The CoLoMoTo Interactive Notebook: Accessible and Reproducible Computational Analyses for Qualitative Biological Networks

Aurélien Naldi [1], Céline Hernandez [1], Nicolas Levy [2,3], Gautier Stoll [4–8], Pedro T. Monteiro [9], Claudine Chaouiya [10], Tomáš Helikar [11], Andrei Zinovyev [12–13], Laurence Calzone [12–13], Sarah Cohen-Boulakia [2], Denis Thieffry [1,*], Loïc Paulevé [2,*]

[1] CNRS UMR8197, INSERM U1024, École Normale Supérieure, PSL Research University, Paris, France
[2]Laboratoire de Recherche en Informatique, Université Paris-Sud, CNRS, Université Paris-Saclay, Paris, France
[3] École Normale Supérieure de Lyon, Lyon, France
[4]Université Paris Descartes/Paris V, Sorbonne Paris Cité, Paris, France
[5]Équipe 11 labellisée Ligue Nationale contre le Cancer, Centre de Recherche des Cordeliers, Paris, France
[6]Institut National de la Santé et de la Recherche Médicale, U1138, Paris, France
[7]Université Pierre et Marie Curie, Paris, France
[8]Metabolomics and Cell Biology Platforms, Gustave Roussy Cancer Campus, Villejuif, France
[9]INESC-ID/Instituto Superior Técnico, University of Lisbon, Lisbon, Portugal
[10]Instituto Gulbenkian de Ciência, Oeiras, Portugal
[11]Department of Biochemistry, University of Nebraska-Lincoln, Lincoln, NE, USA
[12]Institut Curie, INSERM U900, PSL Research University, Paris, France
[13]MINES ParisTech, PSL Research University, CBIO-Centre for Computational Biology, Paris, France
*Corresponding authors: thieffry@ibens.fr ; loic.pauleve@lri.fr

Abstract

Analysing models of biological networks typically relies on workflows in which different software tools with sensitive parameters are chained together, many times with additional manual steps. The accessibility and reproducibility of such workflows is challenging, as publications often overlook analysis details, and because some of these tools may be difficult to install, and/or have a steep learning curve. The CoLoMoTo Interactive Notebook provides a unified environment to edit, execute, share, and reproduce analyses of qualitative models of biological networks. This framework combines the power of different technologies to ensure repeatability and to reduce users' learning curve of these technologies. The framework is distributed as a Docker image with the tools ready to be run without any installation step besides Docker, and is available on Linux, macOS, and Microsoft Windows. The embedded computational workflows are edited with a Jupyter web interface, enabling the inclusion of textual annotations, along with the explicit code to execute, as well as the visualisation of the results. The resulting notebook files can then be shared and re-executed in the same environment. To date, the CoLoMoTo Interactive Notebook provides access to software tools including GINsim, BioLQM, Pint, MaBoSS, and Cell Collective for the modelling and analysis of Boolean and multi-valued networks. More tools will be included in the future. We developed a Python interface for each of these tools to offer a seamless integration in the Jupyter web interface and ease the chaining of complementary analyses.

Keywords: computational systems biology, reproducibility, model analysis, Boolean networks, Python programming language

1 Introduction

Recently, the scientific community has been increasingly concerned about difficulties in reproducing already published results. In the context of preclinical studies, observed difficulties to reproduce important findings have raised controversy (see *e.g.* [7, 15, 40, 43], and [8] for a review on this topic). Although not invalidating the findings, these observations have shaken the community. In 2016, a Nature survey pointed to the multi-factorial origin of this "reproducibility crisis" [4]. Factors related to computational analyses were highlighted, in particular the unavailability of code and methods, along with the technical expertise required to reproduce the computations.

The scientific community is progressively addressing this problem. Prestigious conferences (such as two major conferences from the database community, namely, VLDB[1] and SIGMOD[2]) and journals such as PNAS[3], Biostatistics [38], Nature [41] and Science [54], to name only a few, now encourage or even require published results to be accompanied by all the information necessary to reproduce them.

While the reproducibility challenges have first been observed in domains where deluge of data were quickly becoming available (*e.g.*, Next Generation Sequencing data analyses), the problem is now present in many (if not all) communities where computational analyses and simulations are performed. In particular, the System Biology community is facing a proliferation of approaches to perform a large variety of tasks, including the development of dynamical models, complex simulations, and multiple comparisons between varying conditions of model variants. Consequently, reproducing results from systems biology studies becomes increasingly difficult. Furthermore, although the combination of different tools would provide various new scientific opportunities, this is currently hindered by technical issues.

Several initiatives have been launched by the community to address reproducibility issues for computational modelling of biochemical networks. These include guidelines for model annotations (MIRIAM, [27]) and simulation descriptions (MIASE, [51]), as well as standards for model exchange (SBML, [23]) and simulation parametrizations (SED-ML, [52]). This collective effort is coordinated by the *COmputational Modeling in BIology NEtwork* (COMBINE[4]).

The Consortium for Logical Models and Tools (CoLoMoTo[5]) has been organized to bring together computational modelling researchers and address the aforementioned reproducibility and reusability issues within the sub-domain of logical models and software tools [33]. As a first outcome to foster model exchange and software interoperability, the SBML L3 package qual was developed [9, 10]. In this manuscript, we report the next phase of the CoLoMoTo efforts in the area of reproducibility in computational systems biology: The CoLoMoTo Interactive Notebook, which provides an easy-to-use environment to edit, execute, share, and reproduce analyses of qualitative models of biological networks by seamlessly integrating various logical modelling software tools.

