eBCSgen: A Software Tool for Biochemical Space Language

Matej Troják, David Šafránek, Lukrécia Mertová, and Luboš Brim

Systems Biology Laboratory, Masaryk University, Brno, Czech Republic

Abstract. eBCSgen is a tool for development and analysis of models written in Biochemical Space Language (BCSL). BCSL is a rule-based language for biological systems designed to combine compact description with a specific level of abstraction which makes it accessible to users from life sciences. Currently, eBCSgen represents the only tool completely supporting BCSL. It has the form of a command line interface which is integrated into Galaxy – a web-based bioinformatics platform automating data-driven and model-based analysis pipelines.

1 Introduction

Rule-based modelling is a promising approach in systems biology which can be used to write mechanistic models of complex reaction systems. Compared to traditional mechanistic or mathematical approaches such as reaction-based modelling or ordinary differential equations (ODEs), the rule-based approach provides a compact form of model description that scales well with the size and complexity of the modelled system.

Key features of rule-based languages, such as structures binding [3,10], regulatory interactions [17], modularity [16], or spatial aspects [12], combined with a language-specific level of abstraction, require the development and analysis of rule-based models to be supported by software tools. Moreover, to appropriately reflect the needs of the biological domain, it is necessary to enable analysis of models with incomplete information (e.g., unknown kinetic parameters).

Several existing rule-based languages are provided with well-established software support. Kappa [6] is supported by the Kappa platform [3], providing a model editor, stochastic simulation, several static analysis procedures accompanied by graphical visualisation, and a generator of ODE models. BioNetGen package [10] provides the tool RuleBender [20] for construction, debugging, analysis (e.g. simulation, parameter scan), and visualisation of models (e.g. influence graph, contact map). The software environment BioCham [4] with its custom language [5] supports multiple semantics and allows, for example, checking of temporal properties expressed in CTL, analysing models with respect to FO-LTL properties (measuring the robustness, parameter sensitivity), and simulating models. Some other languages employ embedding to an existing programming language (e.g. Chromar [12], PySB [14]).

Our experiences with using these languages in direct collaboration with biologists have shown that in most cases, it is difficult to train users outside computer science to use these languages directly. Although some of the tools are quite intuitive, they either do not support various useful features or work with a very detailed level of abstraction that makes models hard to understand, maintain, and re-use. To that end, we have introduced Biochemical Space Language (BCSL) [9,18], a high-level rule-based language that combines several features of rule-based frameworks in a single formalism, recently extended by quantitative aspects [19].

In this paper, we present eBCSgen, a tool for the development and analysis of models written in BCSL. The tool is integrated into Galaxy [1], a web-based platform for data-intensive biomedical research. It provides a convenient way to use eBCSgen by the target group of users due to the extensive popularity of Galaxy in biology-oriented community. Our tool provides interactive model editor, model simulation, analysis of the model with respect to PCTL [11] properties, useful static analysis methods, and interactive data visualisation. We demonstrate the tool usability on a case study describing circadian rhythms in cyanobacteria. eBCSgen is available online[1] within Galaxy. It is accompanied with a short tutorial[2].

2 Biochemical Space Language

In this section, we briefly show the primary features of BCSL on several examples. For more details, we recommend [18] for formal definition and [19] for formal semantics and analysis methods.

A BCSL model is given by a set of *rules*, an *initial state*, and a set of *definitions* (optional). A rule describes a pattern how *agents* (structured objects) can interact. Examples of the rules are provided below. The initial state defines the number of individual agents in the initial solution of the system. The definitions assign particular values to parameters. Additionally, there can be fourth part defining complex aliases (see below). In the following expression

```
A(act{off})::cyt + B()::cyt ⇒ A(act{on}).B()::cyt @ k1×[A()::cyt]×[B()::cyt]
```

there is a rule describing the interaction of agent `A()` with agent `B()`, creating a complex `A().B()`; moreover, the agent `A()` changes the state of its feature `act` from `off` to `on`, meaning its activity was turned on. This interaction takes place inside `cyt` (cytosol) physical compartment. The *rate* of the rule is given by a mass action kinetic law with no restriction on the particular state of the agent `A()`, parametrised by a parameter `k1`.

The example above demonstrated the basic features of the language – the formation of a complex and a state change. Additional features are for example complex dissociation (the rule above with opposite direction), agent formation (the left-hand side of the rule is empty), and degradation (no right-hand side).

[1] `https://biodivine-vm.fi.muni.cz/galaxy/`

[2] `https://biodivine.fi.muni.cz/galaxy/eBCSgen/tutorial`

Among these basic constructs, the language offers several syntactic features which make BCSL models more readable and compact. *Nesting* allows "zooming" inside of individual agents to emphasise the particular part of the agent. For example, the rule

```
act{off}:A():A().B()::cell ⇒ act{on}:A():A().B()::cell @ k2×[A().B()::cell]
```

describes the state change of feature `act` *inside* (indicated by single colon symbol) of agent `A()` as a part of complex `A().B()` (localised in compartment `cell`). Note that agent `A(act{off}).B()::cell` represents an equivalent form to that on the left-hand side of the rule without the usage of the nesting operator.

