2$
- $(\mathcal{K}, s) \models \mathbf{E}\psi \Leftrightarrow \exists \pi \text{ with } \pi_0 = s \text{ s.t. } (\mathcal{K}, \pi) \models \psi$
- $(\mathcal{K}, s) \models \mathbf{A}\psi \Leftrightarrow \forall \pi \text{ with } \pi_0 = s \text{ s.t. } (\mathcal{K}, \pi) \models \psi$

Finally, the semantics of the *path* formulae is given by $(\mathcal{K}, \pi) \models \phi$ also defined recursively:

- $(\mathcal{K}, \pi) \models \mathbf{X}\phi \Leftrightarrow (\mathcal{K}, \pi[1]) \models \phi$
- $(\mathcal{K}, \pi) \models (\phi_1 \mathbf{U} \phi_2) \Leftrightarrow \exists i \geq 0. (\mathcal{K}, \pi[i]) \models \phi_2 \wedge \forall j < i. (\mathcal{K}, \pi[j]) \models \phi_1$

The model checking of a given CTL formula proceeds (using an explicit standard algorithm [CJ+18]) by induction in stages, where in each stage, it is determined whether a subformula is true in particular states, starting with “smallest” subformulae, and working up to the given formula. Based on these evaluation steps, it is decided whether the CTL formula holds in the initial state s_0 .

As a simple example, let us consider CTL formula (a) from Figure 39, stating that eventually in the future (EF), we export the protein P out of the cell while the serine residue is phosphorylated. By evaluating the CTL model checking on the Kripke structure created from LTS in Figure 19, we find that the formula is valid.

- a) $\text{EF}(P(\text{S}\{a\}, \text{T}\{i\})::\text{out} > 0)$
- b) $P \leq 0.15 (F(P(\text{S}\{a\}, \text{T}\{i\})::\text{out} > 0))$
- c) $P = ? (F(P(\text{S}\{a\}, \text{T}\{i\})::\text{out} > 0))$

Figure 39: Examples of CTL and PCTL formulae used for model checking.

Next, let us proceed with the quantitative fragment of BCSL. The LTS of a model $\mathcal{B} \in \mathbb{B}_q$ contains quantitative information on the edges, that is, evaluated rate function to a rational number. Since we want to perform model checking using PCTL, we first need to create a data structure containing the probabilities of transitions instead of rates. For this purpose, we use *Discrete Time Markov Chain* (DTMC).

DTMC is a tuple (S, R, l, s_0) where S is the set of states, $R : S \times S \rightarrow [0, 1]$ is the *transition probability matrix*, where for all $s \in S$ we require that $\sum_{s' \in S} R(s, s') = 1$, $l : S \rightarrow 2^{\text{AP}}$ is a *labelling function* which gives the atomic propositions that are true in a state, and $s_0 \in S$ is the *initial state*.

To construct such a DTMC from $\text{LTS}(\mathcal{B})$, we can use the same procedure to create the set of states S , the labelling function l , and the initial state s_0 , as we used in the case of Kripke structure in the qualitative approach.

For the transition probability matrix R , an entry $R(s, s')$ gives the probability of making a transition from state s to state s' . To infer this probability from the $\text{LTS}(\mathcal{B}) = (S', T, L)$, we need to normalise the rates for every state $s \in S$. The procedure needs to sum rates of all outgoing edges from the state s . For that, we define $\text{rate}(s, s')$ equal to n if $(s, (r, n), s') \in T$ for some rule r and equal to zero otherwise. Then, we can calculate the normalisation as follows:

$$\text{norm} = \sum_{s' \in S} \text{rate}(s, s')$$

The entry $R(s, s')$ of the matrix is given as a ratio of the respective rate divided by the value norm . This will, indeed, ensure that the sum of all outgoing probabilities is equal to one.

The probability of following a finite path $s_0 s_1 \dots s_n$ can be computed as $R(s_0, s_1) \cdot R(s_1, s_2) \cdot \dots \cdot R(s_{n-1}, s_n)$. These probabilities for finite paths give rise to a unique probability measure \Pr_{s_0} on the set Path_{s_0} of infinite paths starting in state s_0 defined on the sets of paths having a finite common prefix, such that:

$$\Pr(\{\pi \mid \pi = s_0 s_1 \dots s_n \cdot \pi'\}) = R(s_0, s_1) \cdot R(s_1, s_2) \cdot \dots \cdot R(s_{n-1}, s_n).$$

An example of DTMC for a BCSL model is available in [Figure 40](#).

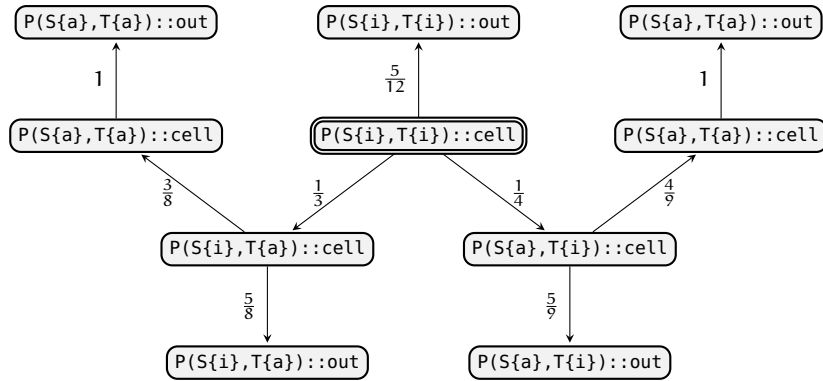


Figure 40: DTMC of the model from [Figure 15](#). The structure has the same shape as the transition system in [Figure 17](#), but the rate values on edges are normalised to probabilities.

This concludes the construction of DTMC that can be used for PCTL model checking. Let us proceed with the logic PCTL [\[HJ94\]](#), which is a probabilistic variant of CTL where the existential and the universal quantification over paths in CTL is replaced with a *probabilistic operator* $\Pi_{\bowtie \rho}(\cdot)$, where $\bowtie \in \{\leq, <, >, \geq\}$ and $\rho \in [0, 1]$ is the *probability*

threshold, that can be applied to a path formula. The formal syntax of PCTL formulae is given by the following grammar:

$$\begin{aligned}\phi &::= \text{True} \mid a \in \text{AP} \mid (\phi \wedge \phi) \mid \neg\phi \mid \Pi_{\bowtie\rho}(\psi) \\ \psi &::= \mathbf{X}\phi \mid (\phi \mathbf{U} \phi)\end{aligned}$$

The semantics of PCTL is the same as that of CTL [CES86] for the fragment where they both coincide. The semantics of the probabilistic operator is:

$$s \models \Pi_{\bowtie\rho}(\psi) \Leftrightarrow \mathbf{Pr}_s(\{\pi \in \text{Path}_s \mid \pi \models \psi\}) \bowtie \rho$$

meaning that the probability measure of the set of paths satisfying ψ is calculated and compared to the threshold ρ , yielding true or false.

The standard qualitative model checking algorithm proceeds in the same way as for CTL, by induction on ϕ . In addition to that, let us discuss a symbolic approach proposed in [Dawo4]. It is based on the derivation of a finite state automaton (FSA) $\mathcal{A} = (S, \Sigma, \delta, S_f)$ from a given DTMC. S is the same set of states as in the DTMC, the alphabet Σ consists of the strictly positive entries of the probability matrix, the set of final states S_f and the transition function δ depend on the path formula under consideration.

The regular language $\mathcal{L}(\mathcal{A}, s)$ recognized by \mathcal{A} with an initial state $s \in S$, corresponds to the (possibly infinite) set of finite paths from s to some final state in S_f , following only transitions allowed by δ .

A regular expression RE over an alphabet Σ is computed using the state-elimination algorithm [Hop08]. The evaluation $\text{val}(\text{RE})$ of the regular expression can be done by replacing union by addition, concatenation by multiplication, and star by the limit of a geometric series (for the formal definition, see [Dawo4]).

The evaluation of a regular expression RE computed for a language $\mathcal{L}(\mathcal{A}, s)$ is the probability measure in s of the set of paths with prefixes in $\mathcal{L}(\mathcal{A}, s)$:

$$\text{val}(\text{RE}) = \mathbf{Pr}_s \left(\left\{ \pi \in \text{Path}_s \mid \begin{array}{l} \exists k \geq 0. \pi[k] \in S_f \wedge \\ \forall l < k, \exists a \in \Sigma. \pi[l+1] \in \delta(\pi[l], a) \end{array} \right\} \right)$$

The model checking problem can then be solved for a state s by evaluating a regular expression RE equivalent to the language recognized by the automaton with the initial state s , that is $s \models \Pi_{\bowtie\rho}(\psi)$ iff $\text{val}(\text{RE}) \bowtie \rho$.

Let us continue with the example provided for CTL model checking and shift it to the probabilistic case. In particular, let us use PCTL formula (b) from Figure 39, stating that the probability of reaching the protein in the particular form is less or equal to 0.15. By evaluating the PCTL model checking on the DTMC available in Figure 40, we find that the formula is valid.

Additionally, in PCTL, we can also directly specify properties which evaluate to a numerical value. This is achieved by replacing the probability bound from the Π operator with ‘=?’. This is only allowed when the Π in question is the outermost operator of the property. The evaluation is then given as $\Pi_{=?}(\psi) = \mathbf{Pr}_s(\{\pi \in \text{Path}_s \mid \pi \models \psi\})$ allowing it to be computed using the symbolic approach as $\Pi_{=?}(\psi) = \text{val}(\text{RE})$. This enables computing the probability that a property holds instead of just comparing it to a threshold and obtaining a boolean result.

Adopting this approach to the previous example, we can actually compute the threshold where the satisfiability of the inner reachability property shifts from valid to violated. Such a PCTL formula is shown in [Figure 39c](#), and running PCTL model checking with the DTMC from [Figure 40](#) gives the value 0.138.

5.3 PARAMETER SYNTHESIS

The previous section explained how to analyse the behaviour of a BCSL model with respect to PCTL property in a non-parametrised setting. This section provides insight into PCTL analysis of parametrised model $\mathcal{B} \in \mathbb{B}_p$, also called *parameter synthesis*. The goal is to find parameter values such that the given PCTL property is satisfied.

We begin with a suitable data structure. We use *Parametric Markov chain* (pMC). pMC is a tuple (S, R, l, P, s_0) where S is a finite set of states, P is a finite set of parameters, $R : S \times S \rightarrow \mathbb{F}_p$ is the *parametric transition probability matrix* where \mathbb{F}_p is a set of rational functions over parameters P , labelling function $l : S \rightarrow 2^{\text{AP}}$ gives the atomic propositions that are true in a state, and $s_0 \in S$ is the *initial state*.

For the construction of such pMC for given $\text{LTS}(\mathcal{B})$ of parametrised model \mathcal{B} , we can use the same procedure to obtain the set of states S , the labelling function l , and the initial state s_0 as we used in the case of DTMC and quantitative approach. Similarly, the parametric transition probability matrix R is constructed in the same way, but instead of rational numbers, we operate symbolically with rate expressions from \mathbb{F}_p over parameters P . For each transition, we get a probabilistic function over parameters. By assigning particular parameter values, we obtain the probability of the transition. An example of a pMC is available in [Figure 41](#).

Let us provide more details on the set of parameters P . We assume a domain $\mathcal{D}(p)$ for each parameter $p \in P$. Then the *parameter space* \mathbf{V} induced by the set of parameters P is defined as the Cartesian product of individual parameter domains $\mathbf{V} = \times_{p \in P} \mathcal{D}(p)$. A *parametrisation* $v \in \mathbf{V}$ is a $|P|$ -tuple holding a single value for each parameter, i.e. $v = (p_1, \dots, p_{|P|})$, assuming an arbitrary ordering on parameters.

For a pMC C , the set of DMTCs induced by the parameter space \mathbf{V} is defined as $\mathcal{C} = \{C_v \mid v \in \mathbf{V}\}$. For each C_v , all parameters in the probability matrix are instantiated to respective components of

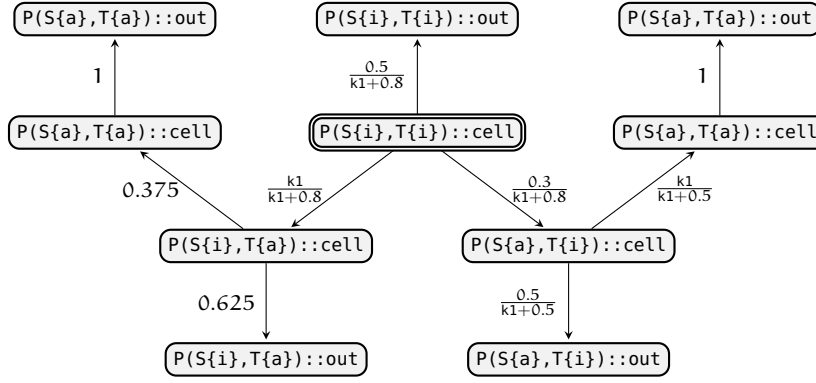


Figure 41: pMC of the model from Figure 15 with unknown parameter $k1$. The structure has the same shape as the transition system in Figure 17, but the values and rate expressions over parameters on edges are normalised.

v . A DTMC C_v is *well-defined* iff $T(s, s') \in [0, 1]$ for all $s, s' \in S$ and $\sum_{s' \in S} T(s, s') = 1$ for all $s \in S$. For every pMC C , we assume the set \mathcal{C} contains only well-defined DTMCs.

Given a parametrised BCSL model $\mathcal{B} \in \mathbb{B}_p$, parameter space \mathbf{V} , and a PCTL formula ϕ , the *problem of parameter synthesis* is to compute a partitioning of parameter space into three disjoint subsets: **TRUE** – the model satisfies the property, **FALSE** – the model does *not* satisfy the property, and **UNKNOWN** – the result is not known. The goal is to minimise the size of the UNKNOWN set.

We solve this problem in three steps: (1) we construct LTS for the given model, (2) we derive a pMC from the LTS by computing a parametric transition probability matrix as a normalisation of the label for all outgoing edges for every state, and (3) we apply a method introduced in [Daw04] and elaborated in [HHZ11], which is very similar to the model checking of DTMC described in the previous section.

The Finite State Automaton for a pMC and a path formula is derived as in the non-parametric case. The regular expression is also evaluated recursively. Operators of union, concatenation, and star on regular expressions, are replaced by addition, multiplication, and inversion for *rational functions*, respectively. Thus, by evaluating the corresponding regular expression, we obtain an algebraic expression of the probability measure of the sets of paths satisfying a path formula as a rational function of parametrisations. We can use the result to check whether the system satisfies a formula for different values of the parameters. Such a formula can also be used to compute robustness measure (see Section 5.4 for details).

This method is applicable to formulae without nested probabilistic operators only, but this does not represent a strong restriction in practice because such formulae are usually not needed to specify the properties of interest.

The computed rational function can be used in parameter space exploration. An SMT solver (such as Z3 [MBo8]) is used to determine whether there exists a parametrisation inside the candidate region of the parameter space whose corresponding instantiated DTMC exceeds a given threshold on the probability.

The general approach is to maintain a set UNKNOWN of regions for which the result is still unknown. Initially, it is represented as the whole parameter space V . Then, it takes a region out of this set and tries to decide its value. The answer can be definite, i.e. either the region satisfies the formula ϕ and is added to set TRUE or it does not satisfy the formula ϕ is added to set FALSE, or the answer is uncertain, and the region is split into smaller subregions. This can be recursively executed until the required precision is met (e.g., coverage of the decided area, a boundary in recursion depth).

The remaining issue is the presence of patterns allowed in atomic propositions of the PCTL property. However, this can be easily solved since a pattern is basically just a compact representation of all possible instantiated agents (resp. multisets), and it can be expressed as a sum of these agents.

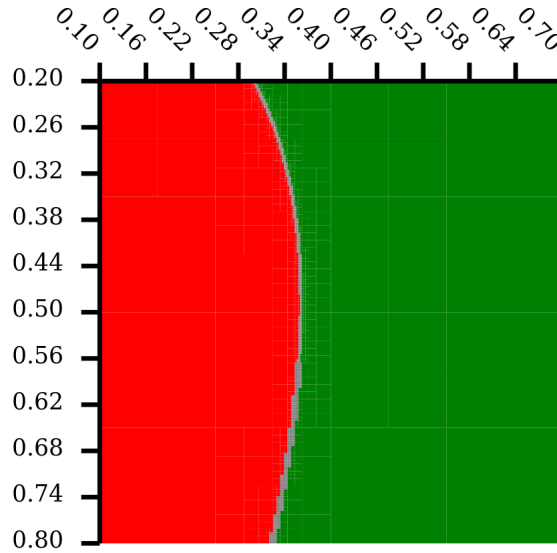


Figure 42: Partitioning of parameter space computed by parameter synthesis of pMC from Figure 41 w.r.t. PCTL property (b) from Figure 39. Values of parameter k_1 are displayed on the x-axis while parameter k_2 on the y-axis. The property is satisfied in the green regions and violated in the red regions. Grey regions are UNKNOWN.

For example, we compute the parameter synthesis for pMC from Figure 41 and PCTL property (b) from Figure 39. For parameters we assume ranges $k_1 \in [0.1, 0.7]$ and $k_2 \in [0.2, 0.8]$ and we obtain partitioning of the parameter space into regions shown in Figure 42. For

a demonstration of parameter synthesis on real biological examples, see [Section 7.2](#).

5.4 ROBUSTNESS ANALYSIS

Instead of mapping the parameter space and determining regions where a particular property is satisfied, it can be helpful to establish a measure of global satisfiability. This measure states how well-preserved the property is with respect to the given parameter space. Commonly, this measure is called the *global robustness*.

The problem of global robustness [Kit07] of a BCSL model \mathcal{B} is defined as:

$$\mathbf{G}_{\phi, \mathbf{V}}^{\mathcal{B}} = \int_{\mathbf{V}} \rho(v) \mathbf{L}_{\phi}^{\mathcal{B}}(v) dv$$

where ϕ is the property of the system under scrutiny, \mathbf{V} is the parameter space, $\rho(v)$ is the probability of the parametrisation $v \in \mathbf{V}$, and *local robustness* $\mathbf{L}_{\phi}^{\mathcal{B}}(v)$ is a measure stating how much the property ϕ is preserved in *parameterisation* v . The local robustness returns for each $v \in \mathbf{V}$ the quantitative model checking result for the respective DTMC (built for the parameterisation v) and the given property ϕ .

We solve this problem for a given parametrised BCSL model $\mathcal{B} \in \mathbb{B}_p$ and a PCTL property ϕ (with the outermost operator $\Pi_{=?}$). We construct pMC from the model followed by the algorithm from [Daw04] to compute the rational function f , determining the probability of satisfying property ϕ for a given parametrisation. Therefore, function f can be directly used for the evaluation of the local robustness.

We consider the parameter space \mathbf{V} as the set of all parameterisations. Since each parameterisation $v \in \mathbf{V}$ has uniform probability, computing $\rho(v)$ is straightforward – it is inversely proportional to the volume of the entire parameter space. Considering all the assumptions, the robustness for the BCSL model \mathcal{B} and a property ϕ is computed as:

$$\mathbf{G}_{\phi, \mathbf{V}}^{\mathcal{B}} = \int_{\mathbf{V}} \frac{1}{|\mathbf{V}|} f(v) dv$$

We show a simple example of robustness analysis on the previous example used in parameter synthesis. For that, we need the parameter space \mathbf{V} and probability function f . The parameter space is given by parameter ranges $k1 \in [0.1, 0.7]$ and $k2 \in [0.2, 0.8]$. The rational function f can be computed as a byproduct of parameter synthesis of PCTL property (c) from [Figure 39](#), when no parameter ranges are specified. This yields the function $f = \frac{3 \times k2}{(k2 + k1) \times (10 \times k2 + 10 \times k1 + 3)}$. With all necessary inputs, the measure of global robustness can be evaluated to value 0.148.

5.5 STATIC ANALYSIS

The previous analysis techniques, to some extent, execute the model and focus on its dynamical behaviour. However, BCSL offers interesting capabilities to study models on the syntactic level without executing them. This is enabled primarily for employed abstraction and high-level structure of agents.

The static analysis techniques we present in this chapter are based on the introduced *compatibility* relation \triangleleft (Section 3.4), which formulates suitable properties for each type of agent. Since complex agents \mathbb{X} encapsulate other agents, we will focus on them.

Let $x_1, x_2 \in \mathbb{X}$ be two arbitrary complex agents. The compatibility relation induces *partial ordering* of agents x_1 and x_2 , written $x_1 \leq x_2$, iff $x_1 \triangleleft x_2$. The universe of complex agents \mathbb{X} with partial order \leq is a partially ordered set \mathbb{X}_{\leq} . However, the partial order of the entire universe of complex agents is not very useful since most of the agents cannot be compared by the compatibility operator. We are interested in particular subsets where every two complex agents can be either compared directly or there exists an agent compatible with both of them.

We therefore formulate the term *compatible set*, that is a finite set $\mathcal{X} \subseteq \mathbb{X}$ such that the following two conditions hold:

1. $\forall x_1, x_2 \in \mathcal{X} \exists x' \in \mathcal{X} : x_1 \triangleleft x' \wedge x_2 \triangleleft x'$,
2. and for each finite set $\mathcal{X}' \subseteq \mathbb{X}$ with $\mathcal{X} \cap \mathcal{X}' = \emptyset$ holds that

$$\forall x \in \mathcal{X} \forall x' \in \mathcal{X}' : \neg(x \triangleleft x' \vee x' \triangleleft x).$$

Informally, the compatible set is *complete* in the sense that all possible compatible agents are present, and it is *disjoint* with all other compatible sets. Since it is its subset, the compatible set \mathcal{X} inherits the partial order of \mathbb{X}_{\leq} . Therefore, a compatible set \mathcal{X} contains partially ordered complex agents with identical sequences in terms of agent names. An example of a compatible set is given in Figure 43.

Lemma 3. *In every compatible set \mathcal{X} , there always exists a global supremum $\sup(\mathcal{X})$.*

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

Lemma 4. *For every complex agent x there exists exactly one compatible set $\mathcal{X} \subseteq \mathbb{X}$ such that $x \in \mathcal{X}$.*

Proof. Let us assume a complex agent x belongs to two compatible sets, namely $x \in \mathcal{X}_1, \mathcal{X}_2$. From the first condition in the definition

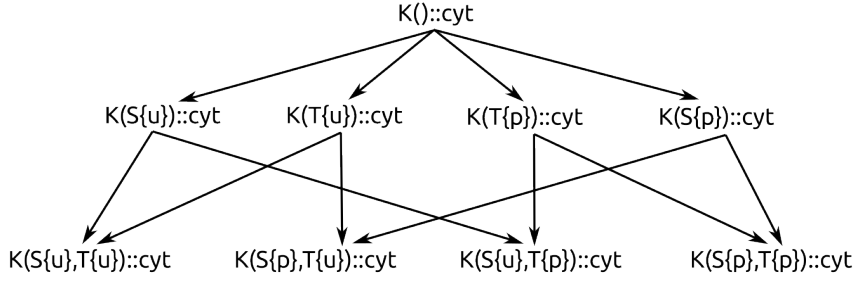


Figure 43: An example of a compatible set \mathcal{X} . The set is formed by a complex in the *cyt* compartment, which has only one structure agent *K* in its sequence. The structure agent *K* has allowed atomic agents *T* and *S* in its composition. These two atomic agents might occur in two states – *u* and *p*. The set is complete – there are all relevant agents bounded by the compatibility operator.

of the compatible set follows that there exists a $x_1 \in \mathcal{X}_1$ such that $x \triangleleft x_1$.

Next, the second condition in the definition of compatible set claims that no complex agent from \mathcal{X}_1 and no complex agent from \mathcal{X}_2 can be compatible. Namely, $x_1 \in \mathcal{X}_1$ cannot be compatible with $x \in \mathcal{X}_2$. However, x and x_1 are compatible ($x \triangleleft x_1$). It follows $x \notin \mathcal{X}_2$, which is a contradiction. \square

We use these two claims to investigate the compatible sets even further. Let $\mathcal{X} \subseteq \mathbb{X}$ be a compatible set and $x \in \mathcal{X}$ a complex agent. A set $\bar{\mathcal{X}} \subseteq \mathcal{X}$ is called *compatible subset* of \mathcal{X} w.r.t. x if the following conditions hold:

1. $\forall x' \in \bar{\mathcal{X}} : x' \triangleleft x \wedge \Delta x'$,
2. $\nexists x'' \in \mathcal{X} \setminus \bar{\mathcal{X}} : x'' \triangleleft x \wedge \Delta x''$.

The compatible subset $\bar{\mathcal{X}}$ formally defines all fully specified agents from the compatible set \mathcal{X} which are compatible with a given element x of the set \mathcal{X} (i.e. there are no compatible agents with them in the set). Note that there exists just one compatible subset for any complex agent x . The reason follows from [Lemma 4](#) and the definition of the compatible subset.

In practice, compatible (sub)sets can be used for finding non-trivial relationships between the rules ([Section 5.5.1](#)) and for static analysis on the level of complexes ([Section 5.5.2](#)).

5.5.1 Rule redundancy elimination

Biological models can grow to large sizes with increasing complexity and decreasing readability. In such cases, it can be easily achieved that there are some redundant rules in a model. These rules do not

cause any semantic difference, only increase the size of the model. We provide a static method to detect such rules and possibly delete them from the model. Please note the redundancy is relevant only in the qualitative context. In the quantitative context, the same rules with different kinetics might have their relevance, yet it is still useful to detect such potential redundancies.

Let $\mathcal{B}_1 = (\mathcal{R} \cup \{R\}, \sigma_a, \sigma_s, M_0)$ and $\mathcal{B}_2 = (\mathcal{R}, \sigma_a, \sigma_s, M_0)$ be BCSL models where R is a rule such that $R \notin \mathcal{R}$. The rule R is *redundant* if $\text{LTS}(\mathcal{B}_1) \sim \text{LTS}(\mathcal{B}_2)$ (where \sim means the graphs of both LTSs are isomorphic, ignoring the values of the labels). The redundant rule R does not add any semantic information to the model. It generally means the LTSs produced from the models with and without the rule are equal.