The teams involved in CoLoMoTo, gathering around 50 researchers within 20 groups and laboratories, have produced various software tools for the qualitative modelling and analysis of biological networks. They are also involved in the development of novel computational methods and models. This method article presents a collective effort to provide the community with a reproducibility-oriented framework combining software tools related to logical modelling. This framework combines the power of different approaches to ensure repeatability and to reduce the requirement of technical knowledge from users. The provided Docker image facilitates the stability of a contained environment needed for repeatable computational modelling

[1] International conference on Very Large Data Bases
[2] ACM's Special Interest Group on Management Of Data.
[3] http://www.pnas.org/site/authors/format.xhtml
[4] http://co.mbine.org
[5] http://colomoto.org

and analyses. The framework includes a set of pre-installed tools from the CoLoMoTo community. On the other hand, specific binding and interfaces integrated in a Jupyter environment reduce the learning curve and improve accessibility. The use of this framework is demonstrated by a case study in a companion protocol article, which consists in a thoroughly annotated Jupiter notebook [28][6].

The method article is structured as follows. Section 2 provides a brief introduction to qualitative models of biological networks and to their analyses. Section 3 describes the main components (Docker image, Python programming interface, Jupyter interactive web interface) of our framework to facilitate the access to CoLoMoTo software tools, a prime prerequisite for the reproducibility of the computational analyses. Section 4 illustrates how our framework can address several challenges related to the reproducibility of computational analyses, ranging from the repeat of a sequence of analyses in the exact same software environment, to the use of alternate methods to reproduce a similar result. Finally, Section 5 provides an introductory guide on how to use the new framework, and Section 6 discusses possible extensions.

2 Qualitative Dynamical Models of Biological Networks

Since the pioneering work of Kauffman [24], Thomas [49], and others, logical (e.g., Boolean) models have emerged as a framework of choice to model complex biological networks, focusing for example on the roles of transcriptional regulatory circuits in cell differentiation and development, signalling pathways in cell fate decisions, etc. (for a review, see *e.g.* Abou-Jaoudé et al. [2]).

2.1 Qualitative modelling

The definition of a qualitative logical model, such as a *Boolean model* usually relies first on the delineation of a *regulatory graph*, where each *node* denotes a regulatory component (*e.g.* a protein or a gene) connected with (positive or negative) *arcs* denoting interactions (activation or inhibition) between its source and target nodes. Each node is modelled as a discrete variable, having a finite number of possible values, typically Boolean, *i.e.*, only two values, 0 or 1, denoting *e.g.* protein absence/inactivity or presence/activity. A *Boolean function* or *rule* is then defined for each node to specify how its value evolves depending on the values of its regulators.

The *state* of a network is modelled as a vector encompassing the (Boolean or multi-valued) values of all the nodes of the regulatory graph, with a prescribed ordering. The state of the network can be updated according to the logical functions defined for each node, triggering a *transition* towards a successor state. When at a given state, several nodes are called for an update, different updating modes can be considered. The *synchronous updating mode* updates all nodes simultaneously, thus leading to a unique successor state. Hence, the dynamical behaviour is fully deterministic. In contrast, the *asynchronous updating mode* updates only one node, choosen non-deterministically, thus leading to different possible successor states. Several variants and extensions of these updating mode have been defined, for instance assigning pre-determined priorities or assigning probabilities to nodes updates, or considering simlultaneous updates of sub-groups of nodes.

2.2 Dynamical analysis

The dynamical behaviour of the model can be represented by its *state transition graph*, where vertices are the different states of the network, and directed edges are the possible *transitions* between states,

[6]The notebook can be previewed and downloaded at https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/usecases/Usecase%20-%20Mutations%20enabling%20tumour%20invasion.ipynb

following a selected updating mode. *Dynamical analyses* consist then in characterizing different properties of this state transition graph.

Attractors are one of the most prominent features studied in Boolean and multi-valued networks. Attractors model the asymptotic behaviours of the system, and correspond to the *terminal strongly connected components* of the state transition graph. Attractors can be of different nature, either reduced to a single *stable state* (or *fixpoint*), from which no transition are possible, or *cyclic* sequences of states, modelling sustained oscillations. From a biological point of view, computing attractors is generally particularly relevant. The presence of multiple (disjoint) attractors can represent alternative cell fates (such as cell differentiation states), while cyclic attractors further represent periodic behaviours (such as cell cycle or circadian rhythms). The computation of attractors is addressed by different software tools, such as BIOLQM [31], GINSIM [32], PINT [36], BOOLSIM [19], BOOLEANNET [3], PYBOOLNET [25], and BOOLNET [30].

Simulations allow capturing the states which are *reachable* from a given (set of) *initial state(s)*. They can consist of random walks in the complete state transition graph and take into account updating priority schemes to distinguish fast versus slow processes and thereby obtain a simpler state transition graph [16] as implemented in the software tool BIOLQM, or user-defined transition probabilities and timing, as implemented into the software tools MABOSS [46, 47] and CELLCOLLECTIVE [22, 50].

Model checking techniques developped for software verification in computer science allow verifying formally dynamical properties on state transition graphs and are regularly employed for analysing biological systems [1, 5, 6, 13]. The properties are specified using so-called *temporal logics* which enable the formulation of queries regarding asymptotic or transient dynamical properties and taking into account all the state transitions of the model. The accordance of a Boolean/multi-valued model with such properties is verified using a general purpose model checker such as NUSMV [11] for which GINSIM and PINT provide access to.

