bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

bioLQM: a java library for the manipulation and conversion of Logical Qualitative Models of biological networks

Aurélien Naldi

CNRS UMR8197, INSERM U1024, École Normale Supérieure, PSL Research University, Paris, France

aurelien.naldi@ens.fr

April 4, 2018

Abstract

Here we introduce bioLQM, a new Java software toolkit for the conversion, modification, and analysis of Logical Qualitative Models of biological regulatory networks, aiming to foster the development of novel complementary tools by providing core modelling operations. Based on the definition of multi-valued logical models, it implements import and export facilities, notably for the recent SBML-qual exchange format, as well as for formats used by several popular tools, facilitating the design of workflows combining these tools. Model modifications enable the definition of various perturbations, as well as model reduction, easing the analysis of large models. Another modification enables the study of multi-valued models with tools limited to the Boolean case. Finally, bioLQM provides a framework for the development of novel analysis tools. The current version implements the usual updating modes for model simulation (notably synchronous, asynchronous, and random asynchronous), as well as some static analysis features for the identification of attractors. The bioLQM software can be integrated into analysis workflows through command line and scripting interfaces. As a Java library, it further provides core data structures to the GINsim and EpiLog interactive tools, which supply graphical interfaces and additional analysis methods for cellular and multi-cellular qualitative models.

Keywords: Biological Networks, Qualitative Modeling, Computational Systems Biology.

1 Introduction

Logical models are highly abstract dynamical models, which have been proposed to study biological regulatory systems in the late 60s [16, 37]. This modelling framework has since gained popularity [5, 31, 34] and has been successfully applied to a wide range of regulatory and signalling systems [1, 14, 23, 33].

In logical models, components are represented by discrete variables with a small range of possible values, representing qualitative differences in activity. Boolean components can only be active (1) or inactive (0), while multi-valued components define increasing integer activity levels. Regulatory effects are often represented as signed arcs between components in the **regulatory graph**. These effects are further formalised as logical rules (also called logical parameters or logical functions), specifying the target activity level of each component according to the current levels of its regulators (a subset of all model components). While interactive software for model definition such as GINsim [25] or The Cell Collective [15] enable the definition of regulatory graphs, bioLQM solely relies on logical rules. However, signed regulatory arcs can be extracted from these logical rules if needed. The relative simplicity of this formalism enables the definition of large models with dozens of components, without requiring precise knowledge of kinetic parameters. A formal definition of logical qualitative models is provided in appendix A.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

The CoLoMoTo consortium was recently founded to facilitate model sharing and foster cooperation in the qualitative modelling community, building on the introduction of the SBML qual exchange format [7, 22]. The bioLQM toolkit presented here reinforces this effort by implementing a collection of model modification and format conversion operations in an extensible architecture illustrated in Figure 1. It can then be used as a building block for novel software development or as a bridge between complementary softwares in complex analysis workflows. It is currently embed in the GINsim software [25], which provides a graphical interface to most of its features. BioLQM is also used as backend for model definition and computation of successor states in Epilog [38], as well as in the CoLoMoTo notebook for model conversion and some dynamical analysis features [26].

Section 2 introduces model loading, saving and converting operations. Section 3 introduces the simulation and dynamical analysis features. Section 4 introduces model modifications. These features can be accessed from a command-line interface or through a scripting API, as discussed in Section 5.

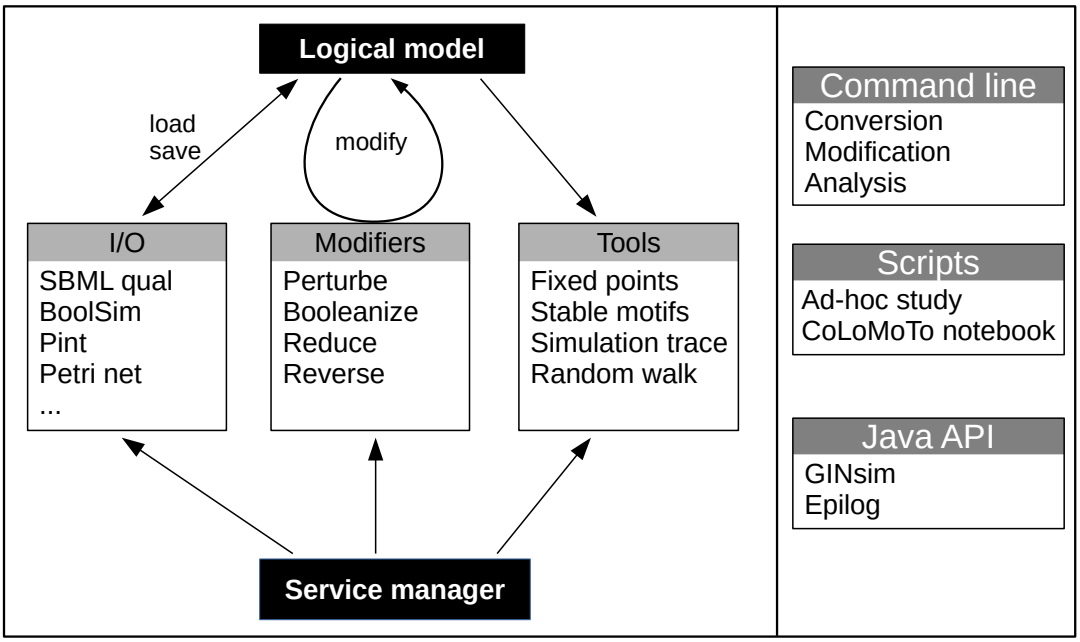


Figure 1: **Global structure of the bioLQM toolkit.** The bioLQM toolkit is centered around a data structure for the representation of logical qualitative models. Based on this data structure, (i) the `I/O` module contains a collection of formats enabling model loading and saving ; (ii) the `modifiers` module contains a collection of model modifiers to transform an input model into a modified model ; (iii) the `tools` module contains a collection of analysis tools. All these feature are accessible through a central `service manager`, which handles service discovery and serves as main entry point for the Java API. A simple `command line launcher` provides quick execution of simple workflows, while a `scripting engine` can be used for more complex use cases.