Theorem 13. *Let $R = (\chi, \omega, \iota, \varphi, \psi)$ and $R' = (\chi', \omega', \iota', \varphi', \psi')$ be two rules such that $|\chi| = |\chi'| = n$ for some $n \in \mathbb{N}$. The rule R' is redundant if $\forall i \in [1, n] : \chi'_i \triangleleft \chi_i$.*

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

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

Applied generally on the entire rule, the produced set of reactions (using the matching) from the redundant rule is actually a subset of reactions produced from the non-redundant rule. \square

In the proof, we used compatible sets of agents and the fact that we can generate reactions from the rules while we are actually enumerating all agents from the compatible set, which are *compatible with* the original agent in the rule. Let us demonstrate this on an example of two rules:

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

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

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

while the rule (2) produces set of four reactions:

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

which is a subset of the previous one. That means the rule (2) is redundant.

5.5.2 Context-based reduction

Simplification of models is a well-established technique for providing some less accurate but computationally feasible insight into the behaviour of models. For this purpose, we introduce context-based reduction, which eliminates all features and their modifications from the model, leaving only complex formations and dissociations. Such a reduction still preserves some properties while making the analysis of the model simpler. This is particularly the case of dynamic analysis, where a minor change in the model specification can dramatically affect the behaviour. We first define a function that simplifies rules and then define the notion of a reduced model and show what kind of information it preserves.

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

Next, we adapt the notion of the reduced rule to a whole model. Let $\mathcal{B} = (\mathcal{R}, \sigma_a, \sigma_s, M_0)$ be a BCSL model (referenced as *initial* for clarification). We define *reduced model* $\tilde{\mathcal{B}} = (\tilde{\mathcal{R}}, \sigma_a, \sigma_s, I)$ such that the following conditions hold:

1. $\forall R \in \mathcal{R}. \theta(R) \in \tilde{\mathcal{R}}$ and $\forall R \in \tilde{\mathcal{R}}. \exists R' \in \theta(R). R' \in \mathcal{R}$
2. for every complex agent $X \in M_0$ holds that $\text{sup}(\mathcal{X}) \in I$ where \mathcal{X} is a compatible set with $X \in \mathcal{X}$, and every complex agent in the reduced model is the image by $\text{sup}(\mathcal{X})$ of a complex agent of the initial model.

Reduced model $\tilde{\mathcal{B}}$ is created from the given BCSL model \mathcal{B} by reducing the context of complexes in the rules to the maximum level. This is achieved by taking supremum from compatible set \mathcal{X} . This procedure can produce some not well-formed rules – such rules are omitted (see Figure 44 for an example). Consequently, only rules creating/destroying agents and complex formation/dissociation should remain. Since we are reducing context, the number of rules in the resulting model is equal to or smaller than the number of rules in the initial model.

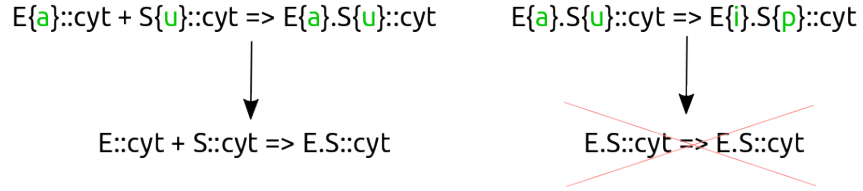


Figure 44: Examples of rule reductions. (left) A rule of complex formation is reduced to a version where none of the states is specified. (right) A rule of state change inside of a complex is reduced to a rule which is not well-formed. It violates the first condition of well-formed rules – an agent has to change during the rule application. Therefore it is removed from the reduced model.

To formulate properties about LTS of reduced models, we first need to extend the notion of compatibility to states. Let \mathcal{B} be a BCSL model and s_1, s_2 two states from its $\text{LTS}(\mathcal{B})$. The state s_1 is *compatible with* state s_2 , written $s_1 \triangleleft s_2$, iff there exists a bijective function $f : s_1 \rightarrow s_2$ such that $\forall X \in s_1 : \sup(X) = f(X)$ where $\mathcal{X} \subseteq \mathbb{X}$ is a compatible set w.r.t. \mathcal{X} .

This allows us to formulate an over-approximation relation on transition systems. Let $\text{LTS}(\mathcal{B}), \text{LTS}(\mathcal{B}')$ be labelled transition systems of some BCSL models $\mathcal{B}, \mathcal{B}'$. The $\text{LTS}(\mathcal{B}')$ is an *over-approximation* of $\text{LTS}(\mathcal{B})$ if for every path $\dots s'_1 s'_2 s'_3 \dots s'_n \dots$ in $\text{LTS}(\mathcal{B}')$ there exists a path $\dots s_1 s_2 s_3 \dots s_m \dots$ in $\text{LTS}(\mathcal{B})$ such that this property holds:

$$\forall s'_i, s'_{i+1} \exists s_k, s_l : (l > k \wedge s_k \triangleleft s'_i \wedge s_l \triangleleft s'_{i+1}).$$

Next, we show that a reduced model $\tilde{\mathcal{B}}$ is an over-approximation of a BCSL model \mathcal{B} in the context of their LTSs. These findings can be used for some types of analyses which avoid the combinatorial explosion of the initial model \mathcal{B} .

Theorem 14. Let x be a complex agent, \mathcal{X} be a compatible set w.r.t. \mathcal{X} , \mathcal{B} be a given BCSL model, and $\tilde{\mathcal{B}}$ be an appropriate reduced model of model \mathcal{B} . If supremum $\sup(\mathcal{X})$ is non-reachable in $\text{LTS}(\tilde{\mathcal{B}})$, then agent x is also non-reachable in the $\text{LTS}(\mathcal{B})$.

Proof. Let us assume a complex agent $\sup(\mathcal{X})$ is non-reachable in $\text{LTS}(\tilde{\mathcal{B}})$, but $x \in \mathcal{X}$ is reachable in $\text{LTS}(\mathcal{B})$. Generally, there is a path

formed from rules in the $LTS(\mathcal{B})$ such that we transform complex agents from initial agents to desired complex agent X . When we move to context of $LTS(\hat{\mathcal{B}})$, there is no such path for $\sup(X)$.

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

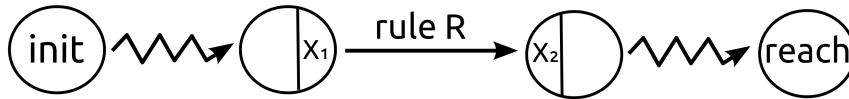
When we are checking whether an agent is reachable in $LTS(\mathcal{B})$ for given model \mathcal{B} , we might first check whether the respective abstract agent (the supremum) is reachable in $LTS(\hat{\mathcal{B}})$ of the reduced model $\hat{\mathcal{B}}$. If this holds, then we are still not certain about the reachability of the agent in its initial form. This has to be checked in $LTS(\mathcal{B})$. However, [Theorem 14](#) states that agent which is not reachable in $LTS(\hat{\mathcal{B}})$ is also not reachable in $LTS(\mathcal{B})$. The usage of the theorem is demonstrated in [Section 7.3](#).

5.5.3 Static non-reachability analysis

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

Theorem 15. *Let \mathcal{B} be a BCSL model and \mathcal{R} its set of rules. Let X be a complex agent. The complex agent X is non-reachable w.r.t. set of rules \mathcal{R} if $\forall R \in \mathcal{R} \forall i > \iota + 1 : \neg(\chi_i \triangleleft X)$, where $R = (\chi, \omega, \iota, \varphi, \psi)$.*

Proof. Let us assume we have a path of states constructed by applying corresponding rules from \mathcal{R} where X is reachable. At some point on the path, we inevitably have to create a complex agent $X_2 \triangleleft X$ from a complex agent X_1 applying a rule R .



It requires there has to be a complex agent X'_2 in the rule, which is compatible with the complex agent X_2 . If there is no such agent, the agent X is non-reachable. \square

Compared to dynamic reachability analysis, [Theorem 15](#) completely avoids any combinatorial explosion and gives an answer only by checking the structural properties of rules. The usage of the theorem is demonstrated in [Section 7.3](#).

5.6 SUMMARY

This chapter establishes an analysis base for BCSL models. Since there are multiple variants of the models, we also assume different analysis techniques for them.

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

Therefore we also established model checking and parameter synthesis techniques for both qualitative and quantitative model variants. Qualitative models are analysed w.r.t. CTL properties, while the quantitative models introduce probabilities that can be expressed using PCTL properties. For parametrised models, we have shown how the unknown parameters can be synthesised, obtaining partitioning of the parameter space to the satisfying and violating regions, as well as how to measure the robustness of the model w.r.t. PCTL property, providing an insight into how the property is preserved across the assumed parameter space.

Finally, since the behaviour of BCSL models can be rather extensive and, therefore, many analysis techniques computationally infeasible, we also provided several static methods that can be used to either reduce the model or directly compute useful properties of the model, such as reveal some non-reachability results.

*In mathematics as in other fields, to
find one self lost in wonder at some
manifestation is frequently the half
of a new discovery.*

PETER GUSTAV LEJEUNE DIRICHLET

WE developed a software tool eBCSgen to support the usability of BCSL. The development of the tool started with the formulation of the language. Originally, its sole purpose was to generate the state transition system for a model (hence the name BCSgen = generator). The first version was developed as a desktop GUI tool. Later we switched to the web-based approach and the tool gained its current name eBCSgen (to distinguish the desktop version).

In its current version (v2.1.0 at the time of writing this thesis), the tool is implemented as a command-line Python package distributed using conda package manager [Ana] in the bioconda channel [Grü+18]. To improve its usability outside of computer science, it is wrapped into a series of tools for Galaxy [Afg+18], a web-based scientific analysis platform used to analyse large datasets. With its three primary features – accessibility, reproducibility, and communication – it is a very convenient and practical alternative to a separate GUI development. It allows connecting individual tools into comprehensive pipelines and the distribution of results among other scientists by sharing computation histories and datasets.

In this chapter, we describe eBCSgen on a detailed level, including its core components, implementation of analysis methods, Galaxy interface, essential visualisation methods, and documentation.

6.1 OVERVIEW

In this section, we comment on the content of eBCSgen source codes, available in the GitHub repository¹. The top-level structure is displayed in Figure 45.

The first part contains a configuration file for the conda environment, with more information in Section 6.4.1. Generally, eBCSgen is distributed via conda as a package after every release. However, for local development and testing, it is useful to easily create an environ-

¹ <https://github.com/sybila/eBCSgen>

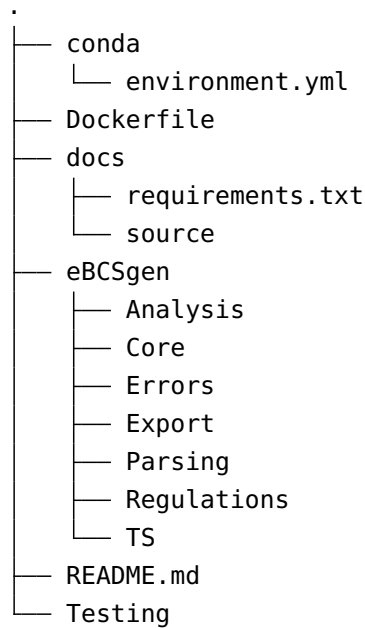


Figure 45: Top-level overview of eBCSgen GitHub repository.

ment with all dependencies. That is the purpose of the YML file, and the conda environment can be created using the following command:

```
conda env create -f conda/environment.yml
```

The Dockerfile is a text document that contains all the build instructions to assemble a docker image [Mer14]. The primary purpose of docker is to automate the deployment of applications inside software containers and the automation of operating system level virtualisation on Linux. In our case, the main motivation was to ship eBCSgen with all its dependencies, specifically including Storm model checker [Deh+17]. A new docker image² is built upon every release.

In docs directory, sources and requirements to build documentation can be found. The documentation is automatically built using Sphinx documentation generator from reStructuredText (.rst) configuration files and source code. It can automatically read docstrings in the code and include them in the resulting documentation pages. These are then, upon every release, automatically created using CI and deployed using ReadTheDocs³. This way, the documentation is always up to date.

The eBCSgen directory contains the source code of the tool and is discussed in detail in the following sections. An overview in a tree-like form is shown in Figure 46. In the README.md file, the user can find installation and development instructions as well as badges indicating the current status of CI/CD services.

² <https://hub.docker.com/r/sybila/ebcsgen>

³ <https://ebcsgen.readthedocs.io/>

Finally, the tool also contains an extensive set of tests that include unit and integration tests, with more details in [Section 6.6](#).

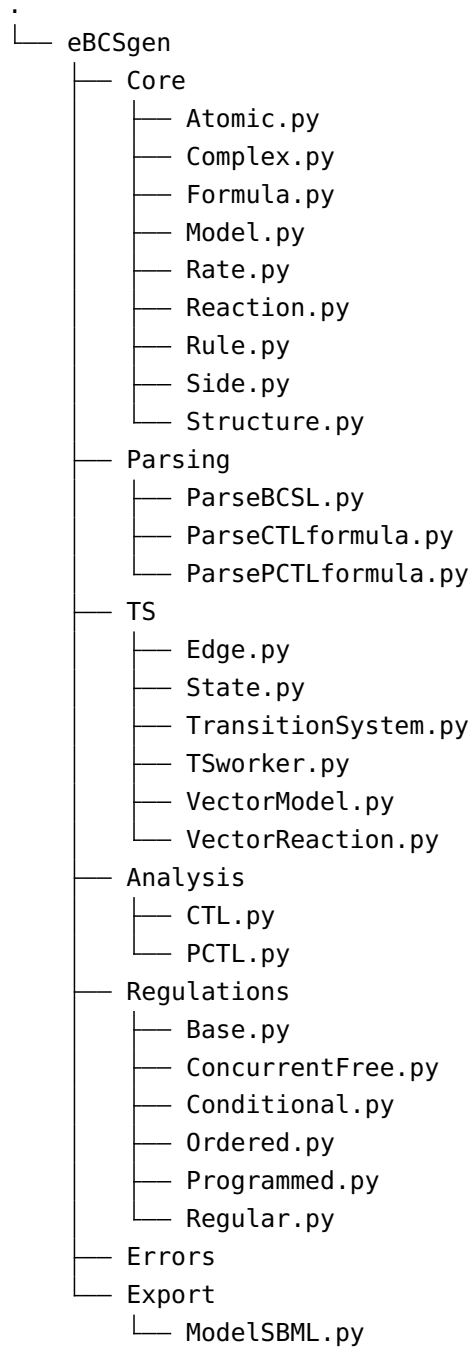


Figure 46: Structure of source code of eBCSgen GitHub repository.

6.1.1 Core

As a core of eBCSgen, we consider the most fundamental objects, such as agents, rules, or model. Their implementation corresponds to

the multiset rewriting-based approach described in [Section 3.4](#). We will refer to this approach as *the definition*.

Just like in the definition, the base objects are `Atomic`, `Structure`, and `Complex`, together forming agents of the language. They carry attributes such as state, name, and composition. They have also implemented various methods, e.g. to validate compatibility with another object of the same class, automatic gathering of signatures content, or, on the other hand, extend missing context based on the given signature.

These objects are used in `Rule`. Just like in the definition, the content of a rule is encoded in several attributes allowing smooth implementation of the rewriting. Each `Complex` is split into agents, containing `Atomics` and `Structures`; attribute `mid` indicates the index of the first agent from the right-hand side; `compartments` associate a compartment to each agent; `complexes` indicate where each complex starts and ends; and finally, `pairs` entangle agents from LHS to RHS.

From the implemented methods, it is worth mentioning there is a method to construct all possible multiset rewriting instantiations (or simply reactions), crucial for the multiset rewriting-based semantics. `Reaction` is a simplified version of the rule, where both sides are represented as a multiset of complexes with no complicated structure. The reason is that the objects are fully specified, and therefore there is no confusion about the meaning of the interaction. Additionally, in the rule, there are also methods for matching and replacement that, in the direct semantics (network-free), find all possible matches of the rule to a given state and apply the rule to chosen match, respectively.

A rule can be associated with a `Rate`. This is an expression of basic mathematical operations over complexes. It allows several symbolic transformations, such as partial evaluation and simplification, as well as complete evaluation in a given state.

The `Model` encapsulates a set of rules, an initial state in the form of a multiset of complexes, dictionary with mapping of parameters to their values, explicit enumeration of unknown parameters (used to determine whether the model is parametrised), optional regulation, and two signature functions (created automatically on the initialisation of the model).

The model supports some utility methods, such as extracting signatures or converting the model to vector representation ([Section 6.1.4](#)). It also has methods for analysis techniques that are executed directly on the model, these include static analysis ([Section 6.2.4](#)), network free simulation ([Section 6.2.1](#)), direct generating of state transition system ([Section 6.1.4](#)), and export to SBML using `multi` package ([Section 6.4.4](#)).

6.1.2 Model syntax

While the syntax for agents and rules is already defined ([Section 3.4](#)), the formal definition does not specify how the whole model should be written. Specifying it on the level of mathematical objects is not very practical. Therefore we developed an alternative notation that can be used to describe all the necessary parts of the model using plain text format.

```

model: rules (inits)? (definitions)? (complexes)? (regulation)?

rules: "#! rules" (rule)+
inits: "#! inits" (init)+
definitions: "#! definitions" (definition)+
complexes: "#! complexes" (cmplx_dfn)+
regulation: "#! regulation" regulation_def

init: const? complex
definition: param "=" number
cmplx_dfn: cmplx_name "=" complex

```

Figure 47: Grammar for the BCSL model in EBNF format.

In [Figure 47](#), there is a grammar specifying the syntax for the model. The already known terms are *rule* and *complex*. The model has five sections, with their beginning determined by symbol `#!` followed by the name of the respective section.

In the *rules* section, each rule is specified on a separate line. The *inits* section defines the initial multiset, where on each line, there is a complex, with an optional stoichiometric prefix (a constant indicating the number of its repetitions). Since the initial multiset can also be empty, this section is optional.

The *definitions* part is used to assign numerical values to parameters. One assignment is defined per line. This section is optional because any rate can be specified directly with embedded constants, or the model can be qualitative (i.e. without rates). Similarly, complex aliases can be defined in the optional *complexes* section.

Finally, the *regulation* section allows assigning single regulation to a model. The syntax for all possible regulations is specified in [Figure 48](#). The regulation type is specified on the first line after the section declaration. Then, a necessary number of lines defining the regulation itself follows.

The regular regulation consists of a single regular expression over rule labels using operators for concatenation `"."`, iteration `"*"`, and disjunction `"|"`. In the programmed regulation, successors for a rule are defined per line, written as a set in standard notation. The ordered and concurrent_free regulations are written in the same way, as an enumeration of pairs in a single line. Finally, the conditional regu-

```

regulation_def: "type" ( regular
                        | programmed
                        | ordered
                        | concurrent_free
                        | conditional
                        )

!regular: "regular" expression
expression: LABEL
          | expression "|" expression
          | expression "." expression
          | expression "*"

programmed: "programmed" successors+
successors: LABEL ":" "{" LABEL ("," LABEL)* "}"

ordered: "ordered" order ("," order)*
order: "(" LABEL ("," LABEL ")" )

concurrent_free: "concurrent-free" order ("," order)*

conditional: "conditional" context+
context: LABEL ":" "{" rate_complex ("," rate_complex)* "}"

```

Figure 48: Grammar for the usage of regulations in the BCSL model. The LABEL represents a defined rule label, typically composed from a string of allowed alphanumerical characters.

lation defines per line the forbidden context to a rule, denoted as a (multi)set in standard notation.

6.1.3 Parsing

It is necessary to provide a way to transform a model from its syntactic representation to its mathematical representation. The parsing module provides a series of transformers that can iteratively convert a model to its basic form (i.e. without any syntactic extension) and eventually create the `Model` object. The transforming process is provided by `lark` library [Shi22].

The model is first parsed to a tree, assigning tokens to individual model parts based on the EBNF grammar. If the model is syntactically incorrect, an object containing details about the syntax error is provided. Otherwise, a `lark` tree is created that can be consequently transformed.

The first step of transformation is to extract complex aliases and remove them from the rules by replacing them with respective com-

plexes. In the next step, the “zooming” syntactic extension is replaced by the basic notation. Then complexes are created inside rates, and regulation is processed. After that, all tree objects are in their basic form, and their respective objects can be constructed.

A significant advantage of the employed approach is that the parsing process takes as an input the keyword `start` that marks the first nonterminal of the grammar, allowing to process directly even smaller objects such as a complex or a rule. This is beneficial for testing purposes (Section 6.6), but above all, in the processing of (P)CTL formulae. Indeed, parts of the parsing module are also transformers for CTL and PCTL formulae. These are implemented using the same approach by providing an EBNF grammar and processed by `lark` transformers. After verifying the form of provided formula, the complexes present inside square brackets are parsed independently.

6.1.4 State transition system

There are two main algorithms to generate the state transition system – direct and indirect. We will first explain the direct approach as it corresponds with the definition. This approach iteratively creates new states by applying each individual rule to them, employing the `match` and `replace` methods. For that, a set of workers can be employed (class `TSworker`). These select a state from the pool of discovered states. Every rule is validated whether it can be applied in the selected state by checking that the matching is successful and the evaluation of the rate (if applicable) is non-zero. Then, if regulation is given, those not satisfying it are eliminated. Finally, corresponding agents are replaced per rule, producing a new state for every possible match. The computed rates are normalised accordingly, producing the probability of every transition.

While the direct approach works in theory, in practice, its complexity rises with the complexity of agents. Potentially, `match` and `replace` operations can become extremely costly when, for example, structure agents have many subdomains because the matching needs to consider all possible conformations. A possible workaround is an indirect approach via a reaction network. The trade-off of this step can be costly again because the corresponding reaction network can be exponentially larger than the number of rules. Still, in some cases, it can be beneficial in combination with speeding up exploration of the state transition system. The main reason for the speed up is that the reaction-based model can be easily converted to representation using vectors over integers, making the `match` and `replace` operations extremely fast.

The crucial step in the vectorisation of a model is ordering. The model has a method `create_ordering`, that creates an ordering of

all possible fully specified complexes. These are created using signatures from rules and the initial state and then stored in a fixed order.

Vectorisation of a state (i.e. multiset of complexes) then means encoding it as a vector of integers such that the number of repetitions of every complex present in the state is written on the corresponding position in the vector based on the ordering. For an example, see [Figure 49](#).

1. ordering:

$(A(d1\{p\})::cyt, A(d1\{u\})::cyt, B\{i\}::cyt, B\{a\}::cyt, B\{i\}::ext, B\{a\}::ext)$

2. state: $\{ B\{i\}::ext, B\{i\}::ext, A(d1\{p\})::cyt, A(d1\{u\})::cyt, A(d1\{u\})::cyt \}$

3. vector: $(1, 2, 0, 0, 2, 0)$

Figure 49: Example of encoding of a state to a vector representation. (1) ordering specifying the position of every possible complex, (2) given state as a multiset of complexes, and (3) vector representation of the state w.r.t. given ordering.

Since in the reaction, both sides are technically a multiset of complexes, the whole reaction can be encoded as a pair of vectors representing substrates and products. For an example, see [Figure 50](#). Then, having a given state and a reaction, deciding whether the reaction is enabled, it is sufficient to check whether the vector representing the state is element-wise greater than the vector representing the substrates. Finally, to apply the reaction (assuming it is enabled) to a state, one needs to subtract substrates and add products as vector operations.

1. $B\{a\}::cyt \Rightarrow B\{a\}::ext$

2. $(0, 0, 0, 1, 0, 0) \rightarrow (0, 0, 0, 0, 1, 0)$

Figure 50: Example of encoding of a reaction to a vector representation. We use the ordering from [Figure 49](#). (1) given reaction, and (2) vector representation of the reaction w.r.t. given ordering.

Complexes present in the associated rate expressions can also be converted to the vector representation using the same approach. The complex represents (possibly) multiple well-defined objects, and all these are marked by number one on their corresponding positions. Other positions have a value of zero. To evaluate the number of respective complexes in a state, first, this vector is multiplied by elements with the vectorised state, obtaining a new vector with only counts of relevant complexes left, and this resulting vector is summed, obtaining the final amount of matching complexes. This process is demonstrated in [Figure 51](#).

1. $2 \times [A()::\text{cyt}]$
2. $2 \times [(1, 1, 0, 0, 0, 0)]$
3. With state $(1, 2, 0, 0, 2, 0)$ from Figure 49:
 - a) $(1, 1, 0, 0, 0, 0) \times (1, 2, 0, 0, 2, 0) = (1, 2, 0, 0, 0, 0)$
 - b) $\text{SUM}(1, 2, 0, 0, 0, 0) = 3$
 - c) $2 \times 3 = 6$

Figure 51: Example of rate evaluation using vector representation. We use the ordering from Figure 49. (1) Given rate expression, (2) vectorised rate expression, (3) evaluation of the rate on a given model where (a) corresponding complexes are filtered, (b) resulting vector is summed to a number, and (c) the rate is evaluated.

With all parts vectorised, a vector model can be constructed. To sum it up, it contains a set of vectorised reactions, a vectorised initial state, the ordering, and, if given, a regulation. Such a model can be used to generate a state transition system indirectly with exactly the same algorithm as above, but the low-level match and replace operations are performed on vectors, making it significantly cheaper. The vector representation of a model can also be easily converted to a set of ODEs, thus enabling deterministic simulation, or the vectors can be fired directly in a stochastic simulation.

Finally, since the size of a transition system can grow to large numbers, we allow limiting the size by setting a global bound on molecule repetitions in states. States that would transition outside this boundary are aggregated to a special state (also referred to as *hell*), simplifying the infinite state transition system by a finite approximation.