It is worth noticing that the number of states of a Boolean or multi-valued network grows exponentially with the number of nodes. The above mentioned methods typically suffer from this complexity, and can be limited with the size of the networks (currently, mostly around fifty to a hundred of nodes, depending on the analysis and the complexity of the dynamics).

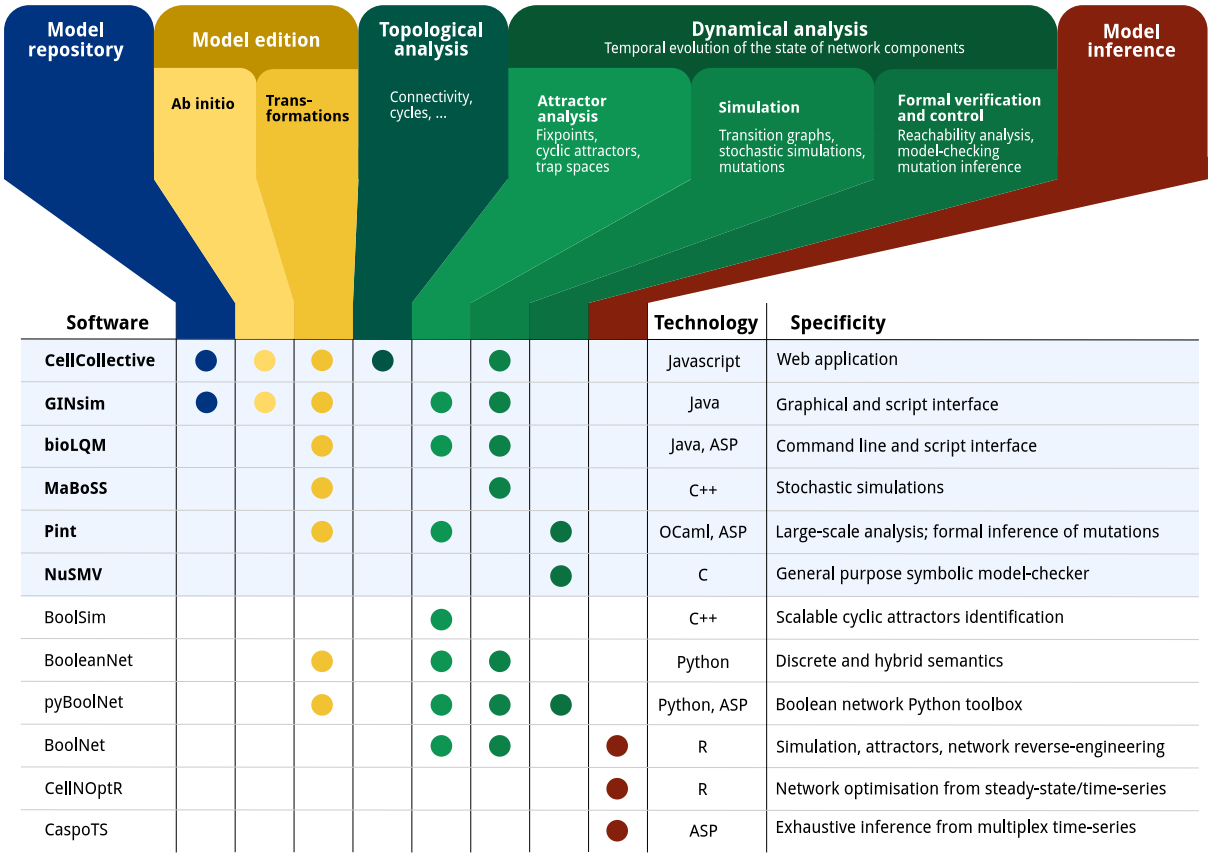
Nevertheless, different approaches enable the analysis of large scale qualitative networks, for instance by using *model reductions*, such as by Naldi et al. [34] which preserves stable states, while cyclic attractors and reachability can be affected in predictable ways, as implemented in BIOLQM, or by using *formal approximations* of dynamics, as implemented in PINT, which allow tackling networks with several thousands of nodes [36, 37].

Figure 1 gives an overview of a range of software tools for analysis of qualitative models, specifying their main features along with the main underlying technologies.

3 Accessibility of CoLoMoTo software tools

The CoLoMoTo Interactive Notebook aims at offering a unified environment for accessing a range of complementary software tools related to the analysis of qualitative models of biological networks. To achieve such a goal, our framework relies on three complementary technologies.

First, we use the Docker system to provide images of pre-installed selected CoLoMoTo software tools, thus reducing significantly the burden of installing individually each software tool. The software installed within Docker images can be executed on GNU/Linux, macOS, and Microsoft Windows, and can be



Software	Technology	Specificity
CellCollective	Javascript	Web application
GINsim	Java	Graphical and script interface
bioLQM	Java, ASP	Command line and script interface
MaBoSS	C++	Stochastic simulations
Pint	OCaml, ASP	Large-scale analysis; formal inference of mutations
NuSMV	C	General purpose symbolic model-checker
BoolSim	C++	Scalable cyclic attractors identification
BooleanNet	Python	Discrete and hybrid semantics
pyBoolNet	Python, ASP	Boolean network Python toolbox
BoolNet	R	Simulation, attractors, network reverse-engineering
CellNOptR	R	Network optimisation from steady-state/time-series
CaspoTS	ASP	Exhaustive inference from multiplex time-series