2 Loading and converting Logical Qualitative models

The increasing use of qualitative models to study biological systems led to the development of various software tools for the logical formalism [2, 12, 19, 25, 36] and related qualitative approaches [3, 28, 35]. Many software tools use their own file format for the definition of models, hindering the delineation of analysis workflows combining complementary software tools. The SBML qual exchange format [7] has recently been proposed to improve interoperability between modeling tools. However SBML support is often missing from existing software and is may not be a priority for newer ones.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

To ease model exchange between software tools that do not all support the SBML qual format, the bioLQM toolkit provides an extensible list of format handlers connected to the internal model representation. Each format is described as a Java class providing annotations (name of the format, default file extension and multi-valued support) along with optional implementations of model import (loading a file into the internal representation) or export (saving the internal representation to a file) operations. A service manager maintains a list of these descriptor classes through service discovery to facilitate the addition of new formats.

The supported formats are listed in Table 1 and in bioLQM documentation [1]. BioLQM uses JSBML [30] to load and save SBML qual models. The other import parsers are based on the antlr parser generator [27]. While some formats natively support multi-valued models, many are limited to the Boolean case. Multi-valued models can be exported to these Boolean formats through an implicit booleanization step, described in section 4.

File extension	multi-valued	Import	Export	Description and associated tools
sbml	x	x	x	SBML qual Exchange format [7]
bnet		x	x	BoolNet, PyBoolNet [18, 19]
booleannet		x	x	booleannet [2]
boolfunction	x		x	Boolean functions
boolsim	x		x	boolsim (genYsis) [12]
cnet	x	x	x	BNS [10]
ginml	x		x	GINsim [25]
tt	x	x	x	Truth table
an	x		x	Pint automata network [28]
apnn, pnml, ina	x		x	Conversion to Petri Net formats [6]
gna	x		x	GNA (Piecewise-linear formalism) [3]
bnd			x	MaBoSS (Stochastic Boolean model) [35]

Table 1: **Available formats.** The Import/Export capabilities are listed in the corresponding columns (all formats can be exported). The formats natively supporting multi-valued models are also identified, other formats rely on implicit model booleanization.

3 Model dynamics and simulation

A **state** of the model is a vector giving the activity levels of all its components. As the activity level of each component is restricted to a finite range, the **state space** (containing all possible states) itself is also finite. However, the total number of possible states grows exponentially with the number of components. We say that a component is **called to update** in a given state if the evaluation of the associated logical rule is different than its current activity level: for example an inactive component can become active. When analysing logical models, we are often interested in **stable states** (also called fixed points, or steady states), in which no component is called to update. Such stable states denote a qualitative equilibrium in which all components can maintain their current activity level.

The dynamics of the model (*i.e.* its evolution over time) is given by transitions between states states of the model, controlled by the updating calls (*i.e.* by the logical rules of the model) and by **updating modes** which define the synchronisation between concurrent updating calls. Various types of updating modes have been introduced, with most software tools focussing on a specific subset. BioLQM aims to provide an extensive set of updating modes in a single toolkit. In the following subsections, we further distinguish deterministic and non-deterministic simulations and provide an overview of all updating modes implemented in bioLQM. While stable states, which have no transition toward other states, do not depend on the updating mode, reachability properties and cyclical attractors can be

[1] See `http://colomoto.org/biolqm/`

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

strongly affected by the choice of updating mode as illustrated in Figure 2. More formal definitions of the updating calls and updating modes are given in appendix B.

Deterministic simulations

In a **deterministic** simulation, each state has a unique successor, except stable states which have no successor at all as we consider here that a successor must denote a change of state. Starting with an initial state, a deterministic simulation yields an ordered list of successive states, called a **trace**. Given a sufficient number of steps, all traces end in an **attractor**, which can be either a stable state or a **cyclical attractor** of length k in which the k-th successor of each state is itself. The `trace` tool, illustrated in appendix D, uses an initial state and a deterministic updater to compute a simulation trace. The following deterministic updaters are supported:

- In the classical **synchronous** updating (also called parallel), all logical rules are applied at the same time [16].
- A **sequential** updating applies all rules in a pre-determined order. Instead of evaluating all rules on the original state before updating all components at once as in the synchronous case, they are evaluated on the state obtained after applying the previous rule. The selected order can then change dramatically the successor state: a different sequential updater can be defined for each possible ordering.
- The **block-sequential** updaters generalizes the sequential updaters by considering groups of components updated synchronously [29]. The definition of this updater relies on an ordered partition of the model components.
- The **synchronous priority** updating is also based on a partition of components into blocks, but only the first block containing updated components will be considered. These updatings are a subset of the priority-based updatings introduced by Fauré et al. [11].