6.1.5 Regulations

Regulations have a specific position in the implementation of eBCSgen. They are an extension that plays a role in the application of rules (or reactions in the case of indirect semantics). Specifically, once a subset of available rules is rendered as enabled for a given state, these are even further reduced based on the regulation. Therefore, every regulation has a `filter` method implemented, which is invoked in this step. Additionally, the introduction of regulations to eBCSgen required changing the implementation of a state since it needs to remember the history of rules that were applied in order to reach it. Indeed, with respect to regulations in general, states are distinguished not just by their content, but also by the history of rule applications.

For the *regular* regulation, the regular expression is handled by `regex` Python package [Bar22]. This regulation requires the full his-

tory of applied rules to determine whether the run is valid. However, during the exploration of the state transition system, it is useful to determine the branches that are forbidden by the regulation as early as possible. For that, the `regex` package supports partial RE matching. This enables us to determine that a prefix composed of rule labels does not satisfy the RE even before it is completed.

The *programmed* regulation stores the successor function in a dictionary, where the set of allowed rules can be easily looked up. This regulation requires only a history of the previously applied rule to determine the validity of the run. The validation is simply done by checking whether the to-be-applied rule is in the successors of the previous rule.

The *ordered* regulation is given by a set of pairs, defining the partial order on the rule labels. However, to simplify the validation process, this set is preprocessed by computing transitive closure. To determine the validity of rule application, it is enough to know previously used rule and make sure they are not in relation.

The *conditional* regulation is represented as a dictionary, assigning a multiset of agents to a rule label. The validation process then requires that if such a rule is about to be applied, the intersection of the current state and the specified multiset of agents must be empty. This regulation does not require any history of applied rules.

Finally, the effects of *concurrent-free* regulation can also be determined just in the current state without any knowledge of the history. The regulation is given as a set of pairs of prioritised and non-prioritised rules. From all enabled rules in the current state, every pair is checked, and if it is present in the set, the non-prioritised rule is eliminated.

6.2 ANALYSIS METHODS

This section provides some implementation details about analysis techniques proposed in [Chapter 5](#).

6.2.1 Simulation

As described in [Section 5.1](#), we have implemented two simulation algorithms – stochastic and deterministic. The stochastic approach works according to a modified version of the standard Gillespie algorithm (as outlined in [algorithm 1](#)). In addition, the implementation assumes an input parameter for maximal simulation time. This parameter is compared to the current simulation time at every step and serves as a termination criterion.

Since BCSL also supports regulations, they need to be reflected on the simulation side. Although it was not explicitly mentioned in the theoretical part, and we also did not pursue the research in this direc-

tion, it is still possible to fuse the regulation effects with quantitative aspects. At every step of the simulation, we evaluate rules that can be applied, and the history of applied rules is also available. Therefore, the regulation effect can be evaluated, and rules that do not satisfy its condition can be eliminated. In principle, this elimination of rules increases the probability of the application of remaining rules, which fits the purpose of the regulations.

The implementation of rules and reactions (the inferred multiset rules) both fit the top-level requirements of the stochastic simulation algorithm (i.e. all needed functions are implemented). Therefore, we allow the algorithm to run in both direct and indirect settings, as was described in the context of transition state enumeration in [Section 6.1.4](#). In that section, we also described the practicability of the vectorisation of states, and we do the same in the case of simulation. To simplify simulated data manipulation and storage, these are stored in `DataFrame` from the `pandas` library. This is, for example, beneficial for computing the mean of multiple runs, where we employ `groupby.mean` implemented for the `DataFrame`. This function groups similar time points of runs and computes their mean while excluding missing values.

The deterministic simulation approach uses a set of ODEs to describe the average model behaviour, and by solving this system of equations, we can sample this average trajectory. Technically, this approach is supported by the vectorised model. Indeed, as outlined in the algorithm description, first, we need to enumerate the reaction network. The network is then vectorised and used to represent the ODE for individual species symbolically. These are then passed to the ODEs solver. We use `lsoda` solver [HP05], implemented by `odeint` function in `scipy.integrate` package [Vir+20]. It can solve the initial value problem for stiff or non-stiff systems of first-order ODEs. The computed time series are again stored using `DataFrame` from the `pandas` package.

6.2.2 Model checking

We have implemented support for CTL and PCTL model checking in the tool. The main motivation behind the support of CTL analysis is to analyse the qualitative behaviour of models with regulations. Indeed, the inherent feature of regulations is to reduce the branching of the transition system, making CTL model checking an ideal tool for validating its effects.

We implemented this feature using `pyModelChecking` [Cas22] Python package developed for explicit model checking for multiple temporal logics. The model checking procedure of this package takes as input a Kripke structure and a CTL formula in a specific format. Therefore, we implemented a translation of the transition system to the

Kripke structure, containing a graph composed of states and edges, with states labelled by particular atomic propositions specified in the CTL formula. To gather the APs, we also implemented a tailor-made parser for CTL formulae using `lark` library. It is able to validate the formula, transform it into a suitable form, and extract APs. The APs are then evaluated for individual states (for details, see [Section 5.2](#)) and corresponding labels assigned to them.

On the other hand, the probabilistic behaviour of the BCSL model can be analysed with respect to PCTL properties. PCTL is an extension of CTL which allows for probabilistic quantification of properties. The semi-symbolic method explained in [Section 5.2](#) is implemented in Storm model checker [\[Deh+17\]](#) and employed by eBCSgen. The tool allows checking whether a given probability threshold is satisfied or finding the probability of satisfaction for given path formula. The process requires generating an explicit Storm file from a given transition system. Similarly to CTL formulae, we have also created a parser to extract APs and create labels for states that satisfy them.

6.2.3 Parameter synthesis

The proposed parameter synthesis methods in [Section 5.3](#) is implemented using Storm model checker [\[Deh+17\]](#). In addition, Storm uses parameter lifting optimisation [\[Qua+16\]](#), which improves the state-elimination approach.

The parameter synthesis is enabled only for parametrised models. In our case, the required data structure for the procedure is pMC in the form of PRISM model [\[KNPo6\]](#). A transition system can be directly exported into the PRISM model.

Regarding the processing of the PCTL formula, we have already commented on it in [Section 6.2.2](#). The only issue is the presence of patterns in atomic propositions of the PCTL property. Since a pattern basically compactly represents all possible instantiated agents (resp. multisets), it can be expressed as a sum of these agents. To that end, we introduce *formulae*, which encode the sum in the PRISM model. Once defined, properties operating with their identifier (in our case, the pattern itself) are valid.

Suppose the PCTL formula has a defined probability threshold. In that case, Storm computes the partitioning of the given parameter space (defined by the user) to regions which satisfy (resp. violate) the property. The output from Storm is additionally post-processed to a CSV file.

If the threshold is not given, a probability function of parameters is computed instead, which evaluates to the probability of satisfaction for a particular parametrisation. With such a function, the robustness of the formula can be computed. For example, package `scipy` [\[Vir+20\]](#) can take the symbolic representation of the function and compute

its definite integral in the assumed parameter space. Moreover, since some discontinuities are possible in the function, it can be beneficial first to analyse it using package `sympy` [Meu+17] and then integrate without these particular points.

6.2.4 Static analysis

The implementation of static analysis techniques to improve the scalability issues of exhaustive computational methods corresponds to their definitions from [Section 5.5](#).

The method for *elimination of redundant rules* is implemented in a less aggressive setting, i.e. it only detects and marks potentially redundant rules without any automatic elimination step. This is left for the user to decide. The absence of a redundant rule in the model does not change the behaviour, but there still can be a reason why it should be left there. Technically, such rules are detected using compatibility operation, where both sides of each pair of rules are checked.

Another analysis technique is used to *reduce the context* of the model to the minimal level in order to produce a smaller and more abstract model. The resulting model still preserves some properties while making the analysis of the model computationally simpler. The technique is implemented by a function which reduces the context recursively for rules and agents used in them. Then the produced rules are checked for meaningfulness, i.e. whether they are still well-defined.

Finally, the static analysis of *unreachability* can be used to check whether an agent is unreachable without enumerating the transition system. This analysis is based on the idea that in order to reach an agent, there must be a rule that produces either the agent itself or at least a compatible one. Therefore, all rules are sequentially checked for this property.

6.3 VISUALISATION

An essential part of the presentation of data produced by eBCSgen is visualisation. The result for both types of simulation can be visualised in an interactive chart, and the result of parameter synthesis can be displayed in a visualisation which shows slices of 2D parameter space projections. Moreover, it is possible to visualise the generated transition system.

6.3.1 Transition system

The transition system can be visualised as a network in an interactive chart, implemented using `visjs` JavaScript library [Alm22]. The

nodes represent the states of the transition system, while the edges represent transitions between the particular states. It is possible to adjust the position of nodes by dragging them, highlight their content by clicking on them, and also display a particular transition by clicking on edge. The initial state from which was the transition system generated is shown in orange colour.

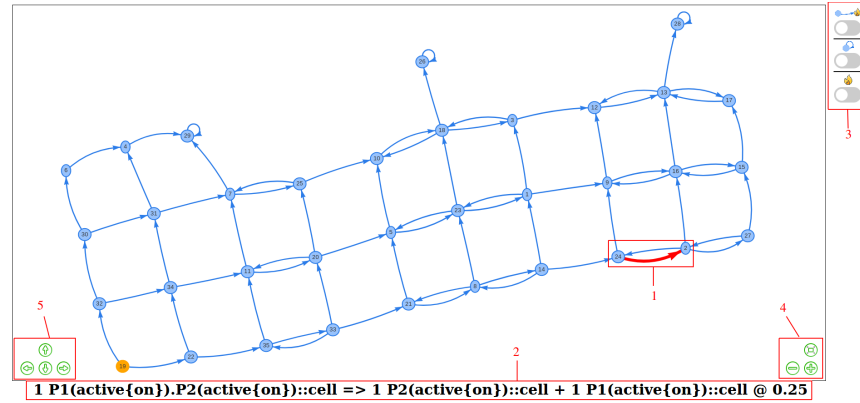


Figure 52: Visualisation of the transition system computed for model.bcs. The key features are highlighted in red frames: (1) highlighted edge, (2) contents of the highlighted object, (3) options for graph details, (4) control panel for zooming options, (5) navigation panel.

Figure 52 shows an example of the visualisation of the transition system. The highlighted features are:

1. *highlighted edge* – edges and nodes can be highlighted by left-clicking on them.
2. *contents of the highlighted object* – the contents of the clicked node or edge are displayed. An edge contains particular interaction responsible for the transition with evaluated probability (or a probability function of parameters). A node contains agent counts present in the state.
3. *options for graph details* – it is possible to adjust three features of the graph by switch buttons: *top* – nodes with a transition to special state “hell” (if any) are highlighted by a squared border, *middle* – self-loops are hidden, *bottom* – special state “hell” (if any) is hidden.
4. *zooming options* – zooming and centering buttons. Zooming can also be achieved by using a mouse wheel.
5. *navigation panel* – vertical and horizontal navigation in the chart. It can also be achieved by left-clicking of mouse followed its dragging on an empty space in the chart.

6.3.2 Simulation

The simulation time series can be visualised in an interactive chart, implemented using `plotly` library [Inc15]. The plot provides basic functionality such as zooming, curves filtering, and exporting png picture. Examples of visualisation for deterministic and stochastic simulations are available in Figure 53.

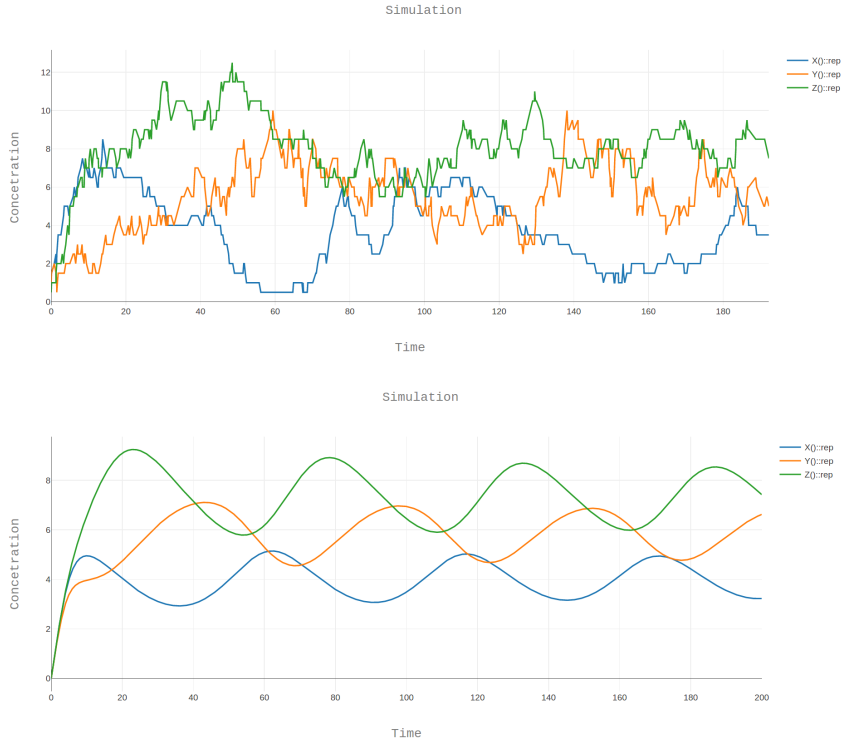


Figure 53: Visualisation of time series data as a result of the simulation. The top picture shows the result of stochastic simulation, and the bottom picture shows the result of deterministic simulation.

6.3.3 Parameter synthesis

The results of parameter synthesis can be visualised in a chart. The visualisation shows green, red, and grey regions where the satisfiability of PCTL formula is true, false, and unknown, respectively. The visualisation allows the user to change chosen parameters on the X and Y axis, which is particularly useful when there are more than two parameters. In that case, slices of parameter space in other dimensions can be chosen. An example of the visualisation of parameter synthesis results is available in Figure 54.

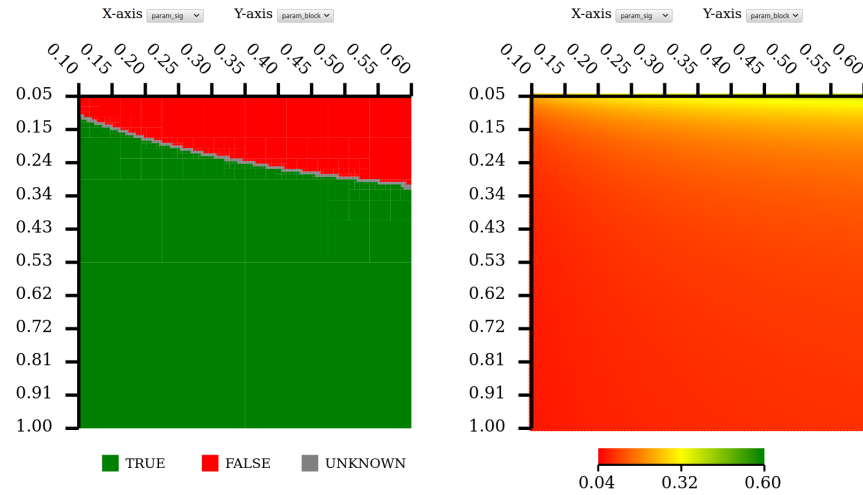


Figure 54: Visualisation of results of parameter synthesis. The *left* picture depicts the partitioning of parameter space as a result of parameter synthesis *with* a defined probability threshold (0.2 in this case). The *right* picture depicts the sampling of the probability function of parameters as a result of parameter synthesis *without* a defined probability threshold. Both pictures display the same parameter space.

6.4 DISTRIBUTION

The tool is distributed using several approaches to reach a broader audience. A standardised way is to use conda package manager ([Section 6.4.1](#)). However, to make the installation process as simple as possible for the user, this requires all the dependencies to be available in conda, which is not the case. A complete solution is to provide a docker container ([Section 6.4.2](#)) that already has all the dependencies installed. However, the target audience is not always skilled enough to use such an advanced distribution technique. For that reason, we also distribute the eBCSgen as a series of Galaxy wrapper tools ([Section 6.4.3](#)), which promotes the command line tool to a web-based unified interface.

6.4.1 Conda

Conda [[Ana](#)] is an open-source package management system and environment management system that allows to create and install separate environments within a filesystem and install software within them. Initially, it was created for Python tools, but it can package and distribute software for any language.

Conda allows simple creating and switching between environments. For example, to create an environment called `ebcsgen_env` and activate it, the following commands can be used:

```
conda create --name ebcsgen_env python=3.9
conda activate ebcsgen_env
```

There are various repositories called *channels* that store the packages. There is the bioconda channel [Grü+18] maintained specifically for biologically-oriented tools. eBCSgen is registered in this channel, and its newest version can be installed with the following command:

```
conda install --channel bioconda --channel conda-forge eBCSgen
```

The registration to a channel is governed by so-called *recipes*. The recipe specifies the location of package sources, required dependencies and metadata. Bioconda channel has a very useful feature for monitoring new versions of its packages. A bot iteratively inspects all source GitHub repositories and detects any new release. On release, it triggers an automatic recipe update and creates a new conda package. This way, the conda package is always up to date with the development managed on GitHub.

The downside of conda is that all dependencies need to be available in a conda channel. This is, however, not always the case, especially for software with specific requirements. Even for Python tools, there are other package managers such as `pip` that make this process intricate. In our case, Storm model checker [Deh+17] is not available on conda, which makes the dependencies list incomplete, and this specific dependency needs to be installed separately.

6.4.2 Docker

A possible improvement compared to conda provides docker [Mer14], but it is less lightweight. The main motivation is to ship eBCSgen with all dependencies, including Storm. Since Storm is already available in a docker container, we could build the image from it. The Dockerfile is a text document that contains all the build instructions to assemble a docker [Mer14] image. As a first step, it is necessary to specify the source container, that is Storm in this case.

```
FROM movesrwth/storm:stable
```

Then the only remaining steps are to install conda and use it to install eBCSgen. The created docker image is available on DockerHub⁴ and a new image is built automatically on every release.

6.4.3 Galaxy

Galaxy is an open-source platform for data analysis. It aims to give biomedical researchers access to computational biology without requiring them to understand computer programming. Galaxy allows

⁴ <https://hub.docker.com/r/sybila/ebcsgen>

using tools from various domains through its unified graphical web interface. These can be plugged into a scientific workflow, satisfying FAIR principles [Wil+16].

6.4.3.1 User interface

First, let us give the basics of using Galaxy. While the described features are general, we use for this purpose a tailor-made instance for eBCSgen called Galaxy-BioDivine⁵.

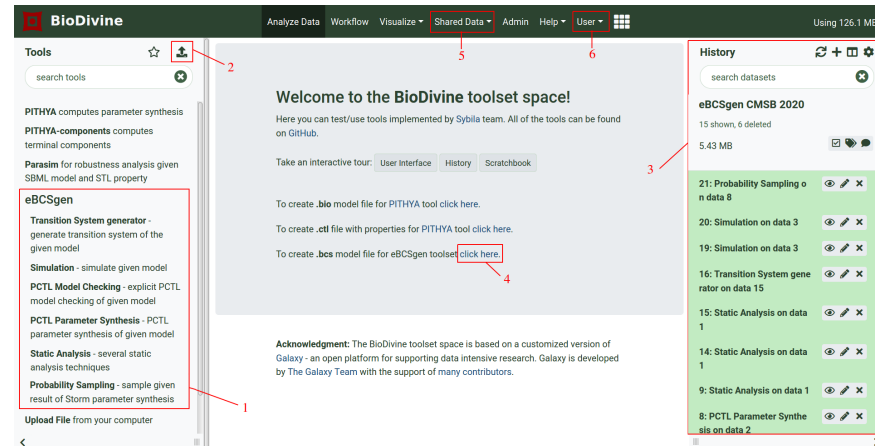


Figure 55: A screenshot of the main page of the Galaxy tool. The key components of UI are highlighted by red frames: (1) available eBCSgen Analytical tools, (2) uploading utility, (3) history of your computations, (4) create new BCSL model, (5) available shared data, (6) userspace.

The Galaxy tool is composed of several components. Figure 55 shows the homepage of Galaxy tool with highlighted key components:

1. *eBCSgen Analytical tools* – the main functionality of eBCSgen can be found here. Individual tools and how to use them is described in Section 6.4.3.2.
2. *uploading utility* – upload local files and download data from the web by entering URLs.
3. *history of your computations* – data space where all files and analysis results are available. This helps to keep track of analysis results origin, allows to easily chain tools, and share reproducible data.
4. *create new BCSL model* – open an empty file in BCSL editor which allows to create a new model from scratch.
5. *shared data* – published data libraries and histories available as reference data.

⁵ <https://biodivine-vm.fi.muni.cz/galaxy/>

6. *userspace* – it is recommended to register and stay logged in at all times. This enables some features such as visualisation of results and saves history for other sessions.

Every tool in the Galaxy UI has its interface, which allows the user to comfortably fill in all required arguments for the tool in order to execute it successfully. In [Figure 56](#), there are highlighted some of the key components:

1. *tool input files* – Galaxy is based on file transformation principle, i.e. each tool takes as input a file and produces an output file (can be multiple on both ends).
2. *additional arguments* – along with files, the tool might require some additional arguments, e.g. textual field, numerical value, or a variant choice.
3. *hidden options* – tool can have some optional or repetitive arguments which can be collapsed/hidden.
4. *execute button* – once all arguments have been filled, the tool can be executed, and its results will appear in the history.

The screenshot shows the Galaxy UI interface for the 'Transition System generator' tool. The interface is divided into several sections:

- Title Bar:** Displays the tool name 'Transition System generator - generate transition system of the given model (Galaxy Version 1.0.0)' and buttons for 'Favorite' and 'Options'.
- Model file:** A section for selecting input files, showing a file named '15: Static Analysis on data 1'.
- Bound [optional]:** A text input field for additional arguments.
- Advanced Options:** A section for hidden options, indicated by a red box and number 3.
- Email notification:** A section with 'Yes' and 'No' buttons for sending notifications.
- Execute Button:** A blue button with a checkmark and the text 'Execute', indicated by a red box and number 4.

Figure 56: A screenshot of a tool in Galaxy UI. The highlighted components in red frames are: (1) an input file to the tool, (2) the argument of the tool, (3) hidden advanced options, (4) the execution button.

The history is a key component of Galaxy UI. It is a timeline of tool results and allows the user to reproduce the tool execution completely. The background colour of the file indicates its state: green – the execution is finished successfully, orange – the execution is

in progress, and red – the execution failed (check the error report). In [Figure 57](#), there are highlighted some of the key components of an item in the history:

1. *ID of the data* – unique identification in the history.
2. *ID of the source data* – the ID of the file used as an input file for the tool.
3. *information* – complete information about the file (e.g. size, format) and job it was created by (e.g. input arguments).
4. *re-run the job* – possibility to run the job again with the same or modified arguments.
5. *visualise* – displays visualisation options for the data. All displayed visualisations can be used to produce a graphical output for the data. However, we recommend particular visualisation tailored for the data format for each individual tool in section [Analytical tools](#).
6. *view file content* – show entire file content.
7. *peek of the file content* – just a peek of the file content or file statistics.

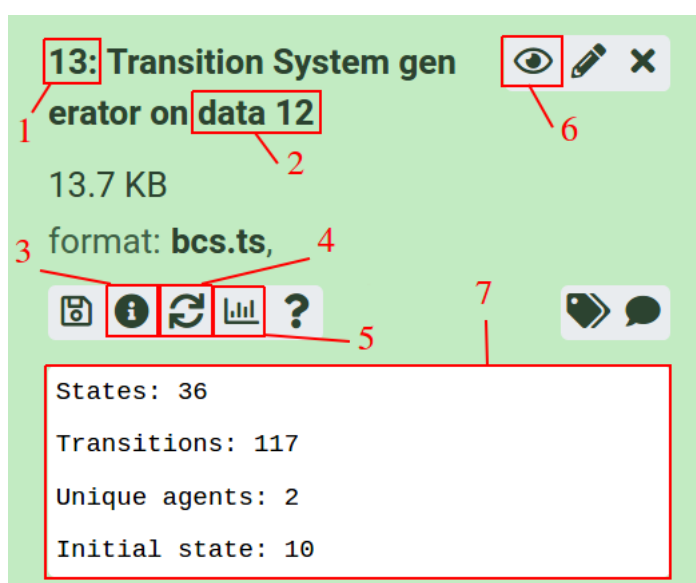


Figure 57: A screenshot of a history item in Galaxy UI. The highlighted components in red frames are: (1) the ID of the data, (2) the ID of the source data, (3) complete information about the data, (4) run the job again, (5) visualise the data, (6) view file content, (7) peek at the file content.

There is also build-in BCSL editor (Figure 58), modified to send requests to a BCSL parser and this way provide an interactive environment to create and edit models. However, this editor is only available in out Biodivine instance and is not distributed in general.

```

1  #! rules
2  r1_S ~ P(S{i}::cell => P(S{a}::cell
3  r1_T ~ P(T{i}::cell => P(T{a}::cell
4  r2 ~ P()::cell => P()::out
5
6  #! inits
7  1 P(S{i},T{i}::cell
8
Unexpected ":", expected one of ")", "," at position [row: 2, column: 14]

```

Figure 58: A screenshot of a BCSL editor in Galaxy. It can highlight the syntax of the language as well as announce detected syntactic errors.

6.4.3.2 eBCSgen wrappers

In this section, we describe individual Galaxy wrappers created to cover the functionality of the tool. These wrappers are available in the main Galaxy toolshed⁶, a Galaxy tools repository making them installable within any Galaxy instance. The sources of the wrappers are available in a GitHub repository⁷, with automatic CI action that deploys a new version on changes. The wrappers use the docker image (Section 6.4.2) since it contains all dependencies.

All wrappers are also described in detail in the tutorial (accessible in Appendix A), including running examples that can be used for user testing of the wrappers.