Figure 1: Feature matrix and specifities of a range of software tools related to qualitative biological networks: CellCollective [22]; GINsim [32]; bioLQM [31]; MaBoSS [46]; Pint [36]; NuSMV [11]; BoolSim [19]; BooleanNet [3]; pyBoolNet [25]; BoolNet [30]; CellNOpt [48]; CaspoTS [35]. The current CoLoMoTo Docker (2018-03-31) ships the software indicated with a bold font and a light blue background. "Model repository" refers to searchable databases of models; "Model edition" refers to the features related to creating and modifying a qualitative model, where "*ab initio*" refers to the interactive model building from scratch, and "transformations" refers to operations such as mutations, Booleanization, model reduction, etc. "Topological analysis" refers to the extraction of features from the regulatory graph, such as the different feedback cycles, graph theory measures, etc. "Dynamical analysis" refers to properties related to the state transition graph of Boolean/multi-valued networks, where "Attractor analysis" refers to the identification of stable states, cyclic attractors, and related features; "Simulation" refers to the sampling of trajectories within the state transition graph, possibly parameterized with stochastic rates and mutations; "Formal verification and control" refers to exhaustive analyses for assessing strictly temporal properties, such as reachability, and deducing mutations for controlling the system. Finally, "Model inference" refers to the derivation of Boolean/multi-valued network which are compatible with given properties and observation data.

accessed by standard workflow systems, such as SnakeMake [26].

Then, we developed a collection of Python modules to provide a unified interface to the features of the selected software tools. The Python modules allow to parametrize and execute the different analyses, and fetch their results which can be then further processed, including by a different tool through its respective Python module.

This uniform Python interface is particularly relevant in the Jupyter web interface [39], where it allows editing executable notebooks on qualitative biological networks by seamlessly combining different software tools.

3.1 The CoLoMoTo Docker image

Overall, we witness a growing ecosystem of software tools based on different technologies and offering a wide range of complementary features. Noteworthy, these tools typically rely on tailored formalisms and settings, which enable specific methods but at the same time affect the results. One obvious example is the consideration of a specific updating mode, as synchronous and asynchronous dynamics may differ extensively. Furthermore, to address increasingly large networks, many tools rely on advanced data-structures and resolution methods, which are implemented in dedicated software libraries. The distribution of these tools then become quite challenging, as they rely on numerous dependencies, some of which are difficult to install or available only for a few operating systems (most of the time GNU/Linux).

The Docker container technology allows to circumvent such distribution issues by providing a mean to supply pre-installed and fully configured software environments in so-called *Docker images*. On GNU/Linux, the execution of a Docker image consists mainly in executing the software in an isolated environment, requiring no operating system virtualization. Therefore, the overhead of using Docker on GNU/Linux is close to zero. A Docker image can also be executed on macOS or Microsoft Windows without any modification. On these operating systems, Docker relies on virtualization technologies, which are relatatively lightweight and result in limited performance loss on recent hardware.

The current CoLoMoTo Docker image `colomoto/colomoto-docker:2018-03-31` contains the following pre-installed software for the logical modelling and analsyis of biological networks: GINsim [32] ; bioLQM [31] ; CellCollective [22] ; MaBoSS [46] ; Pint [36] ; and NuSMV [11].

The CoLoMoTo Docker image then provides access to these tools without requiring any installation step beside installing Docker[7]. For instance, the Docker image can be used in association with a workflow manager to chain and run a series of software functionalities. Supplementary file "*SnakeMake*" provides an example of SnakeMake workflow relying on GINsim and NuSMV in the CoLoMoTo Docker.

An important challenge is the maintenance and extendibility of such Docker images to reduce the complexity of upgrading or adding software tools with their respective dependencies. To that aim, we require that each software tool is independently packaged for GNU/Linux using the *Conda package manager*[8]. We then rely on the dependency management system of Conda to ensure that the correct pre-requisites are installed in the Docker image[9]. A beneficial side effect of this technical choice is that the aforementioned software tools can be installed on GNU/Linux platforms using Conda, without using Docker.

[7] See `https://docker.com` for installation instructions

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

[9] CoLoMoTo-related conda packages are available in the *colomoto* conda channel. See `https://anaconda.org/colomoto`

3.2 A unified interface for calling and chaining tools with Python

The software tools considered for the CoLoMoTo Docker image present different interfaces: CellCollective is a web application, GINsim has a graphical user interface along with a scripting interface, bioLQM has a command line and a scripting interface, Pint has a command line and a Python interface, MaBoSS has a command line interface.

GINsim, CellCollective and bioLQM support the SBML-qual format, while bioLQM provides the conversion of a standard SBML-qual model into Pint or MaBoSS model formats, thereby enabling the exchange of models between all these tools.

The recourse to different interfaces complicates the design of a model analysis combining multiple tools. To address this issue, we have developed a Python interface for each of the tool embedded in the CoLoMoTo Docker image, which greatly ease the execution of different tool functionalities, fetch the results, and use these later as input for other executions.

Each tool comes with a dedicated Python module, providing a set of functions to invoke the underlying software tool appropriately. Therefore, from a single Python shell, one can invoke and chain analyses performed by different tools. This can be seen as an improved command line interface, greatly enhanced by the use of intermediate Python objects. Such an approach also promotes the use of standard Python data-structures to store objects such as model states or graphs, which can then be processed by common Python libraries, *e.g.*, Pandas[10] or NetworkX[11].

Hereafter, we give an overview of the resulting Python programming interface, focusing on the general model input mechanism and the main features implemented for each of the software tools.