Non-deterministic simulations

In a **non-deterministic** simulation, each state can have several successors. Starting with an initial state, a non-deterministic simulation can lead to a large number of alternative trajectories. This type of dynamics if often represented as a **State Transition Graph** (STG), where the nodes are states of the model, and arcs denote possible transitions between these states. Like in the deterministic case, all trajectories end in an attractor, but starting from an initial state, a non-deterministic simulation can lead to several alternative attractors. These attractors can be **stable states**, **cyclical attractors**, as well as sets of intertwined cycles called **complex attractors**. More formally, all attractors are terminal strongly connected components of the STG. State Transition graphs can represent deterministic traces as well as more complex dynamical behaviours. Such a graph can cover several alternative initial states or even all possible states. The current version of bioLQM supports the definition of non-deterministic updaters, enabling the computation of the lists of successor states. However, it does not provide a complete engine for non-deterministic simulations, or a data structure for state transition graphs. GINsim [25] implements these features on top of bioLQM. The following non-deterministic updaters are supported:

- In the **asynchronous** updating, all logical rules are applied independently. All successors of a state differ from it on exactly one component [37].
- In the **complete** updating, any number of components can be updated at once. This set of successors includes all asynchronous successors, as well as the synchronous one.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

- The **priority** updater generalizes the *synchronous priority* introduced above by allowing some of the partitions (priority classes) to be updated asynchronously [11].

Stochastic simulations

Stochastic updaters enable the computation of a single succesor, which is selected randomly among multiple possibilities and can thus change between calls. A stochastic updater can be derived from any non-deterministic updater by assigning identical probabilities to all transitions defined by the original updater. Alternatively, a custom updater can be constructed by defining individual probabilities.

BioLQM provides the `random` tool to compute single random trajectories using the above stochastic updaters. This tool is limited to the construction of individual trajectories and does not provide a complete stochastic analysis. As listed in Table 1, bioLQM enables the conversion of Boolean models to the format of the MaBoSS software, which uses the Gillespie algorithm to estimate the probabilities of Boolean states of a continuous time Markov process, and provides a collection of scripts to further analyse the simulation results [35].

Identification of attractors

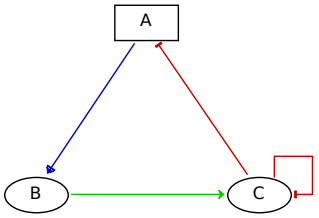
The dynamical analysis of large regulatory networks through model simulation suffers from combinatorial explosion, especially in the non-deterministic case. BioLQM implements two published methods based on constraint-solving for the identification of attractors without explicit state enumeration.

1. The first method enables the identification of **stable states** (fixed points) by extracting and combining stability conditions from the logical rules [21]. BioLQM includes this implementation, using decision diagrams to manipulate stability conditions, and introduces an alternative implementation based on the clingo ASP solver [13], which tends to be slower for small models, but can scale better in some cases. Similar methods are also available in the GNA and Pint tools [3, 28].

2. The efficient identification of cyclical attractors and complex attractors remain a challenging problem, especially as these attractors can depend on the updating mode. **Stable patterns** have recently been proposed as an approximation of complex attractors, which can be identified efficiently and does not depend on the updating mode [17, 40]. Here, a pattern is a partially-defined state where some components have a fixed activity level, while others are undefined. Such a pattern represents all states with matching activity levels for the defined components (*i.e.* 2^k possible states for k undefined Boolean components). A pattern is stable if the images of all included states belong to the pattern (the image of a state is its successor in a synchronous updating). BioLQM proposes an adapted version of the method implemented in PyBoolNet [17, 18] using the clingo ASP solver [13], and introduces a new alternative implementation based on decision diagrams.

While complex attractors are well estimated through stable patterns, their exact identification requires further analysis using external software tools, adapted to the selected updating mode. In the synchronous case, the BNS tool [10] uses a SAT solver to identify cyclical attractors of length k. This approach could be extended to other deterministic updatings, but can not handle non-deterministic cases. In contrast, BoolSim uses symbolic exploration for the identification of complex attractors in the synchronous and asynchronous case [12]. While this approach scales better than simple simulation, it is more sensitive to combinatorial explosion than approaches based on constraint-solving. To perform the analysis provided by the BoolSim and BNS tools, bioLQM can convert models to their respective formats.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.