TRANSITION SYSTEM GENERATOR The transition system generator wrapper (Figure 59) generates the state transition system of the given model. It takes as an input Model file in `bcs1.model` format, as specified in Section 6.1.2. It also has several optional parameters. Bound represents the maximal multiplicity of any agent in any state. If not given, an implicit bound is computed automatically for potentially infinite systems. It implies the construction of a special so-called “hell” state aggregating all states beyond the bound. Maximal computation time gives a possibility to specify the time (in seconds) boundary for the computation. Finally, Maximal TS size limits the maximal number of nodes in the final transition system, terminating the computation once this number is reached.

The output of the wrapper is a file in `bcs1.ts` format, which is JSON with the following structure:

⁶ <https://toolshed.g2.bx.psu.edu/>

⁷ <https://github.com/sybila/galaxytools>

eBCSgen transition system generator - generate transition system of the given model (Galaxy Version 2.1.0_galaxy0)

! Please provide a value for this option.

Model file

No bcsl.model dataset available.

Provide a BCSL model file

Choose network-free approach:

Indirect

Advanced Options

Bound [optional]

Maximal computation time (in seconds) [optional]

Maximal TS size [optional]

✓ Execute

Figure 59: A screenshot of transition system generator Galaxy wrapper.

- **ordering:** Ordered list of all distinct agents.
- **nodes:** Set of nodes representing states of the transition system. Each of them contains a unique ID and a tuple of numbers, representing the multiplicity of the agent on the respective position given by the ordering.
- **edges:** Set of edges representing transitions. Each edge contains the ID of the *source* node (*s*), and ID of the *target* node (*t*), and the probability of the transition (*p*). In the case of the parametrised model, the probability is replaced by a probability function of parameters.
- **initial:** ID of the initial state.

Additionally, the output can be visually explored using the dedicated visualisation [Section 6.3.1](#).

SIMULATION The simulation wrapper ([Figure 60](#)) enables running simulations of a BCSL model within the Galaxy. It takes as an input Model file in `bcsl.model` format. The parameter Choose simulation method allows selecting the simulation method, either stochastic or deterministic. For the stochastic case, additional parameters to choose a network-free approach (direct or indirect method), specify the Number of runs (the average of individual runs is then computed) and Maximum simulation time.

The simulation tool generates time series for every agent. The result is stored in a CSV file, where the first column represents time and the rest of the columns store values of individual agents. Note

The screenshot shows the 'eBCSgen simulation - simulate given model (Galaxy Version 2.1.0_galaxy0)' interface. It features a light blue header bar with a star icon and a dropdown arrow. Below the header, a light blue box contains a warning icon and the text 'Please provide a value for this option.' followed by the label 'Model file'. A file selection area shows 'No bcsl.model dataset available.' with a dropdown arrow and a folder icon. Below this, the text 'Provide a BCSL model file' is displayed. The main form area includes several input fields: 'Choose simulation method:' with a dropdown menu set to 'Stochastic'; 'Choose network-free approach:' with a dropdown menu set to 'Indirect'; 'Number of runs:' with a numeric input field set to '5'; and 'Maximum simulation time:' with a numeric input field set to '125'. At the bottom left, there is a blue button with a checkmark and the text 'Execute'.

Figure 60: A screenshot of simulation Galaxy wrapper.

that specified simulation time might not be achieved in the case of stochastic simulation when there are multiple runs due to averaging the runs. The output can be visually explored using the dedicated visualisation [Section 6.3.2](#).

CTL MODEL CHECKING The CTL model checking wrapper ([Figure 61](#)) enables to perform the model checking of a transition system with respect to a CTL formula. As an input, it takes Computed Transition system file in `bcs1.ts` format and a CTL formula expressing property to be checked on the transition system.

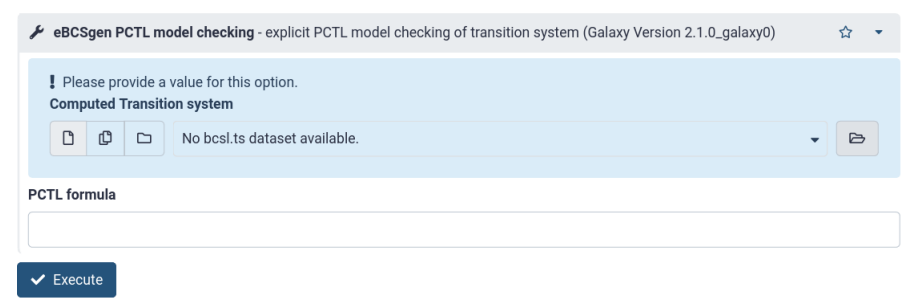
The screenshot shows the 'eBCSgen CTL model checking - explicit CTL model checking of transition system (Galaxy Version 2.1.0_galaxy0)' interface. It features a light blue header bar with a star icon and a dropdown arrow. Below the header, a light blue box contains a warning icon and the text 'Please provide a value for this option.' followed by the label 'Computed Transition system'. A file selection area shows 'No bcs1.ts dataset available.' with a dropdown arrow and a folder icon. Below this, the text 'CTL formula' is displayed above a text input field. At the bottom left, there is a blue button with a checkmark and the text 'Execute'.

Figure 61: A screenshot of CTL model checking Galaxy wrapper.

The textual result is stored in a `ctl.result` file. The text contains number of states satisfying the formula and the boolean value for the initial state.

PCTL MODEL CHECKING The PCTL model checking wrapper ([Figure 62](#)) enables to perform the model checking of a transition system with respect to a PCTL formula. As an input, it takes Computed

Transition system file in `bcs1.ts` format and a PCTL formula expressing a property regarding the probability of an event to occur. The tool allows checking whether a given probability threshold is satisfied or finding the probability of satisfaction for the given path formula.



eBCSgen PCTL model checking - explicit PCTL model checking of transition system (Galaxy Version 2.1.0_galaxy0)

! Please provide a value for this option.

Computed Transition system

No bcs1.ts dataset available.

PCTL formula

Execute

Figure 62: A screenshot of PCTL model checking Galaxy wrapper.

The textual result is stored in a `storm.check` file. The text contains some details about Storm model checker performance. Finally, the boolean or numerical result can be found as `Result` (for initial states) in the file. Additionally, any errors Storm encountered are also shown in this file.

PCTL PARAMETER SYNTHESIS The PCTL parameter synthesis wrapper (Figure 63) enables to run parameter synthesis of a parametrised transition system with respect to a PCTL formula. It takes as an input Computed Transition system file in `bcs1.ts` format and PCTL formula expressing property to be used to perform parameter synthesis on the transition system. If the formula has a form with a probability threshold, then the partitioning of the given parameter space (defined by the user) to regions which satisfy (resp. violate) the property is computed. In that case, Parameter intervals need to be provided – an interval of allowed values has to be specified for each unknown parameter, together forming parameter space. If the threshold in the formula is not given, a probability function of parameters is computed instead, which evaluates to the probability of satisfaction for a particular parametrisation.

The format of the output depends on the input formula. In the case *with* probability threshold, the output is in CSV format and contains segmentation of specified parameter space to regions where the given property is satisfied and violated. For some of the regions, the satisfiability might not be decided due to efficiency reasons. In the other case, a file in `storm.sample` format contains computed probability function of parameters, which can be evaluated to the probability of satisfaction for particular parametrisation.

Additionally, the output can be visually explored using the dedicated visualisation Section 6.3.3.

eBCSgen PCTL parameter synthesis - PCTL parameter synthesis of given parametric transition system (Galaxy Version 2.1.0_galaxy0)

! Please provide a value for this option.
Computed Transition system

PCTL formula

Parameter intervals

1: Parameter intervals

Parameter name:

Interval start:

Interval end:

Figure 63: A screenshot of PCTL parameter synthesis Galaxy wrapper.

STATIC ANALYSIS The static analysis wrapper (Figure 64) enables to run various static analysis techniques on a BCSL model within Galaxy. It takes as an input Model file in `bcsl.model` format. The user can select a specific analysis method: `Static non-reachability` checks whether a `Complex` agent is unreachable before enumerating the entire transition system of the model, `Rule redundancy elimination` detects potentially redundant rules in the model, and `Context based reduction` minimises the context in agents in order to produce a smaller and more abstract model.

eBCSgen static analysis - run static analysis techniques on given BCSL model (Galaxy Version 2.1.0_galaxy0)

! Please provide a value for this option.
Model file

Provide a BCSL model file

Choose static analysis method:

Complex agent:

Figure 64: A screenshot of static analysis Galaxy wrapper.

The results of the tool depend on the particular method which was chosen. For `Static non-reachability` method, the output is textual

and states whether the agent of interest *cannot be reached* or *can possibly be reached* in the model. For Rule redundancy elimination method, the output is a BCSL model with comments added to the file indicating potential (if any) redundant rules. Finally, for Context based reduction method, the output is a BCSL model with all features eliminated, leaving only rules for complex formation and dissociation, production, and degradation.

SBML EXPORT The SBML export wrapper (Figure 65) enables to run an export of a BCSL model in SBML format. It takes as an input Model file in `bcsL.model` format. The results of the tool is an `.xml` file containing equivalent model in SBML format.

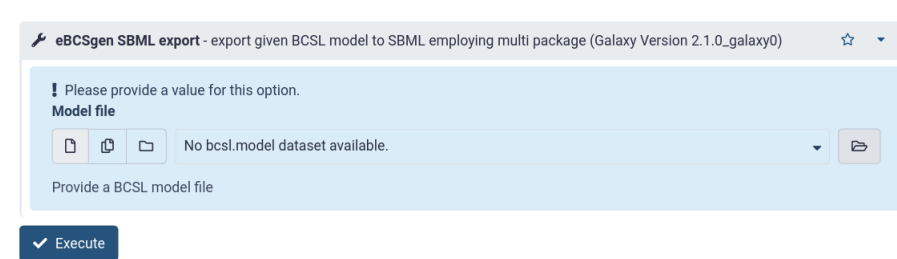


Figure 65: A screenshot of SBML export Galaxy wrapper.

6.4.4 SBML-multi

Finally, to join the community effort on interoperability among other rule-based languages, we implemented support for model export to SBML standard using the package `multi`. The package extends the SBML Level 3 core with the *type* concept, and therefore reaction rules may contain species that can be patterns and be in multiple locations in reaction rules. It allows the SBML standard to encode rule-based models using their native concepts for describing reactions instead of having to apply the rules and unfold the networks prior to encoding in SBML format. This allows us to save BCSL models in a standard format, which enables their analysis beyond the scope of eBCSgen. We implemented the export using a package `libSBML` [Bor+08].

6.5 DOCUMENTATION

Documentation of eBCSgen is deployed using ReadTheDocs. It is built automatically using continuous integration on every pull request and merge with the master branch. Technically, ReadTheDocs provides integration using a Webhook in the GitHub repository that will automatically notify ReadTheDocs when some changes have been made. This process makes sure that the documentation is always up to date.

For the generating of documentation, we use Sphinx. It is accompanied by a configuration file and reStructuredText (.rst) documentation files. In general, in .rst files, it is possible to reference other .rst files, but also individual classes and modules from source code docstrings.

The documentation is available online⁸ and contains the readme file with installation instructions and documentation trees for individual modules, including a description of the most important classes and their methods.

6.6 TESTING

The functionality of the tool is verified using unit testing. The tests are available in `Testing` directory, together constituting approximately one hundred cases with 96% test coverage. We are testing individual methods of classes from the core, generating of transition system, simulations (with mocked randomness to make simulation reproducible), effects of regulations, model checking and parameter synthesis methods, and finally export to SBML.

6.7 SUMMARY

This chapter provides insight into the internal gears of the eBCSgen tool. We provide information on how individual modules are implemented, including the core objects, parsing, interpretation of rules, and effects of regulations. Additional implementation details regarding analysis techniques are provided where necessary, giving a complete picture of the proposed approaches.

The tool is implemented as a Python package distributed using conda package manager. To reach the life-sciences oriented audience, a series of Galaxy wrappers are implemented, covering the whole functionality of the tool. These are also accompanied by several visualisation techniques, improving the user experience with the tool. Finally, eBCSgen also complies with the standards in systems biology for rule-based languages to allow the model exchange to external tools.

⁸ <https://ebcsgen.readthedocs.io/>

*God made the integers, all the rest is
the work of man.*

LEOPOLD KRONECKER

IN this chapter, we demonstrate a variety of case studies using the modelling approach with BCSL. On examples from the biological domain, we first highlight some interesting features of BCSL and how it can be applied to biological phenomena in general. Then, we demonstrate analysis techniques developed for BCSL on such models. We focus on the most significant contributions to the analysis base – namely, we show examples of the application of parameter synthesis and static analysis. Finally, we also provide several examples of how regulations can be used in the modelling, including a more extensive case study demonstrating how they can be particularly used in the early-stage development of models.

7.1 MODELLING WITH BCSL

The purpose of this section is to show evidence of how the modelling approach employed by BCSL can be used to tackle typical patterns found in biology.

7.1.1 Circadian clock

The circadian clock of cyanobacteria [Gol+97; Ish+98], consisting of only three proteins KaiA, KaiB, and KaiC, can be reconstituted *in vitro*, making it the simplest post-translational circadian oscillator currently known [Nak+05]. The additional presence of ATP is, as usual, neglected in model.

The schematic overview of the clock is available in [Figure 66](#). Using BCSL, we described this system in a very concise way while capturing its complex behaviour. The complete model is available in [Figure 67](#). In the following, we describe some important details and interesting aspects that make this model valuable.

Out of the three proteins, the structurally most complicated one is KaiC, which naturally forms a homohexamer. It contains two phosphorylation sites important for its modelling – threonine on position 432 and serine on position 431 [Pat+04] (UniProt: :Q79PF4). These two

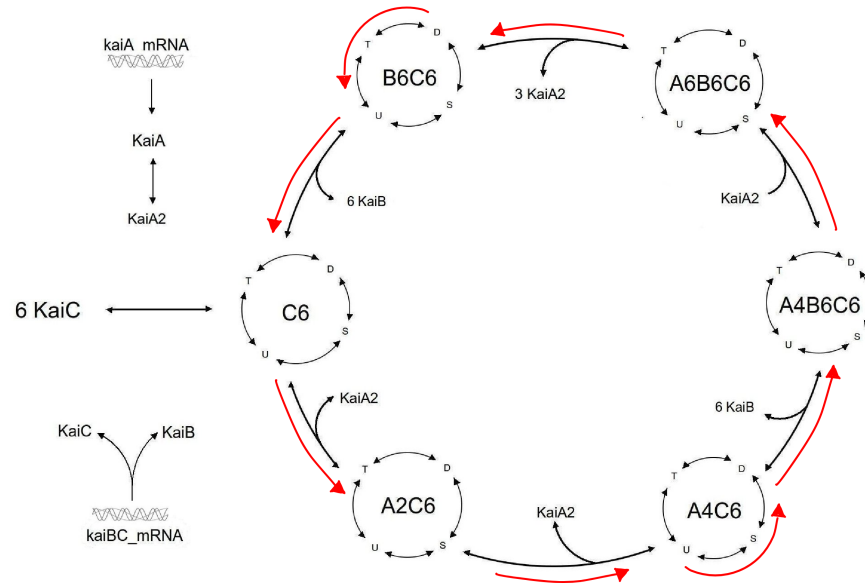


Figure 66: Scheme of the circadian clock. The model exhibits extensive non-deterministic behaviour, which is illustrated by bidirectional arrows in the main cycle in combination with small cycles, representing possible (de)phosphorylation steps. A typical scenario of the clock mechanism is highlighted in red.

amino acid residues can be modified by phosphorylation. However, the sites are accessible only when the protein forms a hexamer.

The KaiA, forming a dimer, binds to the KaiC hexamer and promotes KaiC phosphorylation, whereas the effect of KaiB binding to the KaiC hexamer is opposite as it stimulates KaiC dephosphorylation [Iwa+02; Kat+03; Vil+13]. Several complexes can be formed from mentioned proteins based on the state of KaiC hexamer.

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 [HBA13]. When it comes to the 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. This is defined by a rule *rpS* (resp. *rpT*) where phosphorylation of serine (resp. threonine) residue is possible for all complexes with KaiC included. This is denoted by a variable φ used in place of a complex, and conciseness is increased by using $:$ operator to “zoom” inside of the complex.

An interesting aspect of the cyanobacterial circadian clock mechanism is the formation of KaiC hexamers. The condition that the KaiC protein can be (de)phosphorylated mostly 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


```

#! rules
# (de)phosphorylation
rpS ~ S{u}:KaiC():?::cyt ⇔ S{p}:KaiC():?::cyt ;
? = {KaiC6,KaiA2C6,KaiB6C6,KaiA4C6,KaiA4B6C6,KaiA6B6C6}
rpT ~ T{u}:KaiC():?::cyt ⇔ T{p}:KaiC():?::cyt ;
? = {KaiC6,KaiA2C6,KaiB6C6,KaiA4C6,KaiA4B6C6,KaiA6B6C6}

# KaiC complex formation
rC6 ~ 6 KaiC():cyt ⇔ KaiC6::cyt

# other complexes
rA2 ~ KaiA():cyt + KaiA():cyt ⇔ KaiA2::cyt
rB6C6 ~ KaiC6::cyt + 6 KaiB():cyt ⇔ KaiB6C6::cyt
rA2C6 ~ KaiA2::cyt + KaiC6::cyt ⇔ KaiA2C6::cyt
rA4C6 ~ KaiA2::cyt + KaiA2C6::cyt ⇔ KaiA4C6::cyt
rA4B6C6 ~ KaiA4C6::cyt + 6 KaiB():cyt ⇔ KaiA4B6C6::cyt
rA6B6C6_1 ~ KaiA2::cyt + KaiA4B6C6::cyt ⇔ KaiA6B6C6::cyt
rA6B6C6_2 ~ 3 KaiA2::cyt + KaiB6C6::cyt ⇔ KaiA6B6C6::cyt

# transcription
rfA ~ ⇒ KaiA():cyt
rfBC ~ ⇒ KaiB():cyt + KaiC(S{u},T{u}):cyt

#! complexes
KaiC6 = KaiC().KaiC().KaiC().KaiC().KaiC().KaiC()
KaiB6 = KaiB().KaiB().KaiB().KaiB().KaiB().KaiB()
KaiA2 = KaiA().KaiA()
KaiA2C6 = KaiA2.KaiC6
KaiB6C6 = KaiC6.KaiB6
KaiA4C6 = KaiA2.KaiA2.KaiC6
KaiA4B6C6 = KaiA2.KaiA2.KaiC6.KaiB6
KaiA6B6C6 = KaiA2.KaiA2.KaiA2.KaiC6.KaiB6

```

Figure 67: The model of the circadian clock in cyanobacteria. There are three types of rules – (de)phosphorylation, complex formation/dissociation, and transcription. To simplify the notation, in section complexes, there are aliases defined for individual homo and hetero polymers. These can be used, for example, in phosphorylation rules as variable values, allowing a compact description of the same process on multiple complex variants.

three KaiC proteins are phosphorylated on the serine residue, the other two are phosphorylated on both residues, and the last one is not phosphorylated at all. The number of possible combinations is quite large, all depicted using a single rule rC6.

7.1.2 Fibroblast growth factor signalling pathway

We present a model of *fibroblast growth factor* (FGF) signalling pathway adopted from Yamada et al. model [YTYo4]. The model represents a cascade of signal transduction of Ras-MAPK related signalling molecules. Signalling pathways are typical for single points of failure, which means incorrect behaviour on a particular point in the cascade can influence the rest of the pathway. A schematic overview of the model is depicted in Figure 68.

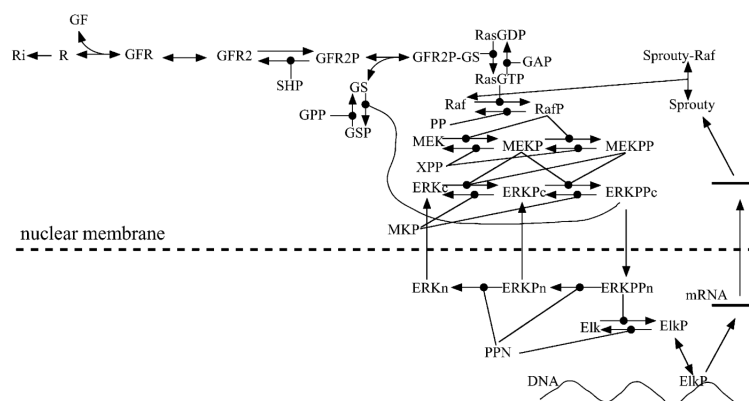


Figure 68: Schematic overview of Yamada et al. model taken from [YTYo4], depicting the common pathways between EGF and FGF. Enzymatic reactions are denoted by arrows with a dot symbol, connected by a line with the respective enzyme. Molecules with a P suffix represent phosphorylated forms, corresponding to the phosphorylation on a particular amino acid residue.

The entire model written in BCSL consists of twenty distinct agents interacting in over fifty rules. Most proteins can undergo phosphorylation (state change from u to p on some amino acid residues). Although the model size is quite significant, it is still negligible compared to the size of the underlying reactions network, which is over three times larger.

The signalling pathway starts with the ligand binding a pair of receptors, followed by their dimerisation, and consequent autophosphorylation, which activates protein kinase activity of the receptor. Phosphorylated tyrosine residues in the cytoplasmic domain of receptors then work as binding sites for several adaptor proteins, such as Grb2-SOS complex (GS), serving as phosphoryl group exchange protein for Ras. That consequently induces Raf phosphorylation. The phosphorylated Raf triggers MAPK cascade, resulting in MEK and ERK phosphorylation. Activated ERK is then translocated to the nucleus and serves as an activator of several transcription factors (e.g. Elk). It also phosphorylates SOS and inactivates it, creating an inhibitory feedback loop.

```

#! rules
FGF()::cyt + R()::cyt ⇔ FGF().R()::cyt
2 FGF().R()::cyt ⇔ FGF().R().FGF().R()::cyt
FGF(Thr(u)).R().FGF().R()::cyt ⇔ FGF(Thr(p)).R().FGF().R()::cyt
FRS(Thr(u)).FGF().R().FGF().R()::cyt ⇒ FRS(Thr(p)).FGF().R().FGF().R()::cyt
FRS(Thr(p)).FGF().R().FGF().R()::cyt ⇒ FRS(Thr(p))::cyt + FGF().R().FGF().R()::cyt
SHP()::cyt + FRS(Thr(p))::cyt ⇒ SHP().FRS(Thr(p))::cyt
FRS(Thr(p)).SHP()::cyt ⇒ FRS(Thr(u)).SHP()::cyt
FRS(Thr(u)).SHP()::cyt ⇒ FRS(Thr(u))::cyt + SHP()::cyt
GPP()::cyt + GS(Thr(p))::cyt ⇒ GPP().GS(Thr(p))::cyt
GS(Thr(p)).GPP()::cyt ⇒ GS(Thr(u)).GPP()::cyt
GS(Thr(u)).GPP()::cyt ⇒ GS(Thr(u))::cyt + GPP()::cyt
ERK(Tyr(p),Thr(p))::cyt + GS(Thr(u))::cyt ⇒ ERK(Tyr(p),Thr(p)).GS(Thr(u))::cyt
GS(Thr(u)).ERK()::cyt ⇒ GS(Thr(p)).ERK()::cyt
GS(Thr(p)).ERK()::cyt ⇒ GS(Thr(p))::cyt + ERK()::cyt
FRS(Thr(p),Tyr(u))::cyt + GS(Thr(u))::cyt ⇔ FRS(Thr(p),Tyr(u)).GS(Thr(u))::cyt
Ras(Thr(u)).FRS().GS()::cyt ⇒ Ras(Thr(p)).FRS().GS()::cyt
Ras(Thr(p)).FRS().GS()::cyt ⇒ Ras(Thr(p))::cyt + FRS().GS()::cyt
GAP()::cyt + Ras(Thr(p))::cyt ⇒ GAP().Ras(Thr(p))::cyt
Ras(Thr(p)).GAP()::cyt ⇒ Ras(Thr(u)).GAP()::cyt
Ras(Thr(u)).GAP()::cyt ⇒ Ras(Thr(u))::cyt + GAP()::cyt
Ras(Thr(p))::cyt + Raf(Thr(u))::cyt ⇒ Ras(Thr(p)).Raf(Thr(u))::cyt
Raf(Thr(u)).Ras()::cyt ⇒ Raf(Thr(p)).Ras()::cyt
Raf(Thr(p)).Ras()::cyt ⇒ Raf(Thr(p))::cyt + Ras()::cyt
PP()::cyt + Raf(Thr(p))::cyt ⇒ PP().Raf(Thr(p))::cyt
Raf(Thr(p)).PP()::cyt ⇒ Raf(Thr(u)).PP()::cyt
Raf(Thr(u)).PP()::cyt ⇒ Raf(Thr(u))::cyt + PP()::cyt
Raf(Thr(p))::cyt + MEK(Ser212(u))::cyt ⇒ Raf(Thr(p)).MEK(Ser212(u))::cyt
MEK(Ser212(u)).Raf()::cyt ⇒ MEK(Ser212(p)).Raf()::cyt
MEK(Ser212(p)).Raf()::cyt ⇒ MEK(Ser212(p))::cyt + Raf()::cyt
Raf(Thr(p))::cyt + MEK(Ser298(u))::cyt ⇒ Raf(Thr(p)).MEK(Ser298(u))::cyt
MEK(Ser298(u)).Raf()::cyt ⇒ MEK(Ser298(p)).Raf()::cyt
MEK(Ser298(p)).Raf()::cyt ⇒ MEK(Ser298(p))::cyt + Raf()::cyt
XPP()::cyt + MEK(Ser212(p))::cyt ⇒ XPP().MEK(Ser212(p))::cyt
MEK(Ser212(p)).XPP()::cyt ⇒ MEK(Ser212(u)).XPP()::cyt
MEK(Ser212(u)).XPP()::cyt ⇒ MEK(Ser212(u))::cyt + XPP()::cyt
XPP()::cyt + MEK(Ser298(p))::cyt ⇒ XPP().MEK(Ser298(p))::cyt
MEK(Ser298(p)).XPP()::cyt ⇒ MEK(Ser298(u)).XPP()::cyt
MEK(Ser298(u)).XPP()::cyt ⇒ MEK(Ser298(u))::cyt + XPP()::cyt
ERK(Thr(u)).MEK()::cyt ⇒ ERK(Thr(p)).MEK()::cyt
ERK(Thr(p)).MEK()::cyt ⇒ ERK(Thr(p))::cyt + MEK()::cyt
ERK(Tyr(u)).MEK()::cyt ⇒ ERK(Tyr(p)).MEK()::cyt
ERK(Tyr(p)).MEK()::cyt ⇒ ERK(Tyr(p))::cyt + MEK()::cyt
MKP()::cyt + ERK(Thr(p))::cyt ⇒ MKP().ERK(Thr(p))::cyt
ERK(Thr(p)).MKP()::cyt ⇒ ERK(Thr(u)).MKP()::cyt
ERK(Thr(u)).MKP()::cyt ⇒ ERK(Thr(u))::cyt + MKP()::cyt
MKP()::cyt + ERK(Tyr(p))::cyt ⇒ MKP().ERK(Tyr(p))::cyt
ERK(Tyr(p)).MKP()::cyt ⇒ ERK(Tyr(u)).MKP()::cyt
ERK(Tyr(u)).MKP()::cyt ⇒ ERK(Tyr(u))::cyt + MKP()::cyt
FRS(Tyr(u)).ERK()::cyt ⇒ FRS(Tyr(p)).ERK()::cyt
FRS(Thr(u),Tyr(p)).ERK()::cyt ⇒ FRS(Thr(u),Tyr(p))::cyt + ERK()::cyt
FRS(Tyr(p))::cyt ⇒ FRS(Tyr(u))::cyt
ERK(Thr(u))::cyt + MEK(Ser212(p),Ser298(p))::cyt ⇒ ERK(Thr(u)).MEK(Ser212(p),Ser298(p))::cyt
Ras(Thr(u))::cyt + FRS(Thr(p),Tyr(u)).GS(Thr(u))::cyt ⇒ 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 ⇒ FRS(Thr(u)).FGF(Thr(p)).R().FGF(Thr(p)).R()::cyt
ERK(Thr(u))::cyt + MEK(Ser212(p),Ser298(p))::cyt ⇒ ERK(Tyr(u)).MEK(Ser212(p),Ser298(p))::cyt
FRS(Tyr(u))::cyt + ERK(Tyr(p),Thr(p))::cyt ⇒ FRS(Tyr(u)).ERK(Tyr(p),Thr(p))::cyt

```

Figure 69: Complete set of rules of BCSL adaptation of Model Yamada et al.