Model input and tool conversions Despite their very different features, all the tools considered here take as input a logical model, in an adequate format. All the related Python modules provide a `load` function, which takes as input the location of the model, being a local file, for instance:

```
m = biolqm.load("path/to/localfile.sbml")
```

a web link to a file, as obtained on GINsim repository[12] for instance:

```
m = biolqm.load("http://ginsim.org/sites/default/...MamCC_Apr2016.sbml")
```

or a web link to the model on CellCollective, for instance:

```
m = biolqm.load("https://cellcollective.org/#5128/lac-operon")
```

In each case, the returned object (identified by `m` in the above examples) is a Python object representing the loaded model and defined specifically for the corresponding tool (Python module). Table 1 lists the supported input format for each software tool.

When possible, Python modules provide functions to convert a model for a compatible tool. These functions are of the form `moduleA.to_moduleB(modelA)`. Figure 2 lists the currently supported model conversions. The following Python code shows an example of usage:

```
lrg = ginsim.load("http://ginsim.org/sites/default/...MamCC_Apr2016.zginml")
lqm = ginsim.to_biolqm(lrg)
an = biolqm.to_pint(lqm)
```

Here, `lrg` is a Python object representing a GINsim model, `lqm` is a Python object representing a bioLQM model, and `an` is a Python object representing a Pint model.

[10] https://pandas.pydata.org/
[11] http://networkx.github.io
[12] http://ginsim.org/models_repository

Software tool	Supported input formats
BIOLQM	SBML-qual (`.sbml`), raw logical functions, truth table
GINSIM	GINML (`.ginml`, `.zginml`)
PINT	Automata network (`.an`)
MABOSS	Dedicated network/configuration files (`.bnd`/`.cfg`)
NUSMV	SMV file (`.smv`)

Table 1: Model input formats for the software tools included in the CoLoMoTo Docker image

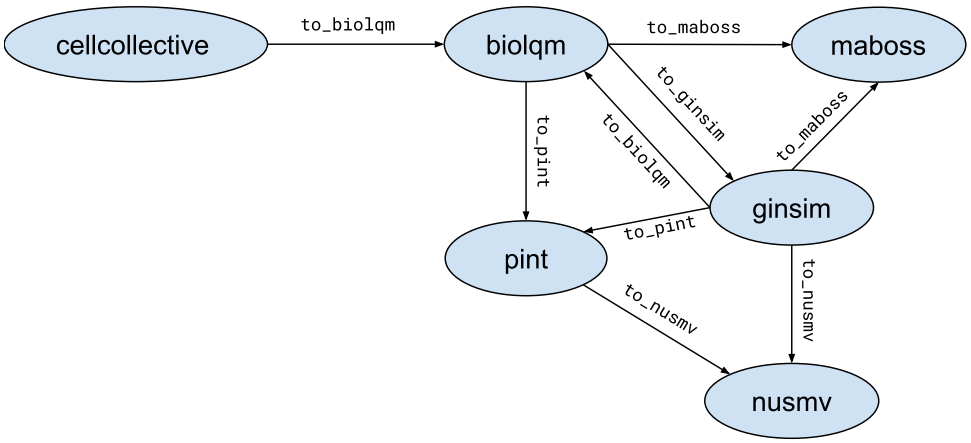


Figure 2: Supported conversion between Python module models

CellCollective– Modelling platform, repository, and knowledge base The `cellcollective` Python module allows connecting to the CELLCOLLECTIVE [22] web application (`https://www.cellcollective.org`), in order to download the models in SBML-qual format and extract network nodes meta-data (*e.g.*, UnitProt identifiers) when available.

The supplementary file "*Notebooks/demo-cellcollective*"[13] provides a brief demonstration of the Python module usage.

GINsim– Regulatory network modelling The `ginsim` Python module provides direct access to the Java programming interface of GINSIM [32]. GINSIM is available and documented at `http://www.ginsim.org`. In particular, besides the export of a GINsim model into various file formats, the Python module allows to visualise the network regulatory graph, with the activation and inhibition relationships between the nodes. The visualisation function (`ginsim.show`) optionally takes as argument a Python dictionary associating a level with each node; then, the nodes of the network are coloured according to these levels.

This is illustrated in the supplementary file "*Notebooks/demo-ginsim*"[14].

bioLQM– Qualitative model toolbox The `biolqm` Python module provides direct access to the Java programming interface of BIOLQM [31]. BIOLQM is available and documented at `http://colomoto.org/biolqm`. BIOLQM supports the conversion of SBML-qual files, GINML files, as well as simple textual files specifying the raw logical functions into the formats associated with the different software tools.

[13]The notebook can be previewed and downloaded at `https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/tutorials/CellCollective/CellCollective%20-%20Knowledge%20Base.ipynb`

[14]The notebook can be previewed and downloaded at `https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/tutorials/GINsim/GINsim%20-%20visualization.ipynb`

Besides the file format features, BIOLQM implements *model modifications*, such as mutations forcing the value of given nodes, iterative model reduction(see above), model reversal, the conversion of multi-valued model into Boolean ones, as well as the computation of *stable states*, *trap spaces*, and *simulations*.

Part of these features are illustrated in the supplementary file "*Notebooks/demo-biolqm*"[15].

Pint– Formal predictions for controlling trajectories The `pypint` Python module provides complete access to PINT [36] features documented at `https://loicpauleve.name/pint`. The software PINT is devoted to the analysis of trajectories in very large-scale asynchronous Boolean and multi-valued networks. Its main features include the verification of the existence of a trajectory reaching a state of interest (*reachability*), the identification of common points between *all* the trajectories leading to a state of interest (*cut sets*), and the *formal prediction of mutations* preventing the existence of any trajectory to the given state.