$m_A = 2$
$m_B = m_C = 1$

$f_A = 2$ if C
$f_B = 1$ if $A = 1$
$f_C = 1$ if B & $!C$

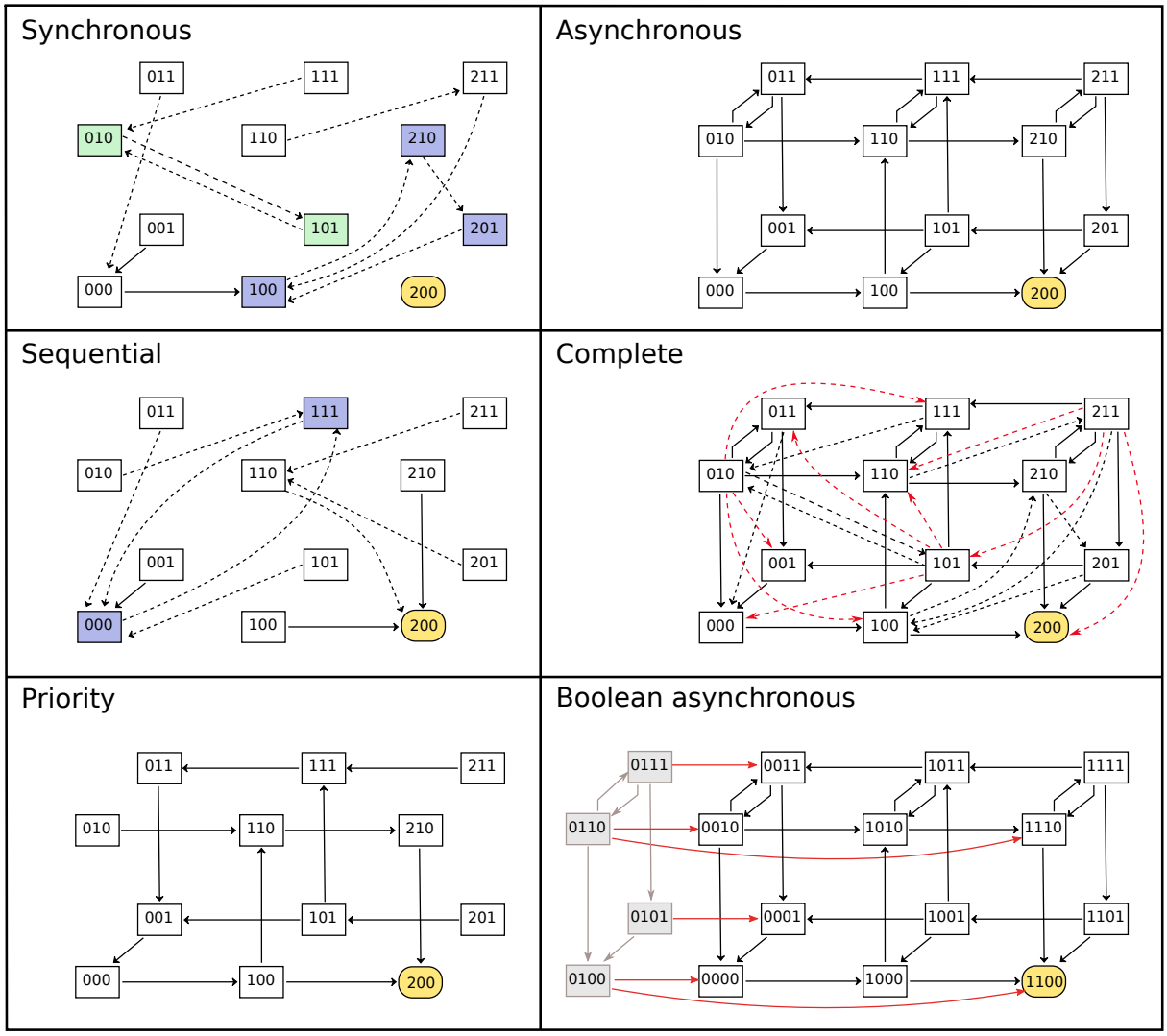


Figure 2: **Comparison of updating modes.** State transition graphs obtained with the multi-valued model shown in the top part using various deterministic (left-side panels) and non-deterministic (right-side panels) updatings. Dashed arcs denote multiple transitions and node colouring emphasises attractors. Note that the stable state is common to all updatings. The sequential and priority updaters follow the implicit ordering of the components. The STG obtained with the complete updating contains all synchronous and asynchronous transitions, as well as additional transitions to leave the states encompassing more than two updating calls. These transitions are coloured in red in the corresponding panel. Finally, the bottom-right panel contains the asynchronous STG obtained for the booleanized version of the model. In this STG, grey nodes and arcs on the left side correspond to non-admissible states and transitions between them. These states are unreachable from the admissible ones, and the transitions enabling to leave this set of states are highlighted.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

4 Model modifications

Several software tools propose to emulate biological **mutations** by generating a model variant in which one or several logical rules have been modified. In bioLQM, such perturbations are a special case of **model modifications**, which consist in computing a new, "modified", model based on an input model and some parameters if needed. While a perturbed model has the same set of components than the original one, other model modifications can have a different set of components. Each modifier can be described by a keyword (identifier for the type of modifier) and an optional string describing the parameters. Using the model modifier API, bioLQM allows to chain a variety of these modifications before model conversion or analysis. The following describes the various types of model modifications implemented in bioLQM.

Perturbations

A **perturbation** (often called **mutation**) enables to change some of the logical rules of a model. BioLQM provides three types of "atomic perturbations" (fixed value, range restriction, and removal of a regulator) which modify a single logical rule, while "multiple perturbations" combine several atomic perturbations. Atomic perturbations are briefly described below, more formal definitions can be found in appendix C. The definition of these perturbations is supported by a simple syntax, as illustrated in appendix D and described in the online documentation.

Perturbations are commonly used to model gene knockouts by **blocking the activity level** of the corresponding component to 0, or ectopic expressions by blocking it to 1. Multi-valued components can also be blocked to a higher activity level (inside their normal acticity range).

Restricting the activity level of multi-valued components to a smaller range enables to account for a partially impaired activity (loss of the higher activity level(s)) or to set a minimal activity level.

Lastly, it is possible to define the **perturbation of a single interaction**, *i.e.* to remove one of the regulators of a component. This type of perturbation enables for example the definition of the loss of a single binding site preventing the action of the source component on a subset of its targets. The removal of an interaction amounts to rewrite the logical rule of its target component. Note that the atomic perturbation describes the effect on a single target: a single "biological mutation" may correspond to a "multiple perturbation" in the model if several targets are affected by the loss of the same binding site. This type of perturbation is also convenient to evaluate the importance of an interaction representing an hypothetical effect.