7.1.3 Tumour growth

Tumour growth is based on unrestricted cell division (*mitosis*). The cell cycle is the process between two mitoses, and it consists of four phases: the resting phase G_1 , the DNA replication phase S , the resting phase G_2 , and the mitosis phase M in which the cells segregate the duplicated sets of chromosomes between daughter cells. The three phases G_1 , S , and G_2 constitute the pre-mitotic phase, also called *interphase*.

We have adopted the model of tumour growth [VR03] to our language. It considers two populations of tumour cells: those in interphase and those in mitosis. We represent the tumour cell as an agent T . The current phase is expressed with an atom phase in its compo-

sition, which can have two different states – *i* for interphase and *m* for mitosis. For simplicity, we omit the compartment from the rules since it does not change and plays no critical role in this model.

```

#! rules
T(P{i}) ⇒ T(P{m}) @ α1 × [T(P{i})]
T(P{m}) ⇒ 2 T(P{i}) @ α2 × [T(P{m})]
T(P{i}) ⇒ @ d1 × [T(P{i})]
T(P{m}) ⇒ @ d2 × [T(P{m})]

#! inits
1 T(P{i})

#! definitions
α2 = 0.5
d1 = 0.3

```

Figure 70: Rules of the tumour growth model. The first rule describes the change of the phase of a cell from interphase to mitosis. The second rule describes the duplication of the cell into two daughter cells. Note that both start in interphase. The last two rules describe the death of cells in both possible states.

The rules of the model are available in [Figure 70](#). Note that this model is a demonstration where all rules are *reaction-based*, i.e. they do not describe an abstract rule, only modification of concrete agents.

Additionally, the rules have defined rate functions, and some of them are parametrised by undefined values. Parameters α_1 and α_2 are present in rules responsible for change of phase and cell division, while parameters d_1 and d_2 are in the rules where the cell *dies*. The values $\alpha_2 = 0.5$ and $d_1 = 0.3$ are constant, and the values of the other two parameters are not specified. For the *initial state*, we assume a single agent $T(P\{i\})$.

The presence of rate functions causes the transition system to be probabilistic and also allows simulating the model on a qualitative level ([Figure 71](#)). In particular, the model gives rise to infinite pMC since the second rule can *generate* additional agents. To obtain a finite abstract probabilistic model, we have heuristically limited the number of states of the model. Particularly, we generate all the states having the number of individuals of both species less or equal to 5, and we introduce a special abstract state (referred to as *hell*) which represents all the other states, which limits the size of possible state space to 6^2 . This approximation is incorrect only in cases when one wants to reach a state which is represented by the special state.

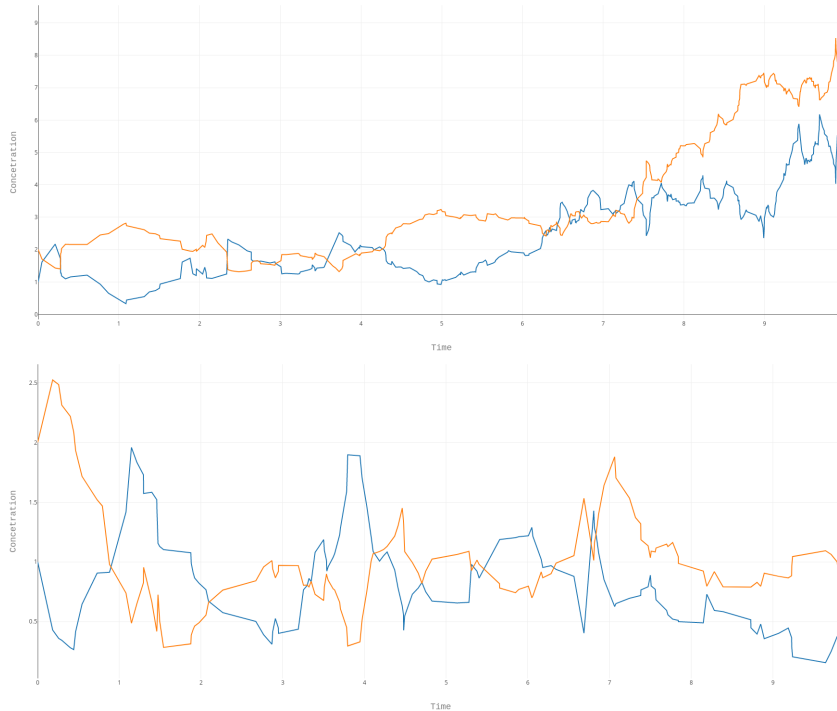


Figure 71: Visualisation of stochastic simulation of tumour growth model under different parametrisations. The cells in interphase are depicted in orange, while cells in mitosis are in blue. The top picture shows the simulation of the model with favourable conditions for tumour growth (with values $a_1 = 2.1$, $d_2 = 0.05$) and the bottom picture shows a different scenario, where the speed of tumour cells divisions and apoptosis is approximately the same (with values $a_1 = 1$, $d_2 = 0.35$), resulting in the stagnating population of cells. Displayed simulations are average of five separate runs.

7.2 PARAMETER SYNTHESIS

Parameter synthesis is a method to investigate parametrised models lacking some detailed quantitative information. We demonstrate this technique on two models partially presented in the previous section. In particular, we investigate a fragment of the circadian clock with partially known quantitative aspects and the tumour growth model, where we already learned about the unknown parameter values and briefly observed the sensitivity of the model on their values.

7.2.1 Circadian clock

In this example, we demonstrate parameter synthesis on a case study from the biological domain. Miyoshi et al. [Miy+07] is an ODE model describing circadian rhythms in cyanobacteria. The model reproduces a robust KaiC phosphorylation cycle in the absence of *de novo* gene

expression, as is observed *in vitro*, as well as its coupling to transcriptional/translational feedback in continuous light conditions *in vivo*. We have adopted this model to our rule-based formalism in a simplified form. The model covers a fragment of the circadian clock mechanism as described in [Section 7.1.1](#), but for this part provides quantitative information that is lacking for the whole clock. Therefore it allows performing different types of analysis, including parameter synthesis.

```

#! rules
KaiC(S{u},T{u}).KaiC(S{u},T{u})::cyt  $\Rightarrow$  KaiC(S{p},T{u}).KaiC(S{p},T{u})::cyt
@ (kcat1  $\times$  [KaiA2()]::cyt)  $\times$  [KaiC(S{u},T{u}).KaiC(S{u},T{u})::cyt] /
(Km + [KaiC(S{u},T{u}).KaiC(S{u},T{u})::cyt])
KaiC(S{p},T{u}).KaiC(S{p},T{u})::cyt  $\Rightarrow$  KaiC(S{u},T{u}).KaiC(S{u},T{u})::cyt
@ (kcat2  $\times$  [KaiB4{a}.KaiA2()]::cyt)  $\times$  [KaiC(S{p},T{u}).KaiC(S{p},T{u})::cyt] /
(Km + [KaiC(S{p},T{u}).KaiC(S{p},T{u})::cyt])
KaiC(S{p},T{u}).KaiC(S{p},T{u})::cyt  $\Rightarrow$  KaiC(S{p},T{p}).KaiC(S{p},T{p})::cyt
@ (kcat3  $\times$  [KaiA2()]::cyt)  $\times$  [KaiC(S{p},T{u}).KaiC(S{p},T{u})::cyt] /
(Km + [KaiC(S{p},T{u}).KaiC(S{p},T{u})::cyt])
KaiC(S{p},T{p}).KaiC(S{p},T{p})::cyt  $\Rightarrow$  KaiC(S{p},T{u}).KaiC(S{p},T{u})::cyt
@ (kcat4  $\times$  [KaiB4{a}.KaiA2()]::cyt)  $\times$  [KaiC(S{p},T{p}).KaiC(S{p},T{p})::cyt] /
(Km + [KaiC(S{p},T{p}).KaiC(S{p},T{p})::cyt])
KaiB4{i}::cyt  $\Rightarrow$  KaiB4{a}::cyt
@ (kcatb2  $\times$  [KaiB4{i}::cyt]) / (Kmb2 + [KaiB4{i}::cyt])
KaiB4{a}::cyt  $\Rightarrow$  KaiB4{i}::cyt
@ (kcatb1  $\times$  [KaiB4{a}::cyt]) / (Kmb1 + [KaiB4{a}::cyt])
KaiC().KaiC()::cyt  $\Rightarrow$  2 KaiC()::cyt @ kdimer  $\times$  [KaiC().KaiC()::cyt]
2 KaiC()::cyt  $\Rightarrow$  KaiC().KaiC()::cyt @ kdimer  $\times$  [KaiC()::cyt]  $\times$  ([KaiC()::cyt] - 1)

#! definitions
Km = 0.602
kcatb2 = 0.346
kcatb1 = 0.602
Kmb2 = 66.75
Kmb1 = 2.423
k11 = 0.0008756
kdimer = 1.77

```

Figure 72: Miyoshi et al. model in BCSL. The first four rules are responsible for the change in the phosphorylation level of KaiC dimers. The rate functions of these rules represent enzymatic laws and are dependent on the current numbers of KaiA dimers and KaiB tetramers. The next two rules change the activity level of the KaiB4 complex, and the last two rules form and disassembly the KaiC dimer. The exact meaning of individual constants and parameters is described in [\[Miy+07\]](#).

The same as in the whole circadian clock system, the core of this model is formed by three main proteins – KaiA, KaiB, and KaiC. The protein KaiC has two phosphorylation sites (S – serine and T – threonine), both of which can be either phosphorylated or unphosphorylated. The simplification is applied to the formation complexes – two KaiC proteins can form a homo-dimer (instead of using hexamers; the goal is to decrease the combinatorial explosion).

Protein KaiA can also form a homo-dimer and act as a kinase for the phosphorylation of KaiC dimers. Since the KaiA dimer cannot un-

dergo any modification, we simplify it to a single agent. Protein KaiB can form a homo-tetramer, which can be either active or inactive as a whole. For this reason and, again, for simplicity, we model it as a single agent.

The KaiA dimer has a positive enzymatic effect on the phosphorylation of KaiC dimers. On the other hand, active KaiB tetramer then serves as an inhibitor of KaiC dimer phosphorylation, i.e. it enhances its dephosphorylation. This is done such that it forms a complex with KaiA dimer and inhibits its phosphorylation efforts.

- *initial state:*

```
#! inits
2 KaiC(S{u},T{u})::cyt
1 KaiB4{a}::cyt
1 KaiA2()::cyt
```

- *property of interest:*

$$\Pi_{\geq 0.99}(\mathbf{F} [\text{KaiC}(\text{S}\{\text{p}\}, \text{T}\{\text{p}\}) . \text{KaiC}(\text{S}\{\text{p}\}, \text{T}\{\text{p}\}) :: \text{cyt}] > 0)$$

- *parameter values:*

$$k_{\text{cat}_2} = 0.539, k_{\text{enz}} = 8.756 \times 10^{-4}, k_{\text{cat}_4} = 0.89$$

- *parameter ranges:*

$$k_{\text{cat}_3} \in [0, 2], k_{\text{cat}_1} \in [0, 1]$$

- *additional rule for construction of KaiA dimer and KaiB4 tetramer complex:*

$$\text{KaiB4}\{a\}::\text{cyt} + \text{KaiA2}()::\text{cyt} \Rightarrow \text{KaiB4}\{a\}.\text{KaiA2}()::\text{cyt}$$

$$@ \text{ k11} \times [\text{KaiB4}\{a\}::\text{cyt}] \times [\text{KaiA2}()::\text{cyt}]$$

Figure 73: Settings for phosphorylation experiment of Miyoshi et al. model. The goal of the experiment is to find parametrisations such that the model reaches the fully phosphorylated level of KaiC dimer. The model is extended by a rule for the construction of a complex, possibly disabling the phosphorylation.

The rules of the model are available in [Figure 72](#). The mechanism of phosphorylation and activation causes the model to have an oscillatory behaviour. For our simplified case, we investigate whether the probability of reaching the phosphorylated KaiC dimer, followed by reaching the unphosphorylated dimer, is close to one.

We assume two different experiments, both having different initial conditions, one additional rule for manipulation of KaiA and KaiB interaction, different unknown parameters, and finally, a different property of interest. The first experiment ([Figure 73](#)) expresses conditions with an unphosphorylated KaiC dimer and the property of reaching

the phosphorylated KaiC dimer. For the second experiment (Figure 74), it is the other way around. The probability for both properties should be close to one since the oscillation should always be present.

- initial state:

```
#! inits
2 KaiC(S{p},T{p})::cyt
1 KaiB4{a}::cyt
1 KaiA2()::cyt
```

- property of interest:

$$\Pi_{\geq 0.99}(\mathbf{F} [\text{KaiC}(\text{S}\{u\},\text{T}\{u\}) . \text{KaiC}(\text{S}\{u\},\text{T}\{u\})::\text{cyt}] > 0)$$

- parameter values:

$$k_{\text{cat}_1} = 0.539, k_{\text{enz}} = 8.756 \times 10^{-4}, k_{\text{cat}_3} = 1.079$$

- parameter ranges:

$$k_{\text{cat}_2} \in [0, 1], k_{\text{cat}_4} \in [0, 2]$$

- *additional rule* for disassembly of KaiA dimer and KaiB₄ tetramer complex:

$$\text{KaiB}_4\{a\} . \text{KaiA}_2()::\text{cyt} \Rightarrow \text{KaiB}_4\{a\}::\text{cyt} + \text{KaiA}_2()::\text{cyt}$$

$$@ \text{ k12} \times [\text{KaiB}_4\{a\} . \text{KaiA}_2()::\text{cyt}]$$

Figure 74: Settings for dephosphorylation experiment of Miyoshi et al. model. The experiment is focused on the dephosphorylation of KaiC dimer enabled by an additional rule for enzymatic complex disassembly.

In Figure 75, there is a visualisation of parameter synthesis for both experiments. The results of the first experiment show that the property is almost always satisfied except for some marginal cases when the parameter values are close to zero. This fact is in agreement with the global robustness degree, which is approximately 0.995. In the second experiment, the property was satisfied in a smaller fraction of parameter space, caused by different initial conditions and the additional rule. However, this difference is very insignificant, which confirms the robustness degree with a value of approximately 0.98. These results confirm that the behaviour of the model is very robust to perturbation of parameters directly responsible for phosphorylation activity, thus showing the oscillatory behaviour is very persistent.

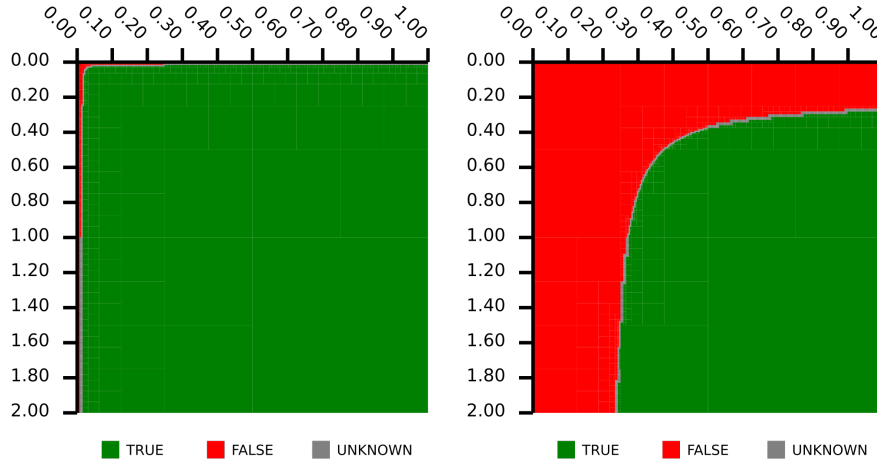


Figure 75: Visualisation of results of parameter synthesis for the Miyoshi model. The *left* picture depicts the results of the phosphorylation experiment (Figure 73). The horizontal axis represents values of the parameter $kcat_1 \in [0, 1]$ and the vertical axis represents values of the parameter $kcat_3 \in [0, 2]$. The *right* picture depicts the results of the dephosphorylation experiment (Figure 74). The horizontal axis represents values of the parameter $kcat_2 \in [0, 1]$ and the vertical axis represents values of the parameter $kcat_4 \in [0, 2]$.

7.2.2 Tumour growth

In Section 7.1.3, we introduced the tumour growth model with two parameters with unknown values. On two simulations from Figure 71 we can see that the behaviour of the model depends on the values of these parameters. Therefore we further investigate the dependency on these parameters.

We are interested in the property of whether the population of tumour cells will reach almost its maximum with a probability higher than 0.5, meaning that the growth is not random but has rather a tendency to grow without limitations. This property can be expressed as $\phi = \Pi_{\geq 0.5}(\mathbf{F} \, T() > 8)$. In Figure 76, there is a visualisation of parameter synthesis. For parameters, we assumed admissible ranges $a_1 \in [0; 3]$ and $d_2 \in [0.001; 0.5]$. The results show that the higher values of the parameter a_1 (cell division) and the lower values of the parameter d_2 increase the probability of property satisfaction. This result is quite expected because both parameters directly influence cell division (a_1) and degradation (d_2) of cells. We have also computed the global robustness degree of the property, which is approximately 0.24. In this case, it can be interpreted as 24% of parameter space satisfies the property $\mathbf{F} \, T() > 8$.

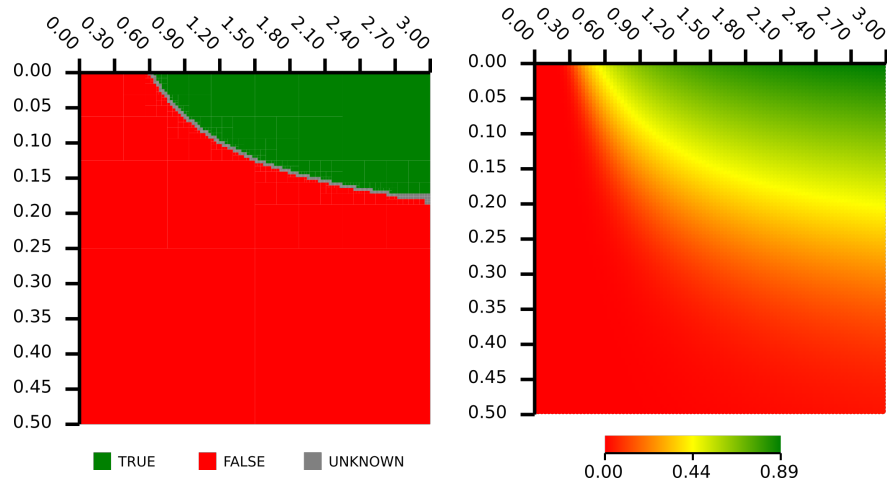


Figure 76: Visualisation of results of parameter synthesis (*left*) and quantitative model checking using sampling (*right*) for property ϕ for the tumour growth model. The horizontal axis represents values of the parameter $a_1 \in [0, 3]$, and the vertical axis represents values of the parameter $d_2 \in [0.001, 0.5]$. The probability threshold 0.5 from the property ϕ is visible in both sampling (approximately the yellow line) and parameter synthesis (the grey line). It shows that the parameter synthesis method gives us a very precise result and is in agreement with quantitative model checking.

7.3 STATIC ANALYSIS

We want to demonstrate the practical purposes of static analysis described in [Section 5.5](#). For this purpose, we use our adapted version of the Yamada et al. model described in [Section 7.1.2](#). We consider initial conditions such that there are all necessary agents in one or two repetitions (when multiple agents are required to create complexes, e.g. FGF). In such a case, the number of reachable states can grow up to 2^{72} , which is too high to be effectively enumerated. In [Figure 77](#), there is a fragment of the model with a focus on the parts used in this analysis. In particular, we want to check whether given complex:

$$\text{FRS}(\text{Thr}\{u\}, \text{Tyr}\{u\}) . \text{FGF}(\text{Thr}\{u\}) . \text{R}() . \text{FGF}(\text{Thr}\{u\}) . \text{R}() :: \text{cyt}$$

is reachable in the model. The agent is formed from FGF proteins which are unphosphorylated (u) on threonine residues (Thr). With the traditional approach, we have to enumerate the entire transition system of the model and then use the model checking method to check it. In our case, we can check if it is non-reachable using *static reachability analysis* ([Theorem 15](#)). The conclusion is that there is no compatible agent on any right-hand side of the rules. Generally, it is not possible to create a complex from FRS and unphosphorylated (u) FGF proteins. It means that the given complex is non-reachable.

Demonstration of *context-based reduction* ([Theorem 14](#)) is provided on the same model as in the previous case. We can compute with the

```

#! rules
r1 ~ FGF() + R() ⇔ FGF().R()
r2 ~ 2 FGF().R() ⇔ FGF().R().FGF().R()
r3 ~ FGF(Thr{u}).R().FGF().R() ⇔ FGF(Thr{p}).R().FGF().R()
r4 ~ FRS(Thr{u}) + FGF(Thr{p}).R().FGF(Thr{p}).R() ⇒
    FRS(Thr{u}).FGF(Thr{p}).R().FGF(Thr{p}).R()
r5 ~ FRS(Thr{u}).FGF().R().FGF().R() ⇒ FRS(Thr{p}).FGF().R().FGF().R()
:

#! inits
2 FGF(Thr{u})
2 R()
1 FRS(Thr{u},Tyr{u})
:

```

Figure 77: A fragment of Yamada et al. model [YTYo4] of FGF signalling pathway written in BCSL. All agents reside in a cytosol *cyt* compartment, which is omitted for simplicity. The rule *r4* requires both threonine residues (Thr) on FGF proteins to be phosphorylated (p). The full model is available in Figure 69.

entire model since we will reduce its context to the minimum. Applying the reduction, there are created 16 bidirectional rules (Figure 78). The size of the transition system has significantly decreased – it has approximately six hundred states and two thousand edges.

```

#! rules
FGF() + R() ⇔ FGF().R()
FGF().R() + FGF().R() ⇔ FGF().R().FGF().R()
FGF().R().FGF().R() + FRS() ⇔ FGF().R().FGF().R().FRS()
FRS() + SHP() ⇔ FRS().SHP()
GS() + GPP() ⇔ GS().GPP()
GS() + ERK() ⇔ GS().ERK()
FRS() + GS() ⇔ FRS().GS()
FRS().GS() + Ras() ⇔ FRS().GS().Ras()
GAP() + Ras() ⇔ GAP().Ras()
Ras() + Raf() ⇔ Ras().Raf()
PP() + Raf() ⇔ PP().Raf()
Raf() + MEK() ⇔ Raf().MEK()
XPP() + MEK() ⇔ XPP().MEK()
MEK() + ERK() ⇔ MEK().ERK()
MKP() + ERK() ⇔ MKP().ERK()
ERK() + FRS() ⇔ ERK().FRS()

```

Figure 78: Yamada et al. model [YTYo4] after context-based reduction was applied. All agents reside in a cytosol compartment, which is omitted for simplicity. The original model is available in Figure 69.

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

7.4 REGULATIONS

We demonstrate the usage of regulations in the context of BCSL on typical patterns often present in biochemical phenomena. In particular, we provide examples of models of the MAPK pathway with applied conditional regulation, cell cycle with programmed regulation, methane combustion with concurrent-free regulation, and circadian clock with regular regulation. Then, we show on a more detailed case study (transport and metabolism of glucose and alternative sugars) a typical scenario of model development in biology and how regulations can assist in this process.

7.4.1 MAPK pathway

Modelling of signalling pathways [Han17; Khoo06; KFL12] comprises complex networks of signal cascades, amplified by numerous positive (activation) and negative (inhibition) feedbacks. While on a quantitative level, such interactions can be covered using enzymatic kinetics [MM+13], on a qualitative level modelling such feedbacks without detailed knowledge of their mechanisms is often difficult. In this case study, we show how to capture the feedback explicitly using regulations.