These features are illustrated in the supplementary file "*Notebooks/demo-pint*"[16].

NuSMV– Model verification The `nusmv` Python module provides a simple interface to the NUSMV model checker [11] for verifying LTL (trace) and CTL (computation tree) temporal logic properties. The specification of LTL and CTL properties can be facilitated using the `colomoto.temporal_logics` Python module, which takes advantage of Python objects for the different logical operators and ease their combination.

Let us consider the following example using the CTL operators from the aforementioned Python module:

```
p1 = AG (S(a=1))
p2 = EF (p1)
```

Here, the variable `p1` is a CTL formula specifying that the node a is active (`S(a=1)`) in all the reachable states (`AG` operator). The variable `p2` is a CTL formula specifying that there exists a trajectory leading to a state (`EF` operator) from which the property `p1` is verified.

In the above example, `S` specifies a property on a state, by giving the values of some nodes of the network. The conversion of a network model into NUSMV format depends on the tool used, sometimes introducing different variable names for the nodes of the original biological network. But this technical point is transparent for the user: the `nusmv` Python module will automatically translate the node names into the correct NUSMV variable names.

The supplementary file "*Notebooks/demo-nusmv*"[17] gives a simple example of usage of the `nusmv` Python module to verify properties of a GINSIM model.

MaBoSS– Stochastic simulations The `maboss` Python module provides an interface to MABOSS [46], available at `https://maboss.curie.fr`, as well as basic plotting functionalities. The purpose of MABOSS is to perform stochastic simulations of a Boolean network where the propensity of transitions (probabilistic rates) are explicitly specified. The Python module allows to fully define and parameterize a model, as well as parsing an existing MaBoSS model and modify it programmatically. The object returned after the simulations can then be used to plot the probability of state activation over time, and

[15]The notebook can be previewed and downloaded at https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/tutorials/bioLQM/bioLQM_tutorial.ipynb

[16]The notebook can be previewed and downloaded at https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/tutorials/Pint/quick-tutorial.ipynb

[17]The notebook can be previewed and downloaded at https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/tutorials/NuSMV/NuSMV%20with%20GINsim.ipynb

the proportion of states in which the simulations ended, in order to estimate the probability of reaching different attractors.

The supplementary file "*Notebooks/demo-maboss*"[18] provides a brief tutorial to the main features of the `maboss` Python module.

Advanced combinations of tools These Python modules provide a unified interface to chain different tools and process their results. The small tutorials referenced above show simple chaining of tools, most of the time using a tool to import a model (e.g. from CellCollective or GINsim) and convert it (using bioLQM) for specific analysis by another tool.

As the Python functions of the different modules relies on standard Python data-structures, such as list and dictionaries, it is possible to easily re-use the result from a tool function as input to the function of a different tool. A simple example is provided in supplementary file "*Notebooks/demo-ginsim*"[19], where we use bioLQM to compute the fixpoints of a GINsim model, and then give one of the resulting state as input to GINsim `show` function to display the computed fixpoint in the regulatory graph.

Moreover, one can use the programmatic features of Python to implement advanced algorithms for executing multiple analyses and process their result. For instance, one can program loops to iterate over a list of results of a preceding analysis from one tool to perform a subsequent analysis on each result with another tool. This is illustrated in the supplementary file "*Notebooks/demo-pint+maboss*"[20] where we use Pint to formally predict combinations of mutations controlling the existence of trajectories towards a specified state; then, we quantify with MaBoSS the efficiency of applying only partially the predicted combinations, by evaluating each related double-mutants. The example involves Python `for` loops and a function to enumerate all possible subsets provided by the standard Python library. The notebook also relies on CellCollective to fetch the model, and on bioLQM to perform the adequate model conversions.

3.3 CoLoMoTo Jupyter interactive notebook

Jupyter[21] is a software providing an interactive web interface for creating documents, called *notebooks*, mixing code, equations, and formatted texts. A notebook typically describes a full analysis workflow, combining textual explanations, the code itself, along with parameters to reproduce the results. A notebook is a single file that can be easily modified, shared, re-executed and visualised online. The short tutorials of the previous section provided in the supplementary file "*Notebooks*" are actually Jupyter notebooks (files with the extension `.ipynb`) and can be re-executed using Jupyter.

A Jupyter notebook is sequence of so-called *cells*, which can contain formatted text, including sections, links, images, tables, etc., or which can contain code in a specified programming language, typically Python. A code cell can be executed (by pressing Shift-Enter) and the value returned by the code is displayed below the cell. The display format is selected according to the type of the returned value (image, graph, list, table, ...) to offer an adequate visualisation.

Having a unified Python interface to invoke the CoLoMoTo software tools, one can directly create Jupyter notebooks for the analysis of qualitative biological networks using these tools, as shown in

[18] The notebook can be previewed and downloaded at https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/tutorials/MaBoSS/MaBoSS%20-%20Quick%20tutorial.ipynb

[19] The notebook can be previewed and downloaded at https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/tutorials/GINsim/GINsim%20-%20visualization.ipynb

[20] The notebook can be previewed and downloaded at https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/tutorials/MaBoSS/Predict%20mutations%20with%20Pint,%20refine%20with%20MaBoSS.ipynb