Model reduction

Model reduction aims to ease the analysis of models with a large number of components by constructing a smaller model involving fewer components, but exhibiting similar dynamical properties. BioLQM provides a model reduction method which updates the logical rules of the remaining components to emulate the effect of the removed components [24, 39]. This reduction preserves key dynamical properties of the model, in particular the stable states and stable patterns. However, it can affect some dynamical properties, depending on the choice of reduced components.

This modifier usually relies on the specification of the set of components to reduce. Some types of reduction can be fully automated. In particular, bioLQM supports the reduction of output components, which was shown to preserve attractors and reachability properties [20], as well as the propagation of fixed components, which has also been shown to preserve attractors [32].

After reduction, the reduced components are not fully eliminated from bioLQM: they are no longer allowed to regulate other components, but they keep a logical rule to allow the computation of their expected value in the reduced model.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Boolean mapping of multi-valued models

As discussed above, some software tools and formats are limited to Boolean models, for example as they rely on specific theoretical results or data structures. To apply such software tools to the analysis of a multi-valued model, we can construct a Boolean model such that its dynamical properties can be transfered to the original multi-valued model.

This **model Booleanization** step is based on the Boolean mapping discussed by Didier, Remy, and Chaouiya [9]. In this mapping, a multi-valued component with a maximal activity level m is replaced by m Boolean components, each denoting increasing activity. All possible states of the original model can then be associated to states of the Boolean model. The logical rules of the new model ensure that we obtain the same transitions between these states. However, some states of the Boolean model are not mapped to states of the original model. These additional states are called "non-admissible states".

The dynamical properties observed on the admissible states of the Boolean model can be transfered to the original model. The implementation proposed here further ensures that all synchronous and asynchronous trajectories starting in with a non-admissible state end in an admissible state after a sufficient number of steps. This property ensures that no attractor contains any non-admissible state (see Figure 2).

Model Booleanization is used automatically when converting multi-valued models to formats supporting only Boolean models. It can also be performed explicitely, like other model modifications.

5 Usage

BioLQM is freely available under the LGPL v3 license. Documentation, source code and releases are available on `http://colomoto.org/biolqm`. It is implemented in the Java programming language and thus requires a Java Runtime Environment (JRE 8). A Java compiler (JDK 8) and the Maven build tool are needed to build it from source.

BioLQM is distributed as a JAR file, which can be launched with the command `java -jar bioLQM.jar`. It is also part of the `conda`[2] package for `GINsim` which was created for the CoLoMoTo Docker image [26]. This package further includes a `bioLQM` command for ease of use.

Hardware requirements strongly depend on the size and structure of models and the operations performed. The complexity of individual logical rules can be a limiting factor: components with tens of regulators could have intricate rules that scale badly. Fortunately, such rules are seldom used in biological models. Any desktop computer should be able to load and convert most models, including large ones. However, the detailled dynamical analysis of models beyond 30 components can rapidly fill the available memory.

The **command-line interface** provides an easy way to convert a model to a different format or to launch a single analysis. It further supports the definition of one or several model modifiers, allowing to save a modified model or to analyse it on the fly. Some examples of command lines are given in appendix D and in the online documentation.

More complex analysis tasks can use the integrated **scripting interface**. Based on the java scripting engine, it supports scripts written in javascript (as part of the java platform) or in another supported language by providing additional libraries (including python and lua among others). An example of script is proposed in appendix E.

[2] `https://conda.io`

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Summary and discussion

The increasing use of logical models of biological regulatory networks led to the development of multiple complementary software tools for their analysis. The recent introduction of the SBML qual format [7] and the formation of the CoLoMoTo consortium [22] aims to facilitate the exchange of models between tools. The bioLQM toolkit enables the use of additional software tools through conversion to their native formats. It provides model conversion operations in the CoLoMoTo notebook [26], enabling the delineation of analysis workflow involving a series of different tools.

BioLQM can also be used to apply various perturbations to the converted models, enabling the study of model variants emulating a knockout, an ectopic activity, or the loss of an interaction. Model modifications include the booleanization of multi-valued models for analysis with tools restricted to a Boolean formalism, as well as model reduction, decreasing the number of components to ease the analysis of complex models.

Finally, bioLQM provides several internal tools for the dynamical analysis of logical models. The first two tools allow the construction of deterministic and stochastic simulation traces, based on a comprehensive collection of updating modes. BioLQM also implements non-deterministic updating modes, which can be used as core components of complete simulation engines, as done by the GINsim [25] and Epilog [38] software suites. Two other tools enable the efficient identification of stable states and the approximation of most complex attractors.

The features described above are organized in a flexible architecture to facilitate the addition of new modules (file formats, model modifications, analysis tools) and to provide a consistent API. In the next version, the configuration API of analysis tools will be improved to provide a better integration with python scripts for use with the new CoLoMoTo notebook. Future work also includes improvements of the representation of the logical rules, which currently rely on decision diagrams (used by the current analysis tools).

Like most modelling tools, bioLQM is currently centered on logical rules, however a complete model may contain important additional information, such as annotations and graphical layout information. The later can be stored along with SBML qual models by using the SBML layout extension. This extension is currently ignored by bioLQM but supported by JSBML and GINsim, it will be supported by future versions of bioLQM. Model annotations are directly supported in SBML core (without additional extensions), however annotations can be defined in any format, hindering interoperability. Further discussions are needed within the community to delineate best practices and ensure that annotations can be shared efficiently.