In particular, this example demonstrates how *conditional* regulation can be used to model inhibition in signalling pathways. The inhibition specification is separated from model rules, and model variants with and without this regulation can be easily compared. This allows us to model the inhibition abstractly without knowing the particular inhibition mechanism, which is often unknown in the early stages of signalling pathway research. Note that the activation can be easily modelled without the use of regulation by placing the enzyme on both sides of a rule – it has to be present but is not modified.

We demonstrate this approach on a fragment of MAPK/ERK signalling pathway [Pea+01], responsible for signal transduction from a receptor on the surface of the cell to the DNA in the nucleus of the cell. MAPK cascade contains three interconnected cycles of MAP kinase (MAPK), MAPK kinase (MAPKK), and MAPKK kinase (MAPKKK). The scheme of the fragment is available in [Figure 79](#).

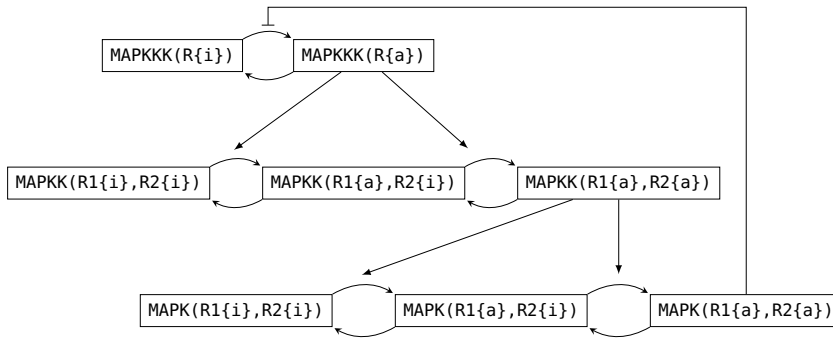


Figure 79: Scheme of MAPK signalling pathway. A fully activated form of the kinase from the upper level has a positive effect on the activation of the kinase from the lower level. The fully activated kinase from the lowest level has a negative effect on the activation of the kinase from the top level.

The specification of the model in BCSL is available in [Figure 80](#). A rule activating respective kinase is labelled by $r\langle i \rangle k$ (e.g. $r2k$ for MAPKK) with a suffix identifying the domain (1 or 2 if applicable). A suffix identifies the forward (fw) and backward (bw) direction of the interaction. For the activation of MAPKK and MAPKKK, the fully activated kinase from the upper cascade level is required. Finally, the conditional regulation models the inhibition by ensuring that rule $r3k_fw$ (activation of MAPKKK) is not enabled when $MAPK(R1\{a\}, R2\{a\})$ is present. Assuming no additional details are known about this interaction, the regulation serves its purpose and captures the desired effects.

The introduction of regulation results in the elimination of some otherwise possible transitions. This, in consequence, means that when MAPKKK is deactivated, and there is an active form of MAPK present, then activation of MAPKKK is not possible. To enable it, first MAPK needs to be deactivated. In terms of biological effects, the active form of MAPK inhibits the activation of MAPKKK.

To model the inhibition without regulation, one would need a detailed knowledge of the inhibitory mechanism. Without that, this could be done, for example, by requiring partially inactive MAPK molecule to be present in the process of MAPKKK activation, and consequently enumerating all options based on possible variants of such MAPK. This, however, becomes extremely laborious by increasing the molecule complexity and the number of inhibitions in the model. Additionally, it does not correspond to biological reality. The advantage of employing the regulation is that the particular mechanisms can be abstracted, yet its effects are reflected in the behaviour of the model. Moreover, it provides an elegant and more concise way to describe such a mechanism.

```

#! rules
# MAPKKK activation
r3k_fw ~ MAPKKK(R{i}) => MAPKKK(R{a})
r3k_bw ~ MAPKKK(R{a}) => MAPKKK(R{i})

# MAPKK activation
r2k1_fw ~ MAPKK(R1{i}) + MAPKKK(R{a}) => MAPKK(R1{a}) + MAPKKK(R{a})
r2k1_bw ~ MAPKK(R1{a}) => MAPKK(R1{i})

r2k2_fw ~ MAPKK(R2{i}) + MAPKKK(R{a}) => MAPKK(R2{a}) + MAPKKK(R{a})
r2k2_bw ~ MAPKK(R2{a}) => MAPKK(R2{i})

# MAPK activation
r1k1_fw ~ MAPK(R1{i}) + MAPKK(R1{a},R2{a}) =>
    => MAPK(R1{a}) + MAPKK(R1{a},R2{a})
r1k1_bw ~ MAPK(R1{a}) => MAPK(R1{i})

r1k2_fw ~ MAPK(R2{i}) + MAPKK(R1{a},R2{a}) =>
    => MAPK(R2{a}) + MAPKK(R1{a},R2{a})
r1k2_bw ~ MAPK(R2{a}) => MAPK(R2{i})

#! inits
1 MAPKKK(R{i})
1 MAPKK(R1{i},R2{i})
1 MAPK(R1{i},R2{i})

#! regulation
type conditional
r3k_fw: { MAPK(R1{a},R2{a}) }

```

Figure 8o: MAPK signalling pathway is represented by a cascade of signal transduction by domain activation of individual MAP kinases. In rules for MAPK and MAPKK activation (change of state from *i* to *a*), the fully activated protein from the upper layer is present (but not modified by the rule), modelling the activation. In the initial state of the system, there are only inactive forms of all proteins. Finally, a conditional regulation is defined for rule *r3k* (only its forward variant, thus *_fw* suffix), which makes sure the rule is not executed when fully activated form of MAPK is present in the state (inhibition).

7.4.2 Cell cycle

The reproduction of cells is controlled by a regulatory network of reactions known as cell cycle [TNo1; TCNNo2]. The cycle consists of four phases. During synthesis (S) phase, the cell replicates all of its components and divides them between the two daughter cells at the end of the mitosis (M) phase. After the S phase, there is gap (G2) phase, where the cell ensures that the duplication of DNA has been completed. The daughter cells are not immediately replicated again, instead they are located at another gap (G1) phase.

In this example, we use a model of the cell cycle as explained in [BO14]. It models several checkpoints in residing cell cycle phases. We adopt DNA and cell size checkpoints (Figure 81). While many studies focus on the mechanistic modelling of the cell cycle [TCNNo2; KNS14; HSH13; MRU11], in this case, we focus on cell response to a checkpoint failure in a population model of the cell cycle. The cell response can be a complex process, comprising a plethora of signalling molecules involved in many chemical reactions. Employing a regulation enables, for example, easily capturing programmed death on a qualitative level. The model uses *programmed* regulation to influence cell response to a detected checkpoint failure, forcing the cell to programmed death. Additionally, this mechanism can be easily turned off, allowing the comparison of model variants.

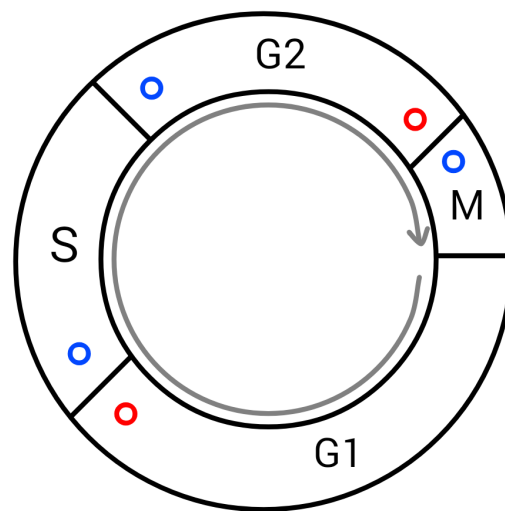


Figure 81: Scheme of the cell cycle. The checkpoints for cell size are marked by red colour in phases G1 and G2 and for DNA damage by blue colour in phases S, G2, and M.

A detailed description of the model is available in Figure 82. Rules r3 to r5 are responsible for phase transition (represented by compartment) from G1 through S, G2, and finally to M. Rule r6 models cell division (mitosis, thus phase M), where a cell forms two daughter cells, starting again in the G1 phase. Rule r7 can be applied to any cell and represents cell death. Finally, rules r1 and r2 model cell size and DNA damage checkpoints, respectively. The state of the respective check can be changed from ok to nok in specified phases (as indicated in Figure 81), representing the fact that the cell detected a respective issue. The question mark ? is used as a wildcard (*variable*) for multiple applicable compartments.

A model run with healthy behaviour is getting through all cycle phases and dividing the cell in mitosis (M). When a cell divides, the daughter cells enter the G1 phase and inherit all attributes of the

mother cell. This includes the state of `size` or `dna`. If any issues with `size` or `dna` are detected (indicated by change from `ok` to `nok` state), the cell should be immediately disposed. In other words, a damaged cell should never divide. This is achieved by used programmed regulation. It ensures that when rule `r1` or `r2` is used, the rule `r7` for cell death is used in the next step. Therefore, no damaged cell is allowed to divide because immediately after the checkpoint failure is detected, it is disposed of by applying rule `r7`.

```

#! rules
# cell size checkpoint
r1 ~ cell(size{ok})::? ⇒ cell(size{nok})::? ; ? = {G1, G2}
# DNA damage checkpoint
r2 ~ cell(dna{ok})::? ⇒ cell(dna{nok})::? ; ? = {G2, S, M}
# phases transition
r3 ~ cell()::G1 ⇒ cell()::S
r4 ~ cell()::S ⇒ cell()::G2
r5 ~ cell()::G2 ⇒ cell()::M
# cell division
r6 ~ cell()::M ⇒ 2 cell()::G1
# cell death
r7 ~ cell()::? ⇒

#! inits
1 cell(dna{ok}, size{ok})::G1

#! regulation
type programmed
r1: {r7}
r2: {r7}

```

Figure 82: The model of the cell cycle, where a single agent `cell` can be present in several compartments, modelling individual cell cycle phases. Moreover, it has two features, `size` and `dna`, both in two possible states `ok` and `nok`, modelling cell size and DNA damage checkpoints, respectively. Once one of the checks does not pass, and the state is changed to `nok`, the *programmed* regulation ensures that the next following rule is `r7`, modelling the cell death. As a process can often be executed in multiple locations (or cell phases in this case), a variable `?` is used to simplify the notation. Please note that for all the other rules, a complete set of rules is allowed as successors (omitted for conciseness).

Let us briefly discuss modelling the system without the use of regulations. There is no mechanism involved which would enforce the programmed death to be executed immediately when an error is detected. The cell would have the option to die, but it could still choose

to do another action and eventually even divide. Similarly to the previous example, such behaviour could be, for example, achieved by introducing an artificial internal state which would switch off when a checkpoint failure is detected and required to be on for all phase transition rules. Alternatively, it could be achieved using already present internal states, but this would lead to a less concise and transparent description.

7.4.3 Methane combustion

In this case study, we use a simple chemical system of methane combustion from [Figure 83](#) to demonstrate the usage of *concurrent-free* regulation. There are two types of combustion – *complete* combustion occurs when there is enough oxygen in the environment, producing carbon dioxide. When there is a lack of oxygen, *incomplete* combustion occurs, producing carbon monoxide. For the sake of this example, let us assume a simplified case where, with enough oxygen, the combustion is always complete. The incomplete combustion is allowed only when there is not enough oxygen for the complete combustion.

Describing such a system using chemical reactions from [Figure 83](#) (ignoring the regulation for the moment) lacks a mechanism ensuring the priority of complete combustion. That means whenever the reaction complete is enabled (there is enough oxygen), the reaction incomplete is possible, too (there are inherently fewer reactants required). The model allows both reactions to compete for oxygen, which does not meet our modelling intentions.

```

#! rules
complete  ~ 2 CH4() + 4 O2() ⇒ 2 CO2() + 4 H2O()
incomplete ~ 2 CH4() + 3 O2() ⇒ 2 CO() + 4 H2O()

#! regulation
type concurrent-free
(complete, incomplete)

```

Figure 83: The methane combustion model described by two chemical reactions, representing complete and incomplete combustion. Concurrent-free regulation is specified by a pair of rule labels, prioritising the first rule in the pair.

To meet the intentions can often lead to the introduction of additional molecules or processes, making the model less readable, extensive, and inaccurate. Moreover, there are cases where the traditional mechanistic approach is not expressive enough to capture the

desired behaviour (as in the case of our example, assuming arbitrary initial conditions).

To tackle this issue, we apply a regulation that influences the rule for incomplete combustion by assigning priority to the rule for complete combustion. Since the two rules are concurrent (i.e. they share reactants), this can be achieved by concurrent-free regulation by assigning priority to rule `complete` over rule `incomplete`. The result is that whenever both rules are enabled, only `complete` one is used, meeting our modelling intentions. The details of the used regulation are available in [Figure 83](#).

7.4.4 Circadian clock

In this example, we demonstrate the usage of *regular* regulation on the model of the circadian clock in cyanobacteria described in [Section 7.1.1](#). The behaviour of the model is very complex. However, while most of the rules can be used at any point, there is a typical scenario that is more likely to happen (red circle highlighted in [Figure 66](#)). In particular, KaiC hexamer is phosphorylated by KaiAs and then dephosphorylated by the additional presence of KaiBs in the protein complex. Nonetheless, the exact numerical valuation of this probability is not known and is generally difficult to obtain. Still, we would like to reduce the behaviour in the model in this manner and examine the resulting behaviour. We can represent unknown quantitative information with a regulation. For that, we use *regular* regulation given by a regular expression (RE) over rules, allowing us to track the most likely behavioural patterns.

```

#! regulation
type regular
#      1      2      3      4
{{rfA,rfBC}}* {rC6_fw,rA2_fw}* rA2C6_fw rA4C6_fw {rpS_fw,rpT_fw}*

#      5      6      7
rA4B6C6_fw rA6B6C6_1_fw rA6B6C6_2_bw {rpS_bw,rpT_bw}* rB6C6_bw rC6_bw* }*

```

Figure 84: A regular expression describing a typical scenario in the model of the circadian clock. For better readability, we split the expression into two lines and annotated each part, each having a specific purpose: (1) formation of Kai proteins, (2) association of homopolymers, (3) association of hetero-polymers of KaiC and KaiA, (4) phosphorylation of KaiC and KaiA complex, (5) attachment of KaiBs, (6) dephosphorylation induced by KaiBs, and (7) complexes disassembly.

The details of employed RE are available in [Figure 84](#). In this RE, we first allow the transcription to take place. This creates several copies of monomers, which can be further used in the formation of homopolymers (KaiC hexamer and KaiA dimer in particular). After that, the

protein complexes can enter the phosphorylation cycle. This starts with the formation of hetero complexes of KaiC and KaiA, promoting the phosphorylation phase. Then, KaiB proteins are attached to the complex, rendering the KaiA phosphorylation activity ineffective and enabling auto-dephosphorylation of KaiC. Finally, all complex parts are disassembled. Please note that both phosphorylation and dephosphorylation phases are not mandatory in the context of a single model run. The respective rules can be applied an arbitrary number of times, including zero. The infinite suffix of empty rules is omitted from RE.

Compared to a typical scenario described by the RE, the model exhibits uncontrolled nondeterminism in the unregulated case, which leads to unfeasible behaviour due to combinatorial explosion. This is usually handled by simulating the model with associated quantitative aspects [Cov14], e.g. by assigning rate laws to the rules, which are not known in this case and are generally difficult to obtain.

7.4.5 Metabolism of sugars in *E. Coli*

We provide a more detailed case study to demonstrate the applicability of regulations on a real-world example. We describe the transport and metabolism of glucose and alternative sugars (for demonstration, we use fructose) in *Escherichia Coli*. An interesting phenomenon of preferential glucose consumption over other carbon sources can be observed.

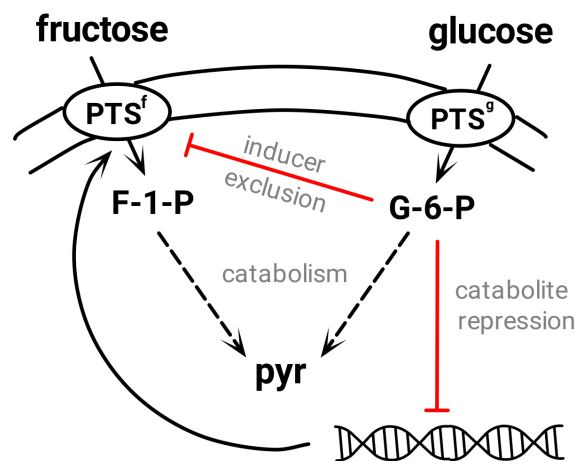


Figure 85: Schematic representation of sugars metabolism in *E. Coli* for the *simple* version with limited knowledge. The dashed arrows represent simplified rules (neglecting irrelevant details to minimise the model size), while the red arrows are involved in the usage of regulations.

The phosphotransferase system (PTS), responsible for the transport and phosphorylation of sugars, plays a crucial role in this mechanism by employing catabolite repression [AOB13] (the process of metabolising one carbon source inhibits the synthesis of enzymes involved in the catabolism of secondary carbon source) and inducer exclusion [Hog+98] (disable membrane proteins involved in the transport of secondary carbon sources by direct binding them). While the current knowledge [Esc+12] about these mechanisms is quite detailed, it was very limited in the early stages of the research [CH76; HD82].

```

#! rules
# fructose
r1 ~ fructose{n}::ext + PTS_f()::mem ⇔ fructose{n}.PTS_f()::mem
r2 ~ fructose{n}.PTS_f()::mem ⇒ fructose{1P}::cyt + PTS_f()::mem

# glucose
r3 ~ glucose{n}::ext + PTS_g()::mem ⇔ glucose{n}.PTS_g()::mem
r4 ~ glucose{n}.PTS_g()::mem ⇒ glucose{6P}::cyt + PTS_g()::mem

# catabolism
r5 ~ glucose{6P}::cyt ⇒ pyruvate()::cyt
r6 ~ fructose{1P}::cyt ⇒ pyruvate()::cyt

# DNA
r7 ~ DNA()::nuc ⇒ DNA()::nuc + PTS_f()::cyt
r8 ~ PTS_f()::cyt ⇒ PTS_f()::mem

#! inits
1 DNA()::nuc
1 fructose{n}::ext
1 glucose{n}::ext
1 PTS_g()::mem

#! regulation
type conditional
r2: {glucose{6P}::cyt}
r7: {glucose{6P}::cyt}

```

Figure 86: The *simple* variant of the sugar metabolism model, schematically available in Figure 85. The initial state can contain an arbitrary amount of input sugars with the additional presence of PTS_g and DNA for PTS_f. The *conditional* regulation ensures the rules r2 and r7 cannot be used when glucose{6P} molecule is present in the cytosol. Other molecules have given no prohibited context.

Before the above-mentioned terms were even established in biology, their regulatory effects were already observed [CH76]. It was observed that in the presence of both glucose and fructose, the bacteria first consume glucose. The presence of fructose is completely ignored, and the bacteria start to metabolise it just after the glucose

source is depleted. The schematic overview of this stage of knowledge is available in [Figure 85](#).

We describe this system by a regulated model available in [Figure 86](#) (we refer to it as a *simple* model). It consists of eight rewriting rules. Rules *r1* to *r4* represent the transport of fructose and glucose by their respective PTS through the membrane. Consequently, phosphorylated intermediate products – fructose 1-phosphate (*fructose{1P}*) and glucose 6-phosphate (*glucose{6P}*) – are catabolised to pyruvate (rules *r5* and *r6*). Finally, the synthesis of fructose PTS is represented by rule *r7* with additional transport from cytosol to membrane (rule *r8*). To describe the observed yet unknown mechanisms of inducer exclusion and catabolite repression, we employ *conditional* regulation. This regulation specifies that once *glucose{6P}* is present (i.e. metabolism of glucose has started), rules *r2* and *r7* cannot be used. The regulation does not specify any additional details. Since it is separated from the rules description, it should immediately bring to the modeller's attention that such knowledge is abstract and explicitly modelled this way, possibly due to insufficient knowledge or is neglected on purpose.

With the progression of research on metabolic processes and their control, more insight was gathered into how these processes work (see [Figure 87](#) for an overview). For example, the glucose-specific PTS is governed by enzyme EII, composed of three subunits A, B, and C. The subunits B and C are membrane-bound, while subunit A has regulatory effects. The phosphorylated form of EII-A transfers its phosphoryl group to glucose, causing its phosphorylation to 6-P form. EII-A is phosphorylated in a cascade of phosphorylations, starting in phosphoenolpyruvate (PEP), through cytoplasmic components Enzyme I. (EI) and phosphohistidine carrier protein Hpr.

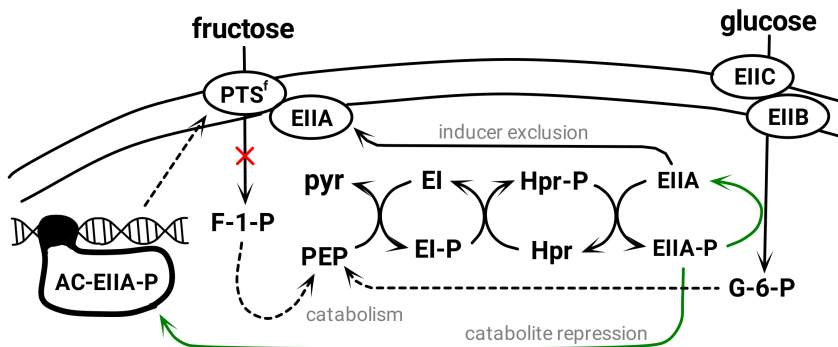


Figure 87: Schematic representation of sugars metabolism in *E. Coli* for the *extended* version with advanced knowledge. The dashed arrows represent simplified rules (neglecting irrelevant details to minimise the model size), while the colourful arrows are involved in the usage of regulations.

The unphosphorylated form of EII-A is responsible for inducer exclusion by directly binding PTS-specific membrane enzymes of secondary sugars, rendering it incapable of transporting the sugar inside the cell.

The phosphorylated form of EII-A also serves as a regulator of catabolite repression by binding adenylate cyclase (AC), which in consequence, activates promoters of many catabolic genes and operons. However, it was observed [Esc+12] that the process of glucose transport regulation is prioritised over catabolite repression. The gathered details are not enough to capture this fact by rules, but we can describe it using a *concurrent-free* regulation, assigning priority to a rule responsible for glucose transport.

We described the advanced knowledge of the metabolism by a regulated model available in Figure 88 (we refer to it as an *extended* model). Compared to the *simple* model, we extended the glucose transport (rules r4 and r5) by EII enzymes, where EII-A is dephosphorylated in the process and can bind PTS of fructose PTS_f (rule r3). The phosphorylated form of EII-A can also bind with AC (rule r8), forming a complex involved in PTS_f synthesis (rule r10). To ensure the priority of glucose transport over PTS_f synthesis, we employ concurrent-free regulation that assigns priority to rule 4 over rule 8. Finally, rules 12 to 14 implement the cascade of phosphorylations from pyruvate through EI and Hpr. Similarly to the previous model version, the regulation emphasises that the knowledge of the mechanism involved in priority decision-making is abstract and incomplete.

Comparing the extended version of the model to the simple one, we successfully eliminated the conditional regulation by implementing additional knowledge from the literature. This, however, exposed another weak spot of rule concurrency, which we implemented by concurrent-free regulation. Perhaps with additional details, it would be possible to eliminate the usage of regulations completely. It can be, for example, achieved by adding quantitative details [Bis+03a], i.e. assigning probability or weights to concurrent rules. These are, however, typically the most challenging details to infer.

7.5 SUMMARY

The purpose of this chapter was to give an insight into the modelling approach using BCSL on practical examples. We first demonstrated several typical patterns from the biological domain. In particular, on the model of the circadian clock of cyanobacteria, we showed how a non-trivial phosphorylation cycle consisting of proteins forming complexes could be compactly represented using the base BCSL and some of its syntactic extensions. On a model of fibroblast growth factor signalling pathway, we showed how the size of the model could be

```

#! rules
# fructose
r1 ~ fructose{n}::ext + PTS_f()::mem ⇔ fructose{n}.PTS_f()::mem
r2 ~ fructose{n}.PTS_f()::mem ⇒ fructose{1P}::cyt + PTS_f()::mem
r3 ~ PTS_f()::mem + EIIA{n}::cyt ⇒ PTS_f().EIIA{n}::cyt

# glucose
r4 ~ glucose{n}::ext - EIIA{p}::cyt + EIIBC()::mem ⇔
    ⇔ glucose{n}.EIIA{p}.EIIBC()::mem
r5 ~ glucose{n}.EIIA{p}.EIIBC()::mem ⇒
    ⇒ glucose{6P}::cyt + EIIA{n}::cyt + EIIBC()::mem

# catabolism to PEP == pyruvate{p}
r6 ~ glucose{6P}::cyt ⇒ pyruvate{p}::cyt
r7 ~ fructose{1P}::cyt ⇒ pyruvate{p}::cyt

# DNA
r8 ~ EIIA{p}::cyt + AC()::cyt ⇒ EIIA{p}.AC()::cyt
r9 ~ DNA()::nuc + EIIA{p}.AC()::cyt ⇒ DNA().EIIA{p}.AC()::nuc
r10 ~ DNA().EIIA{p}.AC()::nuc ⇒
    ⇒ DNA()::nuc + EIIA{p}.AC()::cyt + PTS_f()::cyt
r11 ~ PTS_f()::cyt ⇒ PTS_f()::mem

# phosphorylation cascade
r12 ~ pyruvate{p}::cyt + EI{n}::cyt ⇒ pyruvate{n}::cyt + EI{p}::cyt
r13 ~ EI{p}::cyt + Hpr{n}::cyt ⇒ EI{n}::cyt + Hpr{p}::cyt
r14 ~ Hpr{p}::cyt + EIIA{n}::cyt ⇒ Hpr{n}::cyt + EIIA{p}::cyt

#! inits
1 DNA()::nuc
1 fructose{n}::ext
1 glucose{n}::ext
1 EIIA{p}::cyt
1 EIIBC()::mem
1 EI{n}::cyt
1 Hpr{n}::cyt
1 AC()::cyt

#! regulation
type concurrent-free
(r4, r8)

```

Figure 88: The *extended* variant of the sugar metabolism model, schematically available in [Figure 87](#). The initial state can contain an arbitrary amount of input sugars with the additional presence of all required EII enzymes and cytoplasmatic components. The employed *concurrent-free* regulation is assigning priority to rule r4 over rule r8.

significantly reduced by employing rule-based abstraction. Finally, on a population-based model of tumour growth, we also demonstrated

the application of quantitative rate laws, enabling the possibility of simulating the model.