The syntax using nesting is particularly useful in combination with *complex aliases* and *variables*. The complex alias allows defining a short name for a particular complex (for example, `AB = A().B()`, `AC = A().C()`, `AD = A().D()`). The variable can substitute multiple agents on a particular position in the rule, providing a compact way of aggregating repeating patterns. The rule

```
act{off}:A():?::cell ⇒ act{on}:A():?::cell @ k2×[?::cell]
```

followed by `? = {AB, AC, AD}` describes the previous rule and two additional rules compactly – individual complex aliases are substituted on the position declared by question mark.

The semantics of the model is given by transitive rewriting of the rules starting with the initial state. The rule rewriting is defined by *match–replace* relation, which first selects the suitable candidates from the state (they have to satisfy the left-hand side of the rule) and then they are replaced according to the right-hand side of the rule. During this process, the rate is evaluated as a function of the state. Finally, all outgoing rates from the state are normalised to obtain the probability of the transition. Following the idea of using approximate models with discrete-time semantics [2], the obtained transition system is a Discrete Time Markov Chain (DTMC) or a parametric Markov Chain (pMC) [7,13], depending on whether there are some parameters used in rule rates which do not have defined value in *definitions* section of the model.

3 Implementation

The tool eBCSgen is implemented in Python programming language. It is developed as a command-line tool with a GUI provided by integration into Galaxy [1], 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 an individual GUI development.

The Galaxy interface of eBCSgen offers interactive *model editor*, which can be used to create and edit BCSL models. The interactivity is ensured by automatic syntax highlighting and real-time code validation. Any errors in the model code are immediately highlighted.

The probabilistic behaviour of BCSL model can be analysed with respect to PCTL [11] properties. PCTL is an extension of computation tree logic (CTL) which allows for probabilistic quantification of described properties. The given property is checked using `Storm` model checker [8] after the corresponding transition system is generated. The tool allows checking whether a given probability threshold is satisfied or to find the probability of satisfaction for given path formula. In the case of the parametrised model, Storm is used to solve *parameter synthesis*. If the formula has defined probability threshold, then Storm computes the partitioning of the given parameter space (defined by the user) to regions which satisfy (resp. violate) the property. If the threshold is not given, a probability function of parameters is computed instead, which evaluates to the probability of satisfaction for particular parametrisation.

In addition to these analysis methods, the tool provides *stochastic simulation* and several static analysis techniques to improve the scalability issues of exhaustive computational methods. In particular, we have developed a method to *detect* (potentially) *redundant* rules in the model. The absence of a redundant rule in the model does not change the behaviour of the model since a more general rule already exists in the model. This can be useful in large models to detect potentially conflicting rules and to make the model more compact. 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. Finally, the static analysis of *unreachability* can be used to check whether an agent is unreachable without the need to enumerate the transition system. This analysis is based on the idea that in order to reach an agent, there must be a rule which either produces the agent or its more abstract form.

An important 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 (Fig. 1 left). Moreover, it is possible to visualise the generated transition system and the result of he sampling of the probability function of parameters produced by parameter synthesis.

4 Experimental results

For the purpose of evaluation, we consider the model Miyoshi et al. [15] describing circadian rhythms in cyanobacteria formed by three proteins controlled by repeated phosphorylation and dephosphorylation of key protein and complex formation with other proteins. A simplified version of the model composed of 10 rules in BCSL (in contrast to 27 explicit reactions) has been analysed using a prototype version of eBCSgen in [19].

In this paper, we consider a full version of the model[3] with 9 rules (note the number of rules is lower, but the rules are more detailed, representing almost

[3] The model files and computed analysis results are available here:
`https://biodivine.fi.muni.cz/galaxy/eBCSgen/case-studies/cmsb-2020`
The results for simplified analysis are also publicly available:
`https://biodivine.fi.muni.cz/galaxy/eBCSgen/case-studies/nfm-2020`

700 explicit reactions) using the abstract syntax described above. Analysis of this model provided more detailed results compared to the simplified model. Similarly to the case study in [19], we analysed phosphorylation and dephosphorylation phases separately using PCTL parameter synthesis. The more detailed analyses showed that the oscillatory behaviour is dependent on particular values of parameters responsible for the dephosphorylation phase (Fig. 1 left). The original results showed much higher robustness with respect to parameter values (Fig. 1 right).