[21] http://jupyter.org

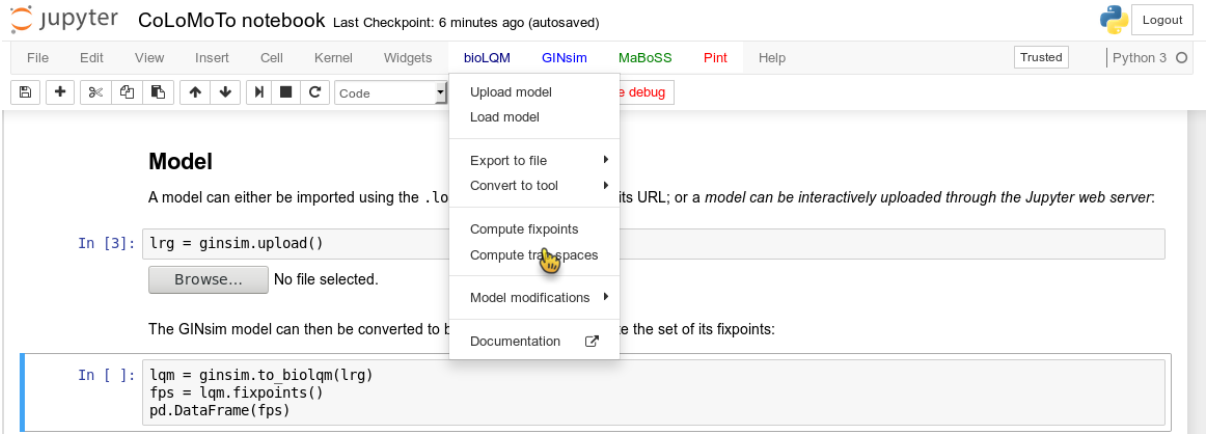


Figure 3: Screenshot during a Jupyter notebook edition showing the menu of the BIOLQM tool

supplementary file "*Notebooks*" and in the companion publication [28] providing a complete model analysis workflow.

To improve the user experience for editing Jupyter notebooks, we added several features in the CoLoMoTo Python modules, which increase interactivity. First, menus provide pre-defined Python code for accessing to the main features of the tools. Figure 3 shows a screenshot during the edition of a Jupyter notebook with its graphical interface. Next, we added the possibility to interactively upload a model file. This feature is particularly useful when used in combination with Docker or on a remote server with no direct access to the user file system. Finally, some Python modules, in particular `maboss` Python model, provides JavaScript widgets to generate Python code interactively.

The Jupyter notebook server is included in the CoLoMoTo Docker image (see the Discussion section for a quick usage guide), while a public demonstration web instance is available at `http://tmpnb.colomoto.org`.

4 Reproducibility of computational analyses

4.1 From repeatability to reproducibility

The literature provides a range of definitions for the reproducibility of *in silico* experiments by analogy to wet lab experiments [12, 14, 17, 18, 21, 29, 44]. Four *levels* of reproducibility are then commonly distinguished.

An *in silico* experiment is said to be **repeated** when it is performed using the same computational set-up as the original experiment. The major goal of the *repeat* task is to check whether the initial experimental result was correct and can be obtained again. The difficulty lies in recording as much information as possible to repeat the experiment so that the same conclusion can be drawn. Interestingly, [17] discusses the granularity at which information (experiments, data sets, parameters, environment) should or could be recorded and underlines the fact that the key point is to determine the right balance between the effort required to record information and the capability of obtaining identical results.

An *in silico* experiment is said to be *replicated* when it is performed in a new setting and computational environment, although similar to the original ones. When it can be successfully replicated, a result has a high level of robustness: it remained valid when using a similar (although different) protocol. A continuum of situations can be considered between repeated and replicated experiments.

A result is then defined as ***reproduced***, in the broadest possible sense of the term by denoting the situation where an experiment is performed within a different environment, with the aim to validate the same scientific hypothesis. In other words, what matters here is the conclusion obtained and not the methodology considered reaching it. Completely different approaches can be designed, different data sets can be used, as long as the experiments support the same scientific conclusion. A reproducible result is thus a high-quality result, confirmed in various ways.

A last very important concept related to reproducibility is that of ***reuse***, which denotes the case where a *different* experiment is performed, with similarities with an original experiment. A specific kind of reuse occurs when a single experiment is reused in a new context (and thus adapted to new needs), the experiment is then said to be *repurposed*.

It is worth noticing that *repeating* and *replicating* may appear to be technical challenges compared to *reproducing* and *reusing*, which are the most important scientific objectives. However, before investigating alternative ways of obtaining a result (to reach reproducibility) or before reusing a given methodology in a new context (to reach reuse), the original experiment has to be carefully tested especially by reviewers or any peer), demonstrating its ability to be at least repeated and hopefully replicated [17, 45].

4.2 Repeat analysis in the same software environment

Ensuring that a sequence of computational analyses can be repeated by other scientists several months or years after its publication is difficult. Indeed, besides software availability, the version of the tools can be crucial: a new version of a tool can change the default parameters and even some features so that the published instructions become obsolete.

Whereas a Docker image addresses efficiently the issue of making software available, providing a safe way for repeating a notebook content years after its creation requires additional technical procedures.

First, CoLoMoTo Docker images are constructed by specifying explicitly the version of each software. Furthermore, an automatic validation procedure is performed by checking that a set of notebooks still execute without error. Once validated, the Docker image is then tagged with a time-stamp, typically the date of the image validation (of the form `YYYY-MM-DD`, *e.g.*, 2018-03-31). These tagged images are then stored in the public Docker image registry, and can be retrieved any time later. The list of existing tags of `colomoto/colomoto-docker` Docker images can be viewed at `https://hub.docker.com/r/colomoto/colomoto-docker/tags/`.