We also demonstrated the usage of parameter synthesis on real-world examples. For this purpose, we used a fragment of the circadian clock represented by a model with partially known quantitative details. Using two experiment setups, we investigated the behaviour of the model with respect to specified PCTL properties and parameter spaces. We also analysed the tumour growth model, finding the regions of specified parameter space leading to uncontrolled cells growth. For both models, we also computed the robustness of the properties, stating a measure of how well it is preserved across the parameter space.

In the next section, we applied the introduced static analysis techniques and reductions to the presented signalling pathway model. We were particularly interested in some (non)reachability properties, investigated both statically and dynamically on a reduced model.

Finally, we dedicated a whole section to a demonstration of the usage of regulations in BCSL. We showed that by using conditional regulation in a signalling pathway, we could directly model the inhibition. We also used programmed regulation in a cell cycle model, where we enforced the programmed death of cells under certain conditions. For a model of methane combustion, we applied concurrent-free regulation to exclude two enabled processes from happening concurrently. In the described circadian clock model, we used regular regulation to target a specific scenario in the vast non-deterministic alternative runs. In addition, on a larger case study of sugar metabolism in *E. Coli*, we showed a typical model development workflow in biology and how regulations could potentially contribute to this process.

CONCLUSIONS

In this thesis, we have contributed to the rule-based universe by introducing a novel BCSL language. The language is developed for describing biochemical processes on a mechanistic level. By employing the rule-based approach, it can serve as a suitable representation of large-scale models. The language is motivated by its core role in CMP, a platform for modelling biological systems, and a need for language with the syntactic simplicity of chemical reactions with specific abstractions. Indeed, besides the pragmatic approach in employing the notation suitable in systems biology, BCSL offers specific features, making it a valuable contribution among other rule-based languages. Specifically, BCSL relaxes the explicit bonding of molecules on the level of complexes. Instead, complexes represent the coexistence of molecules, where the particular biochemical interpretation is left to the modeller. Complexes are also treated rather uniquely in the rewriting rules, not allowing context-free modifications of complex parts.

Technically, we defined the language using fundamental mathematical operations and objects and related it to low-level multiset rewriting systems. The multiset-based formalism was primarily chosen for the closeness of solution representation to the biochemical reality, allowing us to relate such formalisms via a common base easily.

We also explored regulated rewriting in the context of the rule-based approach. Regulated rewriting was established primarily in string grammars and other rewriting systems but was never employed in the modelling of biological processes, specifically using the rule-based approach. We introduced several distinct regulatory mechanisms, first generally on the multiset rewriting systems and then on BCSL. In particular, we presented the following regulated rewriting classes: *regular* given by regular language over rules, *ordered* given by strict partial order on rules, *programmed* given by successors for each rule, *conditional* given by prohibited context for each rule, and *concurrent-free* given by resolving concurrent behaviour of multiple rules. We also explored the properties of individual classes and the relationships among them. We found that while some of the classes form strict subset relations, there are also incomparable classes. Additionally, we investigated expressive power for representatives of regulated classes and found that employing the regulations can increase the expressive power of particular formalism, as was proven for multiset rewriting systems.

To support modelling using BCSL in general and the specific behaviour influenced by regulations, we developed analysis methods to

investigate the properties of models and explore their behaviour. The most common method to observe whether models correspond to desired intentions is simulation. We adopted a standard stochastic simulation algorithm for the rule-based setting as well as described how the deterministic simulation using ODEs can be achieved. To provide more precise guarantees on analysis results, we also explored exact methods. We proposed techniques for CTL and PCTL model checking and parameter synthesis in the case of quantitative BCSL models with unknown parameters. We also extended the existing static analysis base for rule-based models by defining a method for the reduction of models to purely complex-interactions level, provided a method for static non-reachability analysis, and detection of redundant rules. These can be used to study the behaviour of biochemical models and help navigate the research in the right direction.

The proposed analysis techniques were then implemented in our tool, eBCSgen, developed to enable modelling using BCSL and analysis of created models. To reach a broader audience, we decided to employ a standardised way of how to present the tool to the users. Therefore, eBCSgen is distributed as a series of Galaxy tools to simplify the adoption process by the target user community. The usability of the tool is also supported by detailed documentation and usage tutorial, as well as straightforward installation using either Galaxy or conda package manager. The tool is enriched by visualisations suitable for displaying the results from analysis methods, such as parameter space with highlighted regions satisfying the given PCTL formula, simulation plots, and state transition system explorer.

To show how to use the language in practice, apply the most significant analysis techniques, and leverage the regulations in the process of modelling, we also present a series of case studies targeting individual contributions. In particular, we demonstrated the usage of BCSL on the circadian clock in cyanobacteria comprising complex protein interactions, a model of fibroblast growth factor signalling pathway, and a population model of tumour growth with quantitative aspects. The demonstration of analysis methods was focused on parameter synthesis and static analysis using models of already described processes. Finally, we demonstrated the regulations on a model of MAPK signalling pathway substituting inhibition using conditional regulation, a model of cell cycle enforcing programmed death using programmed regulation, a model of methane combustion excluding incomplete combustion using concurrent-free regulation, a model of circadian clock filtering a specific subset of behaviour using regular regulation. Finally, on a more detailed model of metabolism in *E. Coli* we showed the potential impact of regulations on a typical model development workflow in systems biology.

APPENDIX: DIGITAL ATTACHMENTS

All electronic materials used in this thesis are attached in the supplementary files. While the tool eBCSgen with its sources is available on GitHub¹, we also attach its newest version 2.1.0 as an compressed archive.

The materials also include all the examples and case study models used across the thesis as well as the analysis results. These are available as exported Galaxy histories. To investigate a Galaxy history, we recommend to use the following steps to import it in a Galaxy instance:

1. Click on *User* on the top menu of the Galaxy server.
2. Click on *Histories* to view saved histories.
3. Click on *Import history* button on the top right.
4. Select the appropriate importing method – *Upload local file from your computer* in our case.
5. Click the link text to check out your histories if import is successful.

In case you need a quick access to a Galaxy instance, we recommend to use a public server², our Galaxy instance³, or run a local instance using planemo⁴.

EBCSGEN: SOURCES AND DOCUMENTATION

- [eBCSgen-2.1.0.zip](#)

Sources of eBSCgen tool, including sources for documentation⁵ and automatic tests.

- [eBCSgen_tutorial.pdf](#)

Document containing tutorial to eBCSgen tool, with focus on usage in Galaxy environment and explanation of regulations.

¹ <https://github.com/sybila/eBCSgen>

² <https://usegalaxy.org/>

³ <https://biodivine-vm.fi.muni.cz/galaxy/>

⁴ <https://planemo.readthedocs.io/>

⁵ <https://ebcsgen.readthedocs.io/>

- [Galaxy-History-Tutorial-files.tar.gz](#)

Running examples used in the first part of tutorial, covering whole analysis base of eBCSgen in Galaxy.

- [Galaxy-History-Regulated-BCSL-examples.tar.gz](#)

Running examples used in the second part of tutorial dedicated to regulations.

EXAMPLES AND CASE STUDIES

- [Galaxy-History-Regulated-BCSL-examples.tar.gz](#)

Galaxy history containing all examples of regulated BCSL models used in [Section 4.3](#). The history also contains all respective transition systems and a brief CTL analysis of their properties.

- [Galaxy-History-Analysis.tar.gz](#)

Galaxy history containing examples from [Chapter 5](#) in the same order as used in the chapter.

- [Galaxy-History-Case-studies.tar.gz](#)

Galaxy history containing all case studies used in [Chapter 7](#). The history contains relevant transitions system where applicable and computed results of presented analysis methods.

BIBLIOGRAPHY

- [Afg+18] Enis Afgan, Dannon Baker, B  renice Batut, Marius van den Beek, Dave Bouvier, Martin   ech, John Chilton, et al. "The Galaxy Platform for Accessible, Reproducible and Collaborative Biomedical Analyses: 2018 Update". In: *Nucleic Acids Res.* 46.W1 (2018), pp. 537–544.
- [Alm22] B.V. Almende. *vis-network*. Version 9.1.2. 2022. URL: <https://www.npmjs.com/package/vis-network>.
- [Alo41] Church Alonzo. "The Calculi of Lambda Conversion". In: *Annals of Mathematics Studies* 6 (1941).
- [AOB13] Stephanie M. Amato, Mehmet A. Orman, and Mark P. Brynildsen. "Metabolic Control of Persister Formation in *Escherichia Coli*". In: *Molecular cell* 50.4 (2013), pp. 475–487.
- [Ana] *Anaconda Software Distribution*. Version Vers. 2-2.4.0. 2020. URL: <https://docs.anaconda.com/>.
- [And+13] Jakob L. Andersen, Christoph Flamm, Daniel Merkle, and Peter F. Stadler. "Inferring Chemical Reaction Patterns Using Rule Composition in Graph Grammars". In: *Journal of Systems Chemistry* 4.1 (2013), pp. 1–14.
- [And+17] Jakob L. Andersen, Christoph Flamm, Daniel Merkle, and Peter F. Stadler. "An Intermediate Level of Abstraction for Computational Systems Chemistry". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 375.2109 (2017), p. 20160354.
- [And16] Steven S. Andrews. "Smoldyn: Particle-based Simulation with Rule-based Modeling, Improved Molecular Interaction and a Library Interface". In: *Bioinformatics* 33.5 (2016), pp. 710–717.
- [Azi+00] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. "Model-checking Continuous-time Markov Chains". In: *ACM Transactions on Computational Logic* 1.1 (2000), pp. 162–170.
- [BBW18] Michael Backenk  hler, Luca Bortolussi, and Verena Wolf. "Moment-based Parameter Estimation for Stochastic Reaction Networks in Equilibrium". In: *TCBB* 15.4 (2018), pp. 1180–1192.

- [Bai+20] Christel Baier, Christian Hensel, Lisa Hutschenreiter, Sebastian Junges, Joost-Pieter Katoen, and Joachim Klein. “Parametric Markov Chains: PCTL Complexity and Fraction-free Gaussian Elimination”. In: *Information and Computation* 272 (2020).
- [Bar+08] Roberto Barbuti, Giulio Caravagna, Andrea Maggiolo-Schettini, and Paolo Milazzo. “An Intermediate Language for the Simulation of Biological Systems”. In: *Electronic Notes in Theoretical Computer Science* 194.3 (2008), pp. 19–34.
- [Bar+17] Jiří Barnat, Nikola Beneš, Luboš Brim, Martin Demko, Matej Hajnal, Samuel Pastva, and David Šafránek. “Detecting Attractors in Biological Models with Uncertain Parameters”. In: *International Conference on Computational Methods in Systems Biology*. Springer. 2017, pp. 40–56.
- [Bar+11] Jiří Barnat, Luboš Brim, Adam Krejčí, Adam Streck, David Šafránek, Martin Vejnár, and Tomáš Vejpušek. “On Parameter Synthesis by Parallel Model Checking”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 9.3 (2011), pp. 693–705.
- [Bar22] Matthew Barnett. *regex*. Version 2022.10.31. 2022. URL: <https://pypi.org/project/regex>.
- [BO14] Kevin J. Barnum and Matthew J. O’Connell. “Cell Cycle Regulation by Checkpoints”. In: *Cell cycle control*. Springer, 2014, pp. 29–40.
- [Bat+10] Gregory Batt, Michel Page, Irene Cantone, Gregor Goessler, Pedro Monteiro, and Hidde De Jong. “Efficient Parameter Search for Qualitative Models of Regulatory Networks using Symbolic Model Checking”. In: *Bioinformatics* 26.18 (2010), pp. i603–i610.
- [BYG17] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal Methods for Discrete-time Dynamical Systems*. Vol. 89. Studies in Systems, Decision and Control. Springer, 2017.
- [Ben+17] Nikola Beneš, Luboš Brim, Martin Demko, Matej Hajnal, Samuel Pastva, and David Šafránek. “Discrete Bifurcation Analysis with Pithya”. In: *International Conference on Computational Methods in Systems Biology*. Springer. 2017, pp. 319–320.
- [Ben+16] Nikola Beneš, Luboš Brim, Martin Demko, Samuel Pastva, and David Šafránek. “A Model Checking Approach to Discrete Bifurcation Analysis”. In: *International Symposium on Formal Methods*. Springer. 2016, pp. 85–101.

- [Ben+18] Nikola Beneš, Luboš Brim, Samuel Pastva, David Šafránek, Matej Troják, Jan Červený, and Jakub Šalagovič. “Fully Automated Attractor Analysis of Cyanobacteria Models”. In: *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE. 2018, pp. 354–359.
- [BKVo3] Marc Bezem, Jan W. Klop, and Roel de Vrijer. *Term Rewriting Systems*. Cambridge University Press, 2003.
- [BCG95] Girish Bhat, Rance Cleaveland, and Orna Grumberg. “Efficient On-the-fly Model Checking for CTL”. In: *Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science*. IEEE. 1995, pp. 388–397.
- [Bis+03a] Stefano Bistarelli, Iliano Cervesato, Gabriele Lenzini, Roberto Marangoni, and Fabio Martinelli. “On Representing Biological Systems Through Multiset Rewriting”. In: *International Conference on Computer Aided Systems Theory*. Springer. 2003, pp. 415–426.
- [Bis+03b] Stefano Bistarelli, Iliano Cervesato, Gabriele Lenzini, and Fabio Martinelli. *Relating Process Algebras and Multiset Rewriting for Security Protocol Analysis*. Carnegie Mellon University, 2003.
- [Bli+06] Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Hlavacek. “A Network Model of Early Events in Epidermal Growth Factor Receptor Signaling that Accounts for Combinatorial Complexity”. In: *Biosystems* 83.2-3 (2006), pp. 136–151.
- [Boc+15] Christoph Bock, Luca Bortolussi, Thilo Krüger, Linar Mikeev, and Verena Wolf. “Model-based Whole-genome Analysis of DNA Methylation Fidelity”. In: *Hybrid Systems Biology*. Springer, 2015, pp. 141–155.
- [Boo87] Ronald V. Book. “Thue Systems as Rewriting Systems”. In: *Journal of Symbolic Computation* 3.1-2 (1987), pp. 39–68.
- [Bor+08] Benjamin J. Bornstein, Sarah M. Keating, Akiya Jouraku, and Michael Hucka. “libSBML: An API Library for SBML”. In: *Bioinformatics* 24.6 (2008), pp. 880–881. ISSN: 1367-4803.
- [BMS16] Luca Bortolussi, Dimitrios Milios, and Guido Sanguinetti. “Smoothed Model Checking for Uncertain Continuous-time Markov Chains”. In: *Information and computation* 247 (2016), pp. 235–253. ISSN: 0890-5401.
- [BS18] Luca Bortolussi and Simone Silvetti. “Bayesian Statistical Parameter Synthesis for Linear Temporal Properties of Stochastic Models”. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2018, pp. 396–413.

- [Bou+18a] Pierre Boutillier, Ferdinanda Camporesi, Jean Coquet, Jérôme Feret, Kim Quyên Lý, Nathalie Theret, and Pierre Vignet. “KaSa: A Static Analyzer for Kappa”. In: *International Conference on Computational Methods in Systems Biology*. Springer. 2018, pp. 285–291.
- [Bou+18b] Pierre Boutillier, Mutaamba Maasha, Xing Li, Héctor F. Medina-Abarca, Jean Krivine, Jérôme Feret, Ioana Cristescu, Angus G. Forbes, and Walter Fontana. “The Kappa Platform for Rule-Based Modeling”. In: *Bioinformatics* 34.13 (2018), pp. i583–i592.
- [Bri+13] Luboš Brim, Milan Češka, Sven Dražan, and David Šafránek. “Exploring Parameter Space of Stochastic Biochemical Systems using Quantitative Model Checking”. In: *International Conference on Computer Aided Verification*. Springer. 2013, pp. 107–123.
- [Bri+01] Luboš Brim, David Gilbert, Jean-Marie Jacquet, and Mojmir Křetínský. “Multi-agent Systems as Concurrent Constraint Processes”. In: *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer. 2001, pp. 201–210.
- [Bri+96] Luboš Brim, Jean-Marie Jacquet, David Gilbert, and Mojmir Křetínský. “A Process Algebra for Synchronous Concurrent Constraint Programming”. In: *International Conference on Algebraic and Logic Programming*. Springer. 1996, pp. 165–178.
- [Bry01] Randal E. Bryant. *Graph-based Algorithms for Boolean Function Manipulation*. Tech. rep. Carnegie-Mellon University Pittsburgh PA School Of Computer Science, 2001.
- [Bur+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and Lain-Jinn Hwang. “Symbolic Model Checking: 1020 States and Beyond”. In: *Information and computation* 98.2 (1992), pp. 142–170.
- [CFS06] Laurence Calzone, François Fages, and Sylvain Soliman. “BIOCHAM: An Environment for Modeling Biological Systems and Formalizing Experimental Knowledge”. In: *Bioinformatics* 22.14 (2006), pp. 1805–1807.
- [CFL17] Ferdinanda Camporesi, Jérôme Feret, and Kim Quyên Lý. “KaDE: A Tool to Compile Kappa Rules into (Reduced) ODE Models”. In: *International Conference on Computational Methods in Systems Biology*. Springer. 2017, pp. 291–299.
- [Caro8] Luca Cardelli. “From Processes to ODEs by Chemistry”. In: *The 5th IFIP International Conference On Theoretical Computer Science (TCS)*. Boston, MA: Springer US, 2008, pp. 261–281.

- [Cas22] Alberto Casagrande. *pyModelChecking*. Version 1.3.3. 2022. URL: <https://pypi.org/project/pyModelChecking>.
- [Cer94] Iliano Cervesato. *Petri Nets as Multiset Rewriting Systems in a Linear Framework*. Citeseer, 1994.
- [Cer95] Iliano Cervesato. “Petri Nets and Linear Logic: a Case Study for Logic Programming”. In: *Proceedings of the 1995 Joint Conference on Declarative Programming*. Palladio Press. 1995, pp. 313–318.
- [Čes+14] Milan Česka, David Šafránek, Sven Dražan, and Luboš Brim. “Robustness Analysis of Stochastic Biochemical Systems”. In: *PLOS ONE* 9.4 (2014), e94553.
- [Chy+13] Lily A. Chylek, Edward C. Stites, Richard G. Posner, and William S. Hlavacek. “Innovations of the Rule-based Modeling Approach”. In: *Systems Biology*. Springer, 2013, pp. 273–300.
- [Cim+02] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. “NuSMV 2: An Open-source Tool for Symbolic Model Checking”. In: *International Conference on Computer Aided Verification*. Springer. 2002, pp. 359–364.
- [CHo8] Federica Ciocchetta and Jane Hillston. “Bio-PEPA: an Extension of the Process Algebra PEPA for Biochemical Networks”. In: *Electronic Notes in Theoretical Computer Science* 194.3 (2008), pp. 103–117.
- [CHo9] Federica Ciocchetta and Jane Hillston. “Bio-PEPA: A Framework for the Modelling and Analysis of Biological Systems”. In: *Theoretical Computer Science* 410 (2009), pp. 3065–3084.
- [Cla+12] Allan Clark, Vashti Galpin, Stephen Gilmore, Maria Luisa Guerriero, and Jane Hillston. “Formal Methods for Checking the Consistency of Biological Models”. In: *Advances in Systems Biology*. Springer, 2012, pp. 461–475.
- [CH76] B. Clark and W. H. Holms. “Control of the Sequential Utilization of Glucose and Fructose by *Escherichia Coli*”. In: *Microbiology* 95.2 (1976), pp. 191–201.
- [CJ+18] Edmund M. Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model Checking*. MIT press, 2018.
- [Cla97] Edmund M. Clarke. “Model Checking”. In: *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer. MIT press, 1997, pp. 54–56.

- [CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. "Automatic Verification of Finite-state Concurrent Systems using Temporal Logic Specifications". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 8.2 (1986), pp. 244–263.
- [Cla+08] Edmund M. Clarke, James R. Faeder, Christopher J. Langmead, Leonard A. Harris, Sumit Kumar Jha, and Axel Legay. "Statistical Model Checking in BioLab: Applications to the Automated Analysis of T-cell Receptor Signaling Pathway". In: *International Conference on Computational Methods in Systems Biology*. Springer. 2008, pp. 231–250.
- [Cou+92] Costas Courcoubetis, Moshe Vardi, Pierre Wolper, and Mihalis Yannakakis. "Memory-efficient Algorithms for the Verification of Temporal Properties". In: *Formal methods in system design* 1.2-3 (1992), pp. 275–288.
- [CC77] Patrick Cousot and Radhia Cousot. "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fix-points". In: *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM. 1977, pp. 238–252.
- [Cov14] Markus W. Covert. *Fundamentals of Systems Biology: from Synthetic Circuits to Whole-cell Models*. CRC Press, 2014.
- [Dan+09] Vincent Danos, Jérôme Feret, Walter Fontana, Russ Harmer, and Jean Krivine. "Rule-based Modelling and Model Perturbation". In: *Transactions on Computational Systems Biology XI*. Berlin, Heidelberg: Springer, 2009, pp. 116–137.
- [Dan+07] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. "Rule-based Modelling of Cellular Signalling". In: *International conference on concurrency theory*. Springer. 2007, pp. 17–41.
- [DHW13] Vincent Danos, Russ Harmer, and Glynn Winskel. "Constraining Rule-Based Dynamics with Types". In: *Mathematical Structures in Computer Science* 23.2 (2013), pp. 272–289.
- [DK07] Vincent Danos and Jean Krivine. "Formal Molecular Biology Done in CCS-R". In: *Electronic Notes in Theoretical Computer Science* 180.3 (2007), pp. 31–49.
- [Das04] Jürgen Dassow. "Grammars with Regulated Rewriting". In: *Formal Languages and Applications*. Springer, 2004, pp. 249–273.

- [Dawo4] Conrado Daws. "Symbolic and Parametric Model Checking of Discrete-time Markov Chains". In: *International Colloquium on Theoretical Aspects of Computing*. Springer. 2004, pp. 280–294.
- [Děd+16] Tadeáš Děd, David Šafránek, Matej Troják, Matej Klement, Jakub Šalagovič, and Luboš Brim. "Formal Biochemical Space with Semantics in Kappa and BNGL". In: *Electronic Notes in Theoretical Computer Science* 326 (2016), pp. 27–49.
- [Deh+17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. "A Storm is Coming: A Modern Probabilistic Model Checker". In: *International Conference on Computer Aided Verification*. Springer. 2017, pp. 592–600.
- [Delo2] Giorgio Delzanno. "An Overview of MSR (C): A CLP-based Framework for the Symbolic Verification of Parameterized Concurrent Systems". In: *Electronic Notes in Theoretical Computer Science* 76 (2002), pp. 65–82.
- [DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. "Rewrite Systems". In: *Formal models and semantics*. Elsevier, 1990, pp. 243–320.
- [Don10] Alexandre Donzé. "Breach, a Toolbox for Verification and Parameter Synthesis of Hybrid Systems". In: *International Conference on Computer Aided Verification*. Springer. 2010, pp. 167–170.
- [DCL10] Alexandre Donzé, Gilles Clermont, and Christopher J. Langmead. "Parameter Synthesis in Nonlinear Dynamical Systems: Application to Systems Biology". In: *Journal of Computational Biology* 17.3 (2010), pp. 325–336.
- [Don+11] Alexandre Donzé, Eric Fanchon, Lucie Martine Gattepaille, Oded Maler, and Philippe Tracqui. "Robustness Analysis and Behavior Discrimination in Enzymatic Reaction Networks". In: *PLOS ONE* 6.9 (2011), e24246.
- [DMo7] Alexandre Donzé and Oded Maler. "Systematic Simulation using Sensitivity Analysis". In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2007, pp. 174–189.
- [ELoo] Michael B. Elowitz and Stanislas Leibler. "A Synthetic Oscillatory Network of Transcriptional Regulators". In: *Nature* 403.6767 (2000), pp. 335–338.
- [Eng+09] Heinz W. Engl, Christoph Flamm, Philipp Kügler, James Lu, Stefan Müller, and Peter Schuster. "Inverse Problems in Systems Biology". In: *Inverse Problems* 25.12 (2009), p. 123014.