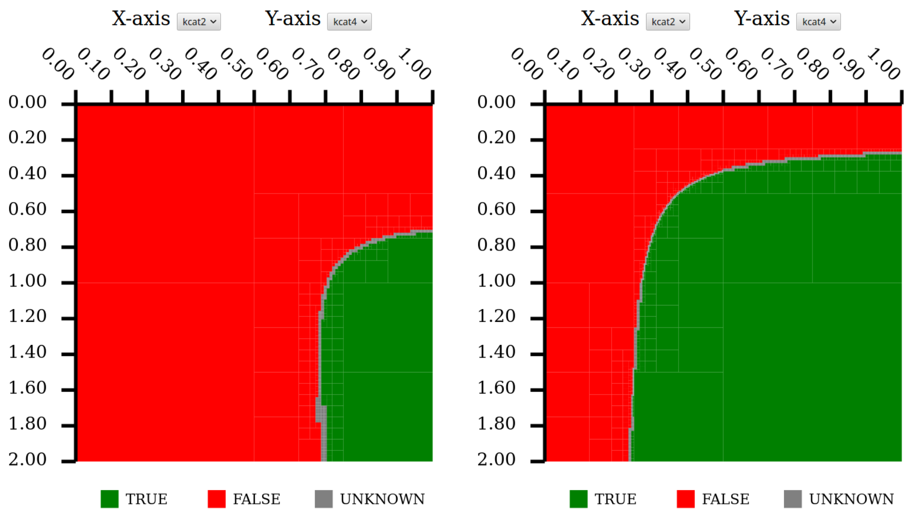

Fig. 1. Visualisation of the partitioning of the parameter space as a result of parameter synthesis for Miyoshi et al. model. The results are computed for the dephosphorylation phase with respect to the property of reaching a fully dephosphorylated target protein complex with a probability higher than `0.99`. The *left* figure shows original results from [19] for a simplified version of the model. The *right* figure shows results for the full version of the model.

5 Conclusion

We presented the tool eBCSgen with its primary features and capabilities. The tool serves as a base for development and analysis of models written in BCSL. It focuses on user-accessibility of its features. Since the most of the computational analysis is performed by external tools (e.g. PCTL property checking in Storm, simulation in Python package `scipy`), we focused on the language description, tool capabilities, visualisation as important factors for the user experience, and briefly explained experimental results from the biological domain. Regarding the performance of eBCSgen, the present bottleneck is the generation of explicit transition system. Our future steps are to focus on the scalability issues trying to completely avoid this step using symbolic approaches or on-the-fly techniques.

References

1. Afgan, E., et al.: The Galaxy Platform for Accessible, Reproducible and Collaborative Biomedical Analyses: 2018 Update. Nucleic acids research 46(W1), W537–W544 (2018)
2. Barbuti, R., et al.: An Intermediate Language for the Stochastic Simulation of Biological Systems. TCS 410(33-34), 3085–3109 (2009)
3. Boutillier, P., et al.: The Kappa Platform for Rule-Based Modeling. Bioinformatics 34(13), i583–i592 (2018)
4. Calzone, L., et al.: BIOCHAM: An Environment for Modeling Biological Systems and Formalizing Experimental Knowledge. Bioinformatics 22(14), 1805–1807 (2006)
5. Chabrier-Rivier, N., et al.: The Biochemical Abstract Machine BIOCHAM. In: International Conference on Computational Methods in Systems Biology. pp. 172–191. Springer (2004)
6. Danos, V., Laneve, C.: Formal Molecular Biology. Theoretical Computer Science 325, 69–110 (2004)
7. Daws, C.: Symbolic and Parametric Model Checking of Discrete-time Markov Chains. In: ICTAC. pp. 280–294. Springer (2004)
8. Dehnert, C.o.: A Storm is Coming: A Modern Probabilistic Model Checker. In: CAV. pp. 592–600. Springer (2017)
9. Děd, T., et al.: Formal Biochemical Space with Semantics in Kappa and BNGL. Electr. Notes Theor. Comput. Sci. 326, 27–49 (2016)
10. Harris, L.A., et al.: BioNetGen 2.2: Advances in Rule-based Modeling. Bioinformatics 32(21), 3366–3368 (2016)
11. Hasson, H., Jonsson, B.: A Logic for Reasoning about Time and Probability. FAOC 6, 512–535 (1994)
12. Honorato-Zimmer, R., et al.: Chromar, a Language of Parameterised Agents. Theoretical Computer Science 765, 97–119 (2019)
13. Lanotte, R., et al.: Parametric Probabilistic Transition Systems for System Design and Analysis. FAOC 19(1), 93–109 (2007)
14. Lopez, C.F., et al.: Programming Biological Models in Python using PySB. Molecular systems biology 9(1) (2013)
15. Miyoshi, F., et al.: A Mathematical Model for the Kai-protein-based Chemical Oscillator and Clock Gene Expression Rhythms in Cyanobacteria. Journal of Biological Rhythms 22(1), 69–80 (2007)
16. Pedersen, M., et al.: A High-level Language for Rule-based Modelling. PloS one 10(6), e0114296 (2015)
17. Romers, J.C., Krantz, M.: rxncon 2.0: A Language for Executable Molecular Systems Biology. bioRxiv (2017)
18. Troják, M., et al.: Executable Biochemical Space for Specification and Analysis of Biochemical Systems. arXiv 2002.00731 (2020)
19. Troják, M., et al.: Parameter Synthesis and Robustness Analysis of Rule-Based Models. In: NASA Formal Methods Symposium (2020), to appear.
20. Xu, W., et al.: RuleBender: A Visual Interface for Rule-based Modeling. Bioinformatics 27(12), 1721–1722 (2011)