The reproductibility of model analysis relies on sharing both the model itself and the definition of simulation parameters, in particular initial states and updating modes. A single initial state can be defined in the SBML qual file. Additional initial states and simulation parameters fall in the scope of the Simulation Experiment Description Markup Language (SED-ML) format [4], which does not yet support qualitative models. Ongoing discussions should lead to extensions of the SED-ML format and the Kinetic Simulation Algorithm Ontology [8] to describe model modifications and simulation parameters. These extensions will then be integrated into bioLQM and other qualitative modelling software.

Conflict of Interest Statement

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Author Contributions

AN designed and implemented the software and wrote the paper.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Funding

AN acknowledges support from the French Agence Nationale pour la Recherche (ANR), in the context of the project SCAPIN [ANR-15-CE15-0006-01].

Acknowledgments

This work benefited from the feedback of members of the CoLoMoTo consortium (colomoto.org). Pedro Monteiro, Claudine Chaouiya and Denis Thieffry provided feedback for the integration in GINsim and Epilog. Julien Dorier and Gautier Stoll helped with the BoolSim and MaBoSS formats respectively. Loic Paulevé provided feedback and implemented the Pint format. Francisco Plana implemented the block-sequential updater. Hannes Klarner implemented the CNET and bnet formats. Celine Hernandez tested the API and provided feedback.

References

[1] W. Abou-Jaoudé et al. "Logical Modeling and Dynamical Analysis of Cellular Networks". In: *Frontiers in Genetics* 7 (2016), p. 94.

[2] I. Albert et al. "Boolean network simulations for life scientists". In: *Source Code Biol. Med.* 3.1 (2008), p. 16.

[3] G. Batt et al. "Genetic network analyzer: a tool for the qualitative modeling and simulation of bacterial regulatory networks". In: *Methods Mol. Biol.* 804 (2012), pp. 439–62.

[4] F. T. Bergmann et al. "Simulation Experiment Description Markup Language (SED-ML) Level 1 Version 2." In: *J. Integr. Bioinform.* 12.2 (2015), p. 262.

[5] S. Bornholdt. "Systems biology. Less is more in modeling large genetic networks". In: *Science* 310.5747 (2005), pp. 449–51.

[6] C. Chaouiya et al. "Petri net representation of multi-valued logical regulatory graphs". In: *Nat. Comput.* 10.2 (2011), pp. 727–750.

[7] C. Chaouiya et al. "SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools". In: *BMC Syst. Biol.* 7.1 (2013), p. 135.

[8] M. Courtot et al. "Controlled vocabularies and semantics in systems biology". In: *Mol. Syst. Biol.* 7.1 (2014), pp. 543–543.

[9] G. Didier, E. Remy, and C. Chaouiya. "Mapping multivalued onto Boolean dynamics". In: *J. Theor. Biol.* 270.1 (2011), pp. 177–184.

[10] E. Dubrova and M. Teslenko. "A SAT-Based Algorithm for Finding Attractors in Synchronous Boolean Networks". In: *IEEE/ACM Trans. Comput. Biol. Bioinform.* 8.5 (2011), pp. 1393–1399.

[11] A. Fauré et al. "Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle". In: *Bioinformatics* 22.14 (2006), pp. 124–31.

[12] A. Garg et al. "Synchronous versus asynchronous modeling of gene regulatory networks". In: *Bioinformatics* 24.17 (2008), pp. 1917–1925.

[13] M. Gebser et al. "Potassco: The Potsdam Answer Set Solving Collection". In: *AI Commun.* 24.2 (2011), pp. 107–124.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

[14] T. Helikar et al. "A comprehensive, multi-scale dynamical model of ErbB receptor signal transduction in human mammary epithelial cells". In: *PLoS One* 8.4 (2013), e61757.

[15] T. Helikar et al. "The Cell Collective: toward an open and collaborative approach to systems biology". In: *BMC Syst. Biol.* 6 (2012), p. 96.

[16] S. Kauffman. "Metabolic stability and epigenesis in randomly constructed genetic nets". In: *J. Theor. Biol.* 22.3 (1969), pp. 437–67.

[17] H. Klarner, A. Bockmayr, and H. Siebert. "Computing Symbolic Steady States of Boolean Networks". In: *Cellular Automata*. Vol. 8751. Lect Notes Comput Sci. 2014, pp. 561–70.

[18] H. Klarner, A. Streck, and H. Siebert. "PyBoolNet: a python package for the generation, analysis and visualization of boolean networks." In: *Bioinformatics* 33.5 (2017), pp. 770–772.

[19] C. Müssel, M. Hopfensitz, and H. Kestler. "BoolNet–an R package for generation, reconstruction and analysis of Boolean networks". In: *Bioinformatics* 26.10 (2010), pp. 1378–1380.

[20] A. Naldi, P. Monteiro, and C. Chaouiya. "Efficient Handling of Large Signalling-Regulatory Networks by Focusing on Their Core Control". In: *Computational Methods for Systems Biology*. Lect. Notes Comput. Sci. 2012, pp. 288–306.

[21] A. Naldi, D. Thieffry, and C. Chaouiya. "Decision Diagrams for the Representation and Analysis of Logical Models of Genetic Networks". In: *Computational Methods for Systems Biology*. Vol. 4695. Lect. Notes Comput. Sci. 2007, pp. 233–247.

[22] A. Naldi et al. "Cooperative development of logical modelling standards and tools with CoLoMoTo". In: *Bioinformatics* 31.7 (2015), pp. 1154–1159.

[23] A. Naldi et al. "Diversity and plasticity of Th cell types predicted from regulatory network modelling". In: *PLoS Comput. Biol.* 6.9 (2010), e1000912.