- [Esc+12] Adelfo Escalante, Ania Salinas Cervantes, Guillermo Gosset, and Francisco Bolívar. "Current Knowledge of the Escherichia Coli Phosphoenolpyruvate–Carbohydrate Phosphotransferase System: Peculiarities of Regulation and Impact on Growth and Product Formation". In: *Applied microbiology and biotechnology* 94.6 (2012), pp. 1483–1494.
- [FH07a] Jasmin Fisher and Thomas A. Henzinger. "Executable Cell Biology". In: *Nature biotechnology* 25.11 (2007), p. 1239.
- [FB96] Walter Fontana and Leo W. Buss. "The Barrier of Objects: From Dynamical Systems to Bounded Organizations". In: (1996).
- [FMVPO4] Rudolf Freund, Carlos Martín-Vide, and Gheorghe Păun. "From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies". In: *Theoretical Computer Science* 312.2-3 (2004), pp. 143–188.
- [FH07b] Perry A. Frey and Adrian D. Hegeman. *Enzymatic Reaction Mechanisms*. Oxford University Press, 2007.
- [GBH09] Vashti Galpin, Luca Bortolussi, and Jane Hillston. "HYPE: a Process Algebra for Compositional Flows and Emergent Behaviour". In: *International Conference on Concurrency Theory*. Springer, 2009, pp. 305–320.
- [GHBo8] Vashti Galpin, Jane Hillston, and Luca Bortolussi. "HYPE Applied to the Modelling of Hybrid Biological Systems". In: *Electronic Notes in Theoretical Computer Science* 218 (2008), pp. 33–51.
- [Gil76] Daniel T. Gillespie. "A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions". In: *Journal of computational physics* 22.4 (1976), pp. 403–434.
- [Gil77] Daniel T. Gillespie. "Exact Stochastic Simulation of Coupled Chemical Reactions". In: *The journal of physical chemistry* 81.25 (1977), pp. 2340–2361.
- [Gil00] Daniel T. Gillespie. "The Chemical Langevin Equation". In: *The Journal of Chemical Physics* 113.1 (2000), pp. 297–306.
- [GM11] Stefania Gnesi and Franco Mazzanti. "An Abstract, On-the-fly Framework for the Verification of Service-oriented Systems". In: *Rigorous software engineering for service-oriented systems*. Springer, 2011, pp. 390–407.
- [Gol+97] Susan S. Golden, Masahiro Ishiura, Carl H. Johnson, and Takao Kondo. "Cyanobacterial Circadian Rhythms". In: *Annual review of plant biology* 48.1 (1997), pp. 327–354.

- [Grü+18] Björn Grüning, Ryan Dale, Andreas Sjödin, Brad A. Chapman, Jillian Rowe, Christopher H. Tomkins-Tinch, Renan Valieris, and Johannes Köster. “Bioconda: Sustainable and Comprehensive Software Distribution for the Life Sciences”. In: *Nature Methods* 15.7 (2018), pp. 475–476. ISSN: 1548-7091.
- [HHZ11] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. “Probabilistic Reachability for Parametric Markov Models”. In: *International Journal on Software Tools for Technology Transfer* 13.1 (2011), pp. 3–19.
- [Han17] John T Hancock. *Cell Signalling*. Oxford University Press, 2017.
- [HD82] W. Harder and Lubbert Dijkhuizen. “Strategies of Mixed Substrate Utilization in Microorganisms”. In: *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 297.1088 (1982), pp. 459–480.
- [Har+16] Leonard A. Harris, Justin S. Hogg, Jose-Juan Tapia, John Sekar, Sanjana Gupta, Ilya Korsunsky, Arshi Arora, Dipak Barua, Robert P. Sheehan, and James R. Faeder. “BioNet-Gen 2.2: Advances in Rule-based Modeling”. In: *Bioinformatics* 32.21 (2016), pp. 3366–3368.
- [HJ94] H. Hasson and Bengt Jonsson. “A Logic for Reasoning about Time and Probability”. In: *Formal Aspects of Computing* 6 (1994), pp. 512–535.
- [HSH13] Mostafa Herajy, Martin Schwarick, and Monika Heiner. “Hybrid Petri Nets for Modelling the Eukaryotic Cell Cycle”. In: *Transactions on Petri Nets and Other Models of Concurrency VIII*. Springer, 2013, pp. 123–141.
- [HBA13] S. Hertel, Ch. Brettschneider, and I. M. Axmann. “Revealing a Two-loop Transcriptional Feedback Mechanism in the Cyanobacterial Circadian Clock”. In: *PLoS Computational Biology* 9.3 (2013), e1002966.
- [Higo8] Desmond J. Higham. “Modeling and Simulating Chemical Reactions”. In: *SIAM review* 50.2 (2008), pp. 347–368.
- [HP05] A. C. Hindmarsh and L. R. Petzold. *LSODA, Ordinary Differential Equation Solver for Stiff or Non-stiff System*. 2005.
- [Hla+19] William S. Hlavacek, Jennifer A. Csicsery-Ronay, Lewis R. Baker, María del Carmen Ramos Álamo, Alexander Ionkov, Eshan D. Mitra, Ryan Suderman, Keesha E. Erickson, Raquel Dias, Joshua Colvin, et al. “A Step-by-step Guide to Using BioNetFit”. In: *Modeling Biomolecular Site Dynamics*. Springer, 2019, pp. 391–419.

- [Hog+98] Boris M. Hogema, Jos C. Arents, Rechien Bader, Kevin Eijkemans, Toshifumi Inada, Hiroji Aiba, and Pieter W. Postma. "Inducer Exclusion by Glucose 6-phosphate in *Escherichia Coli*". In: *Molecular microbiology* 28.4 (1998), pp. 755–765.
- [Holo4] Gerard J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Vol. 1003. Addison-Wesley Reading, 2004.
- [HZ+17] Ricardo Honorato-Zimmer, Andrew J. Millar, Gordon D. Plotkin, and Argyris Zardilis. "Chromar, a Rule-based Language of Parameterised Objects". In: *Theoretical Computer Science* 335 (2017), pp. 49–66.
- [Hop08] John E. Hopcroft. *Introduction to Automata Theory, Languages, and Computation*. Pearson Education India, 2008.
- [Huc+03] Michael Hucka, Andrew Finney, Herbert M Sauro, Hamid Bolouri, John C. Doyle, Hiroaki Kitano, Adam P. Arkin, Benjamin J. Bornstein, Dennis Bray, Athel Cornish-Bowden, et al. "The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models". In: *Bioinformatics* 19.4 (2003), pp. 524–531.
- [HK97] Michael Huth and Marta Kwiatkowska. "Quantitative Analysis and Model Checking". In: *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*. IEEE. 1997, pp. 111–122.
- [IH88] Atsunobu Ichikawa and Kunihiro Hiraishi. "Analysis and Control of Discrete Event Systems Represented by Petri Nets". In: *Discrete Event Systems: Models and Applications*. Springer, 1988, pp. 115–134.
- [Inc15] Plotly Technologies Inc. *Collaborative data science*. 2015. URL: <https://plot.ly>.
- [Ish+98] Masahiro Ishiura, Shinsuke Kutsuna, Setsuyuki Aoki, Hideo Iwasaki, Carol R. Andersson, Akio Tanabe, Susan S Golden, Carl H. Johnson, and Takao Kondo. "Expression of a Gene Cluster kaiABC as a Circadian Feedback Process in Cyanobacteria". In: *Science* 281.5382 (1998), pp. 1519–1523.
- [Iwa+02] Hideo Iwasaki, Taeko Nishiwaki, Yohko Kitayama, Masato Nakajima, and Takao Kondo. "KaiA-stimulated KaiC Phosphorylation in Circadian Timing Loops in Cyanobacteria". In: *Proceedings of the National Academy of Sciences* 99.24 (2002), pp. 15788–15793.

- [Jan+14] Nils Jansen, Florian Corzilius, Matthias Volk, Ralf Wimmer, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. “Accelerating Parametric Probabilistic Verification”. In: *The International Conference on Quantitative Evaluation of SysTems (QEST)*. Springer. 2014, pp. 404–420.
- [Jha+09] Sumit K. Jha, Edmund M. Clarke, Christopher J. Langmead, Axel Legay, André Platzer, and Paolo Zuliani. “A Bayesian Approach to Model Checking Biological Systems”. In: *International Conference on Computational Methods in Systems Biology*. Springer. 2009, pp. 218–234.
- [JL11] Sumit Kumar Jha and Christopher James Langmead. “Synthesis and Infeasibility Analysis for Stochastic Models of Biochemical Systems using Statistical Model Checking and Abstraction Refinement”. In: *Theoretical Computer Science* 412.21 (2011), pp. 2162–2187.
- [Joh+11] Mathias John, Cédric Lhoussaine, Joachim Niehren, and Cristian Versari. “Biochemical Reaction Rules with Constraints”. In: *European symposium on programming*. Springer. 2011, pp. 338–357.
- [KF09] Sertac Karaman and Emilio Frazzoli. “Sampling-based Motion Planning with Deterministic μ -calculus Specifications”. In: *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE. 2009, pp. 2222–2229.
- [Kat+03] Mitsunori Katayama, Takao Kondo, Jin Xiong, and Susan S. Golden. “LdpA Encodes an Iron-sulfur Protein Involved in Light-dependent Modulation of the Circadian Period in the Cyanobacterium *Synechococcus Elongatus* PCC 7942”. In: *Journal of bacteriology* 185.4 (2003), pp. 1415–1422.
- [Kea+20] Sarah M. Keating, Dagmar Waltemath, Matthias König, Fengkai Zhang, Andreas Dräger, Claudine Chaouiya, Frank T. Bergmann, Andrew Finney, Colin S. Gillespie, Tomáš Helíkar, et al. “SBML Level 3: an Extensible Format for the Exchange and Reuse of Biological Models”. In: *Molecular systems biology* 16.8 (2020), e9110.
- [KJ18] A. Khalid and S. K. Jha. “Calibration of Rule-Based Stochastic Biochemical Models using Statistical Model Checking”. In: *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2018, pp. 179–184.
- [Khoo6] Boris N. Kholodenko. “Cell-signalling Dynamics in Time and Space”. In: *Nature reviews Molecular cell biology* 7.3 (2006), pp. 165–176.

- [Kit02] Hiroaki Kitano. “Computational Systems Biology”. In: *Nature* 420.6912 (2002), pp. 206–210.
- [Kit04] Hiroaki Kitano. “Biological Robustness”. In: *Nature Reviews Genetics* 5.11 (2004), p. 826.
- [Kit07] Hiroaki Kitano. “Towards a Theory of Biological Robustness”. In: *Molecular systems biology* 3.1 (2007), p. 137.
- [Kle+14] Matej Klement, Tadeáš Děd, David Šafránek, Jan Červený, Stefan Mueller, and Ralf Steuer. “Biochemical Space: A Framework for Systemic Annotation of Biological Models”. In: *Electronic Notes in Theoretical Computer Science* 306 (2014), pp. 31–44.
- [Kle+13] Matej Klement, David Šafránek, Tadeáš Děd, Ales Pejznoch, Ladislav Nedbal, Ralf Steuer, Jan Červený, and Stefa Mueller. “A Comprehensive Web-based Platform for Domain-specific Biological Models”. In: *Electronic Notes in Theoretical Computer Science* 299 (2013), pp. 61–67.
- [KFL12] Paweł Kocieniewski, James R. Faeder, and Tomasz Lipniacki. “The Interplay of Double Phosphorylation and Scaffolding in MAPK Pathways”. In: *Journal of Theoretical Biology* 295 (2012), pp. 116–124.
- [Kos82] S. Rao Kosaraju. “Decidability of Reachability in Vector Addition Systems”. In: *Symposium on Theory of computing*. Vol. 82. ACM. 1982, pp. 267–281.
- [KNS14] Andres Kriete, Eishi Noguchi, and Christian Sell. “Introductory Review of Computational Cell Cycle Modeling”. In: *Cell Cycle Control* (2014), pp. 267–275.
- [Kri63] Saul Kripke. “Semantical Considerations on Modal Logic”. In: *Acta Philosophica Fennica* 16 (1963), pp. 83–94.
- [KNP06] Marta Kwiatkowska, Gethin Norman, and David Parker. “Quantitative Analysis with the Probabilistic Model Checker PRISM”. In: *Electronic Notes in Theoretical Computer Science* 153.2 (2006), pp. 5–31.
- [KNP10] Marta Kwiatkowska, Gethin Norman, and David Parker. “Symbolic Systems Biology”. In: Jones and Bartlett, 2010. Chap. Probabilistic Model Checking for Systems Biology, pp. 31–59.
- [KNP11] Marta Kwiatkowska, Gethin Norman, and David Parker. “PRISM 4.0: Verification of Probabilistic Real-time Systems”. In: *International conference on computer aided verification*. Springer. 2011, pp. 585–591.

- [LLM14] Diego Latella, Michele Loreti, and Mieke Massink. “On-the-fly Probabilistic Model Checking”. In: *arXiv preprint arXiv:1410.7469* (2014).
- [LS14] Jérôme Leroux and P. Schnoebelen. “On Functions Weakly Computable by Petri Nets and Vector Addition Systems”. In: *International Workshop on Reachability Problems*. Springer. 2014, pp. 190–202.
- [Lip76] Richard Lipton. “The Reachability Problem Requires Exponential Space”. In: *Department of Computer Science. Yale University* 62 (1976).
- [LF16] Bing Liu and James R. Faeder. “Parameter Estimation of Rule-based Models using Statistical Model Checking”. In: *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE. 2016, pp. 1453–1459.
- [Lop+13] Carlos F. Lopez, Jeremy L. Muhlich, John A. Bachman, and Peter K. Sorger. “Programming Biological Models in Python using PySB”. In: *Molecular systems biology* 9.1 (2013).
- [LW16] Alexander Lück and Verena Wolf. “Generalized Method of Moments for Estimating Parameters of Stochastic Reaction Networks”. In: *BMC Systems Biology* 10.1 (2016), p. 98.
- [MNO4] Oded Maler and Dejan Nickovic. “Monitoring Temporal Properties of Continuous Signals”. In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.
- [MRU11] Carsten Maus, Stefan Rybacki, and Adelinde M. Uhrmacher. “Rule-based Multi-level Modeling of Cell Biological Systems”. In: *BMC systems biology* 5.1 (2011), pp. 1–20.
- [Mer14] Dirk Merkel. “Docker: Lightweight Linux Containers for Consistent Development and Deployment”. In: *Linux journal* 2014.239 (2014), p. 2.
- [Mes92] José Meseguer. “Conditional Rewriting Logic as a Unified Model of Concurrency”. In: *Theoretical computer science* 96.1 (1992), pp. 73–155.
- [Meu+17] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, et al. “SymPy: Symbolic Computing in Python”. In: *PeerJ Computer Science* 3 (2017), e103.
- [MM+13] Leonor Michaelis, Maud L. Menten, et al. “Die Kinetik der Invertinwirkung”. In: *Biochem Z* 49.333–369 (1913), p. 352.
- [Mil99] Robin Milner. *Communicating and Mobile Systems: The π Calculus*. Cambridge university press, 1999.

- [Min67] Marvin Lee Minsky. *Computation*. Prentice-Hall Englewood Cliffs, 1967.
- [Miy+07] Fumihiko Miyoshi, Yoichi Nakayama, Kazunari Kaizu, Hideo Iwasaki, and Masaru Tomita. “A Mathematical Model for the Kai-protein-based Chemical Oscillator and Clock Gene Expression Rhythms in Cyanobacteria”. In: *Journal of Biological Rhythms* 22.1 (2007), pp. 69–80.
- [MBo8] Leonardo de Moura and Nikolaj Bjørner. “Z3: An Efficient SMT Solver”. In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2008, pp. 337–340.
- [Mül+06] Stefan Müller, Josef Hofbauer, Lukas Endler, Christoph Flamm, Stefanie Widder, and Peter Schuster. “A Generalized Model of the Repressilator”. In: *Journal of mathematical biology* 53.6 (2006), pp. 905–937.
- [Nak+05] Masato Nakajima, Keiko Imai, Hiroshi Ito, Taeko Nishiwaki, Yoriko Murayama, Hideo Iwasaki, Tokitaka Oyama, and Takao Kondo. “Reconstitution of Circadian Oscillation of Cyanobacterial KaiC Phosphorylation in Vitro”. In: *Science* 308.5720 (2005), p. 414.
- [NNH15] Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer, 2015.
- [Pat+04] Rekha Pattanayek, Jimin Wang, Tetsuya Mori, Yao Xu, Carl H. Johnson, and Martin Egli. “Visualizing a Circadian Clock Protein: Crystal Structure of KaiC and Functional Insights”. In: *Molecular cell* 15.3 (2004), pp. 375–388.
- [PMR12] Loïc Paulevé, Morgan Magnin, and Olivier Roux. “Static Analysis of Biological Regulatory Networks Dynamics using Abstract Interpretation”. In: *Mathematical Structures in Computer Science* 22.4 (2012), pp. 651–685.
- [Pau+10] Loïc Paulevé, Simon Youssef, Matthew R. Lakin, and Andrew Phillips. “A Generic Abstract Machine for Stochastic Process Calculi”. In: *Proceedings of the 8th International Conference on Computational Methods in Systems Biology*. ACM. 2010, pp. 43–54.
- [Pău07] Gheorghe Păun. “Membrane Computing and Brane Calculi (some personal notes)”. In: *Electronic Notes in Theoretical Computer Science* 171.2 (2007), pp. 3–10.
- [Pea+01] Gray Pearson, Fred Robinson, Tara Beers Gibson, Bing-e Xu, Mahesh Karandikar, Kevin Berman, and Melanie H. Cobb. “Mitogen-activated Protein (MAP) Kinase Pathways: Regulation and Physiological Functions”. In: *Endocrine reviews* 22.2 (2001), pp. 153–183.

- [PPP15] Michael Pedersen, Andrew Phillips, and Gordon D. Plotkin. “A High-level Language for Rule-based Modelling”. In: *PLOS ONE* 10.6 (2015), e0114296.
- [PP10] Michael Pedersen and Gordon D. Plotkin. “A Language for Biochemical Systems: Design and Formal Specification”. In: *Transactions on Computational Systems Biology XII: Special Issue on Modeling Methodologies*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 77–145.
- [Pet81] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981.
- [Pnu77] Amir Pnueli. “The Temporal Logic of Programs”. In: *18th Annual Symposium on Foundations of Computer Science (SFSC)*. IEEE. 1977, pp. 46–57.
- [PR00] David Poole and Adrian E. Raftery. “Inference for Deterministic Simulation Models: the Bayesian Melding Approach”. In: *Journal of the American Statistical Association* 95.452 (2000), pp. 1244–1255.
- [Qua+16] Tim Quatmann, Christian Dehnert, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. “Parameter Synthesis for Markov Models: Faster than Ever”. In: *International Symposium on Automated Technology for Verification and Analysis*. Springer. 2016, pp. 50–67.
- [Ram+14] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M. Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. “Model Predictive Control with Signal Temporal Logic Specifications”. In: *53rd IEEE Conference on Decision and Control*. IEEE. 2014, pp. 81–87.
- [RS02] Aviv Regev and Ehud Shapiro. “Cells as Computation”. In: *Nature* 419.6905 (2002), p. 343. ISSN: 0028-0836.
- [RSS00] Aviv Regev, William Silverman, and Ehud Shapiro. “Representation and Simulation of Biochemical Processes using the π Calculus Process Algebra”. In: *Biocomputing 2001*. World Scientific, 2000, pp. 459–470.
- [Riz+09] Aurélien Rizk, Gregory Batt, François Fages, and Sylvain Soliman. “A General Computational Method for Robustness Analysis with Applications to Synthetic Gene Networks”. In: *Bioinformatics* 25.12 (2009), pp. i169–i178.
- [Šaf+11] David Šafránek, Jan Červený, Matěj Klement, Jana Pospíšilová, Luboš Brim, Dušan Lazár, and Ladislav Nedbal. “E-photosynthesis: Web-based Platform for Modeling of Complex Photosynthetic Processes”. In: *BioSystems* 103.2 (2011), pp. 115–124.
- [Sch98] K. A. Schafer. “The Cell Cycle: a Review”. In: *Veterinary pathology* 35.6 (1998), pp. 461–478.

- [Sco93] Roger S. Scowen. “Generic Base Standards”. In: *Proceedings 1993 Software Engineering Standards Symposium*. IEEE. 1993, pp. 25–34.
- [SSVS13] Carla Seatzu, Manuel Silva, and Jan H. Van Schuppen. *Control of Discrete-event Systems*. Vol. 433. Springer, 2013.
- [SS63] John C. Shepherdson and Howard E. Sturgis. “Computability of Recursive Functions”. In: *Journal of the ACM (JACM)* 10.2 (1963), pp. 217–255.
- [Shi22] Erez Shinan. *lark*. Version 1.1.5. 2022. URL: <https://pypi.org/project/lark>.
- [Shm+17] Fedor Shmarov, Nicola Paoletti, Ezio Bartocci, Shan Lin, Scott A. Smolka, and Paolo Zuliani. “SMT-based Synthesis of Safe and Robust PID Controllers for Stochastic Hybrid Systems”. In: *Haifa Verification Conference*. Springer. 2017, pp. 131–146.
- [SD21] Jonathan Slack and Leslie Dale. *Essential Developmental Biology*. John Wiley & Sons, 2021.
- [SFE11] Michael W. Sneddon, James R. Faeder, and Thierry Emonet. “Efficient Modeling, Simulation and Coarse-graining of Biological Complexity with NFsim”. In: *Nature Methods* 8.2 (2011), pp. 177–183.
- [Sor+13] Oksana Sorokina, Anatoly Sorokin, J. Douglas Armstrong, and Vincent Danos. “A Simulator for Spatially Extended Kappa Models”. In: *Bioinformatics* 29.23 (2013), p. 3105.
- [Ste+14] Melanie I. Stefan, Thomas M. Bartol, Terrence J. Sejnowski, and Mary B. Kennedy. “Multi-state Modeling of Biomolecules”. In: *PLoS Computational Biology* 10.9 (2014), e1003844.
- [Tho90] Wolfgang Thomas. “Automata on Infinite Objects”. In: *Formal Models and Semantics*. Elsevier, 1990, pp. 133–191.
- [Tro+22] Matej Troják, David Šafránek, Branislav Brozmann, and Luboš Brim. “eBCSgen 2.0: Modelling and Analysis of Regulated Rule-Based Systems”. In: *International Conference on Computational Methods in Systems Biology*. Springer. 2022, pp. 302–309.
- [Tro+16] Matej Troják, David Šafránek, Jakub Hrabec, Jakub Šalagovič, Františka Romanovská, and Jan Červený. “E-cyanobacterium.org: A Web-based Platform for Systems Biology of Cyanobacteria”. In: *International Conference on Computational Methods in Systems Biology*. Vol. 9859. LNBI. Springer. Springer, 2016, pp. 316–322.

- [Tro+20a] Matej Troják, David Šafránek, Lukrécia Mertová, and Luboš Brim. “Parameter Synthesis and Robustness Analysis of Rule-based Models”. In: *NASA Formal Methods Symposium*. Springer. 2020, pp. 41–59.
- [Tro+20b] Matej Troják, David Šafránek, Lukrécia Mertová, and Luboš Brim. “eBCSgen: A Software Tool for Biochemical Space Language”. In: *Computational Methods in Systems Biology*. Ed. by Alessandro Abate, Tatjana Petrov, and Verena Wolf. Cham: Springer International Publishing, 2020, pp. 356–361.
- [TvB20] Matej Troják, David Šafránek, and Luboš Brim. “Executable Biochemical Space for Specification and Analysis of Biochemical Systems”. In: vol. 350. *International Workshop on Static Analysis and Systems Biology (SASB)*. Elsevier, 2020, pp. 91–116.
- [Tro+20c] Matej Troják, David Šafránek, Lukrécia Mertová, and Luboš Brim. “Executable Biochemical Space for Specification and Analysis of Biochemical Systems”. In: *PLOS ONE* 15.9 (2020), pp. 1–24.
- [TCNN02] John J. Tyson, Attila Csikasz-Nagy, and Bela Novák. “The Dynamics of Cell Cycle Regulation”. In: *Bioessays* 24.12 (2002), pp. 1095–1109.
- [TN15] John J. Tyson and Béla Novák. “Models in Biology: Lessons from Modeling Regulation of the Eukaryotic Cell Cycle”. In: *BMC Biology* 13.1 (2015), pp. 1–10.
- [TN01] John J. Tyson and Bela Novák. “Regulation of the Eukaryotic Cell Cycle: Molecular Antagonism, Hysteresis, and Irreversible Transitions”. In: *Journal of theoretical biology* 210.2 (2001), pp. 249–263.
- [Vil+13] S. A. Villarreal, R. Pattanayek, D. R. Williams, T. Mori, X. Qin, C. H. Johnson, M. Egli, and P. L. Stewart. “CryoEM and Molecular Dynamics of the Circadian KaiB–KaiC Complex Indicates that KaiB Monomers Interact with KaiC and Block ATP Binding Clefts”. In: *Journal of molecular biology* 425.18 (2013), pp. 3311–3324.
- [VR03] Minaya Villasana and Ami Radunskaya. “A Delay Differential Equation Model for Tumor Growth”. In: *Journal of Mathematical Biology* 47.3 (2003), pp. 270–294.
- [Vir+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature methods* 17.3 (2020), pp. 261–272.

- [Voioo] Eberhard O. Voit. *Computational Analysis of Biochemical Systems: A Practical Guide for Biochemists and Molecular Biologists*. Cambridge University Press, 2000.
- [Wen+14] John E. Wenskovitch, Leonard A. Harris, Jose-Juan Tapia, James R. Faeder, and G. Elisabeta Marai. “MOSBIE: A Tool for Comparison and Analysis of Rule-based Biochemical Models”. In: *BMC Bioinformatics* 15.1 (2014), p. 316.
- [Wil+16] Mark D. Wilkinson, Michel Dumontier, I. Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, et al. “The FAIR Guiding Principles for Scientific Data Management and Stewardship”. In: *Scientific data* 3.1 (2016), pp. 1–9.
- [Xu+11] Wen Xu, Adam M. Smith, James R. Faeder, and G. Elisabeta Marai. “RuleBender: A Visual Interface for Rule-based Modeling”. In: *Bioinformatics* 27.12 (2011), pp. 1721–1722.
- [YTY04] Satoshi Yamada, Takaharu Taketomi, and Akihiko Yoshimura. “Model Analysis of Difference Between EGF Pathway and FGF Pathway”. In: *Biochemical and Biophysical Research Communications* 314.4 (2004), pp. 1113–1120.
- [Yan+08] Jin Yang, Michael I. Monine, James R. Faeder, and William S. Hlavacek. “Kinetic Monte Carlo Method for Rule-based Modeling of Biochemical Networks”. In: *Physical Review E* 78.3 (2008), p. 031910.
- [ZMS18] Fengkai Zhang and Martin Meier-Schellersheim. “SBML Level 3 Package: Multistate, Multicomponent and Multicompartment Species, Version 1, Release 1”. In: *Journal of Integrative Bioinformatics* 15.1 (2018).