[24] A. Naldi et al. "Dynamically consistent reduction of logical regulatory graphs". In: *Theor. Comput. Sci.* 412.21 (2011), pp. 2207–2218.

[25] A. Naldi et al. "Logical modelling and analysis of cellular regulatory networks with GINsim 3.0". In: *bioRxiv* (2018).

[26] A. Naldi et al. "The CoLoMoTo Interactive Notebook: Accessible and Reproducible Computational Analyses for Qualitative Biological Networks". In: *BioRxiv* (2018).

[27] T. J. Parr and R. W. Quong. "ANTLR: A predicated-LL(k) parser generator". In: *Software: Practice and Experience* 25.7 (1995), pp. 789–810.

[28] L. Paulevé. "Pint: A Static Analyzer for Transient Dynamics of Qualitative Networks with IPython Interface". In: *Computational Methods for Systems Biology*. Vol. 10545. Lect. Notes Comput. Sci. Springer, 2017, pp. 370–316.

[29] F. Robert. *Discrete Iterations : a Metric Study*. Springer, 1986.

[30] N. Rodriguez et al. "JSBML 1.0: providing a smorgasbord of options to encode systems biology models". In: *Bioinformatics* 31.20 (2015), pp. 3383–6.

[31] A. Saadatpour and R. Albert. "Boolean modeling of biological regulatory networks: a methodology tutorial". In: *Methods* 62.1 (2013), pp. 3–12.

[32] A. Saadatpour, R. Albert, and T. C. Reluga. "A Reduction Method for Boolean Network Models Proven to Conserve Attractors". In: *SIAM J. Appl. Dyn. Syst.* 12.4 (2013), pp. 1997–2011.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

[33] Saez-Rodriguez, J. and Simeoni, L. and Lindquist, J. and Hemenway, R. and Bommhardt, U. and Arndt, B. and Haus, U. and Weismantel, R. and Gilles, E. and Klamt, S. and Schraven, B. "A logical model provides insights into T cell receptor signaling". In: *PLoS Comput. Biol.* 3.8 (2007), e163.

[34] R. Samaga and S. Klamt. "Modeling approaches for qualitative and semi-quantitative analysis of cellular signaling networks". In: *Cell Commun. Signal* 11.1 (2013), p. 43.

[35] G. Stoll et al. "MaBoSS 2.0: an environment for stochastic Boolean modeling". In: *Bioinformatics* 33.14 (2017), pp. 2226–2228.

[36] C. Terfve et al. "CellNOptR: a flexible toolkit to train protein signaling networks to data using multiple logic formalisms". In: *BMC Syst. Biol.* 6 (2012), p. 133.

[37] R. Thomas. "Boolean formalization of genetic control circuits". In: *J. Theor. Biol.* 42.3 (1973), pp. 563–85.

[38] Varela, P and Mendes, N and Monteiro, P and Faure, A and Chaouiya, C. "EpiLog: a novel tool for the qualitative modelling of epithelial patterning". In: *INForum*. 2013.

[39] A. Veliz-Cuba. "Reduction of Boolean network models". In: *J. Theor. Biol.* 289 (2011), pp. 167–72.

[40] J. G. T. Zañudo and R. Albert. "An effective network reduction approach to find the dynamical repertoire of discrete dynamic networks". In: *Chaos* 23.2 (2013), p. 025111.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

A Logical Qualitative Model

A **Logical Qualitative Model** $\mathcal{L} = (\mathcal{G}, \mathcal{M}, \mathcal{F})$ with n components is defined by:

- $\mathcal{G} = \{g_i\}_{i=1\dots n}$, the set of its components.
- $\mathcal{M} = \{m_i \in \mathbb{N}^*\}_{i=1\dots n}$, the maximal activity levels of these components,
  $\mathcal{S}_i = [0, m_i]$ is the activity range of the component g_i, and $\mathcal{S} = \prod \mathcal{S}_i$ is the state space.
- $\mathcal{F} = \{f_i : \mathcal{S} \to \mathcal{S}_i\}_{i=1\dots n}$, the logical functions defining its dynamical behavior.

A **Boolean model** $(\mathcal{G}, \mathcal{F})$ is a logical qualitative model such that all $m_i = 1\ \forall i \in [1, n]$ (*i.e.* $\mathcal{S} = [0, 1]^n$).

B State space and dynamics

In the following, we define the state and dynamics of a model, which can be computed from its logical functions f_i. For simplicity, we use i as a shorthand for g_i when no ambiguity is possible.

A (qualitative) state $x \in \mathcal{S}$ of the model is a vector giving the activity levels of all components, where x_i denotes the activity of the component i: $x = (x_i \in \mathcal{S}_i)_{i=1\dots n}$. A component i is **called to update** in the state x if $f_i(x) \neq x_i$. The state x is a **stable state** (also called fixed point, or steady state) if no component is called to update, *i.e.* if $x_i = f_i(x)\ \forall i \in 1 \dots n$.

We define unitary update functions $u_i : \mathcal{S} \to \mathcal{S}_i$ such that $u_i(x) = x_i + \Delta_i(x)$, where

$$\Delta_i(x) = \begin{cases} 0 & \text{if } f_i(x) = x_i \\ 1 & \text{if } f_i(x) > x_i \\ -1 & \text{if } f_i(x) < x_i \end{cases}.$$

In the Boolean case, these unitary functions are identical to the logical functions f_i by construction. They enforce unitary transitions in the multi-valued case.

We call $f(x) = (f_i(x))_{i=1\dots n}$ the **image** of the state x, and $u(x) = (u_i(x))_{i=1\dots n}$ its **unitary image**.

Given a state $x \in \mathcal{S}$ and a subset of the components $p \subset \mathcal{G}$, let $\overline{x}^p$ be the state where all the components of the subset have been updated at once:

$$\overline{x}^p_i = \begin{cases} u_i(x) & \text{if } i \in p \\ x_i & \text{otherwise} \end{cases}.$$

For simplicity, $\overline{x}^i$ denotes $\overline{x}^{\{i\}}$.

Deterministic updatings

- In the **synchronous** updating, the unique successor of x is its unitary image $u(x)$.
- In the **sequential** updating following the default order of components, the unique successor of the state x is $(u_n \circ \dots \circ u_1)(x)$. A different sequential updater can be defined for each possible ordering of $\mathcal{G}$.
- A **block-sequential** updater is based on an ordered partition of $\mathcal{G}$ in m non-overlapping subsets ($m \leq n$). Let $\mathcal{P} = (p_1, \dots, p_m \mid p_k \subset \mathcal{G}\ \forall k)$ be this ordered partition. The function $v_k : \mathcal{S} \to \mathcal{S}$ such that $v_k(x) = \overline{x}^{p_k}$ updates all components of the subset p_k synchronously. The unique successor of the state x by the block-sequential updater defined by the ordered partition $\mathcal{P}$ is then given by combining these functions: $(v_m \circ \dots \circ v_1)(x)$.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

- The **synchronous priority** updating is based on the partitioning $\mathcal{P}$ defined for the block-sequential case, but only the first updated subset is considered. If the state x is not a stable state, then there exists k such that the subset p_k is the first one containing updated components: $v_k(x) \neq x$ and $v_i(x) = x \ \forall i < k$. Then $v_k(x)$ is the successor of x in this updating. Note that this type of updating is the deterministic subset of the non-deterministic priority updatings defined below.

Non-deterministic updatings

- In the **asynchronous** updating, all logical functions are applied independently and all successors of the state x differ from x by exactly one component: the set of successors of x is $\{\overline{x}^i \neq x \ \forall i \in \mathcal{G}\}$.

- In the **complete** updating, any number of components can be updated at once: the set of successors of x is $\{\overline{x}^p \neq x \ \forall p \subset \mathcal{G}\}$. This set of successors includes all asynchronous successors, as well as the synchronous one.

- The **priority** updater generalizes the *synchronous priority* defined above by considering asynchronous updates between the blocks (priority classes). Starting with $\mathcal{P} = (p_1, \ldots, p_m \mid p_k \subset \mathcal{G} \ \forall k)$, the ordered partition of $\mathcal{G}$ defined above, each subset p_k is further partitioned in a_k subsets $p_{k,1}, \ldots, p_{k,a_k}$. If the state x is not a stable state, then there exists k such that the subset p_k is the first one containing updated components: $v_k(x) \neq x$ and $v_i(x) = x \ \forall i < k$. Then $v_{k,1}(x), \ldots, v_{k,a_k}(x)$ are the a_k successors of x in this updating.

C Model modifications

Range restriction

We call $f_i^{a,b}$ the restriction of the function f_i to the range $[a, b]$ (with $0 \leq a \leq b \leq m_i$) such that:

$$f_i^{a,b}(x) = \begin{cases} a & f_i(x) < a \\ b & f_i(x) > b \\ f_i(x) & \text{otherwise: } a \leq f_i(x) \leq b \end{cases}.$$

Perturbation of an interaction

The removal of an interaction (i, j) (*i.e.* the removal of i from the regulators of j) further requires the specification of $v \in \mathcal{S}_i$ and leads to the modification of f_j, the logical rule associated to j.

Let $x^{i,v}$ be the state such that $x_i^{i,v} = v$ and $x_k^{i,v} = x_k \ \forall k \neq i$. The perturbation constructs the modified function $f_j^{i:v} = f_j(x^{i,v})$.

bioRxiv preprint doi: https://doi.org/10.1101/287011; this version posted April 5, 2018. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

D Command-line examples

Launching bioLQM without arguments displays a help message, listing all available features:

```
bioLQM
```

Converting a model in the SBML qual format to the BoolSim format:

```
bioLQM model.sbml model.boolsim
```

Loading a model, applying a knockout (of the `Akt` component) and saving the resulting modified model:

```
bioLQM model.sbml -m perturbation:Akt%0 perturbed.sbml
```

The syntax for model perturbations is further described in the online documentation. The modified model can also be saved to a different format (combining model modification and conversion):

```
bioLQM model.sbml -m perturbation:Akt%0 perturbed.boolsim
```

Loading a model and computing the stable states:

```
bioLQM model.sbml -r fixpoints
```

In this more complex example, the loaded model is perturbed before being reduced, then the stable states of the resulting model are computed:

```
bioLQM model.sbml -m perturbation:Akt%0 -m reduce:fixed -r fixpoints
```

E Scripting example

Calling bioLQM in script mode:

```
bioLQM -s generate_perturbations.js model
```

Content of the script file for the generation of all possible knockout perturbations:

```
filename = lqm.args[0]
model = lqm.load(filename)
nodes = model.getComponents()
for (i in nodes) {
    node = nodes[i]
    perturbed = lqm.modify(model, 'perturbation', node+'%0')
    lqm.save(perturbed, filename+"_"+node+"_KO.sbml")
}
```