When sharing a notebook, and notably when attaching it to a publication, it is highly recommended specifying the time-stamp of the Docker image in which the notebook has been executed. Then, by downloading the image with this specific tag, it is ensured that other users will repeat the execution in exactly the same software environment.

To help to follow this recommendation, we took two technical decisions. First, we do not use the default non-persistent tag for Docker images (*latest*). It means that the user has to always specify explicitly the time-stamp of the CoLoMoTo Docker image. To remove the burden of actively checking the list of existing time-stamps, we provide a script which, by default, will fetch the most recent Docker image (see Section 5). Second, when loading a CoLoMoTo-related Python module within a Docker container, a textual message indicating the time-stamp of the Docker image is displayed. Therefore, when created within a CoLoMoTo Docker image, notebooks always contain the required information to repeat their execution.

Because a Jupyter notebook is a single file containing everything to execute it, one can easily check if it can be *replicated* in a different software environment, *i.e.*, using a more recent CoLoMoTo Docker image.

Moreover, a notebook can be easily *repurposed* by modifying some arguments of the Python function calls, for instance changing the input model or analysis parameters. One can even define interactive notebooks describing a common model analysis, so that the user only need to provide the input model and execute the Jupyter code cells, as shown in the Supplementary file "*Notebooks/demo-interactive-fixpoints*"[22] for the computation and visualisation of the stable states of a bioLQM model.

4.3 Reproduce analysis with a different method

Reproducing a same analysis with two different methods is a good mean to increase confidence in the results, as it reduces the chance of software misuse or that the results are affected by a software bug.

The subset of software tools selected for this first CoLoMoTo Docker image presentation already provides redundant implementations of equivalent model analyses, in particular for the identification of stable states and for the verification of temporal properties with NuSMV. To help switch between two tools for performing the same task, we harmonised the usage of Python module functions to ensure that the same functions with the same arguments generate equivalent results with different tools.

4.3.1 Stable states

There exists several methods to compute the full set of stable states (or fixed points) of a logical model, relying on different data-structures and different algorithms.

The software bioLQM implements the computation of stable states for Boolean and multi-value logical model using a Java implementation of decision diagrams. In contrast, the software Pint implements the computation of stable states of Automata networks (a generalisation of logical networks) using Boolean satisfaction constraints. As bioLQM provides a conversion of Boolean/multi-valued network into equivalent Automata networks, it is possible to compute the stable states of a model with both software tools.

Both `biolqm` and `pypint` Python modules provide a `fixpoint` function which take as input the model instance of the corresponding tool, and both returns a list of Python dictionaries describing the states being fixpoints. Provided `lqm` is a bioLQM model, the following Python code compute its stable states with both tools:

```
fps_biolqm = biolqm.fixpoints(lqm)
fps_pint = pypint.fixpoints(biolqm.to_pint(lqm))
```

The supplementary file "*Notebooks/demo-reproducibility-fixpoints*"[23] shows a complete example of reproduction of stable state computation using BioLQM and Pint.

4.3.2 Temporal property verification (model-checking)

Both GINsim and Pint allow to export their respective model into NuSMV format, where temporal properties can be specified using LTL or CTL (see Section 2.2). However, as the input formalism for these tool relies on different paradigms (logical rule-centred specification in the case of Boolean/multi-valued networks in GINsim; transition-centred specification (à la Petri nets) in the case of Automata networks in Pint), the generated NuSMV models have different features. Nevertheless, in the appropriate settings, the verification of an equivalent CTL or LTL property should give the same result.

[22] The notebook can be previewed and downloaded at https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/tutorials/bioLQM/Fixpoints%20(interactive).ipynb

[23] The notebook can be previewed and downloaded at https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/tutorials/Reproducibility%20-%20fixpoints.ipynb

Hence, the functions `ginsim.to_nusmv` and `pypint.to_nusmv` are implemented in such ways that, when using their default options, the resulting NuSMV models, albeit different, should produce identical results for identical temporal logic properties. Note, however, that each tool provides specific options for the NuSMV export, which can lead to incomparable results.

The following Python code uses operators defined in the Python module `colomoto.temporal_logics` to specify a property `p` meaning that from any state, there always exists a trajectory leading to a cyclic attractor where the level of node a can always oscillate. Then, assuming `lrg` is a GINsim model, the code uses GINsim and Pint conversions to NuSMV to perform model verification.

```
p = EF (AG (EF (S(a=0)) & EF (S(a=1))))

nusmv_ginsim = ginsim.to_nusmv(lrg)
nusmv_ginsim.add_ctl(p)
nusmv_ginsim.verify()

nusmv_pint = pypint.to_nusmv(ginsim.to_pint(lrg))
nusmv_pint.add_ctl(p)
nusmv_pint.verify()
```

Note that the Python object `p` represents the CTL property to be tested, whatever the origin of the model (GINsim or Pint).

The supplementary file "*Notebooks/demo-reproducibility-modelchecking*"[24] provides a more detailed example of the reproduction of model-checking results using GINsim and Pint.

5 Quick-usage guide

On GNU/Linux, macOS, or Microsoft Windows, provided that Docker and Python are installed, a helper script to run the CoLoMoTo Docker image and the embedded Jupyter notebook can be installed and upgraded from a terminal using the following command[25]:

```
pip install -U colomoto-docker
```

The Docker image and the Jupyter notebook interface can be started by executing the following command in a terminal[26]:

```
colomoto-docker
```

Without any argument, the command will use the most recent CoLoMoTo Docker image. To use the image at a specific tag, append the `-V` option (*e.g.*, `colomoto-docker -V 2018-03-31`).

The execution of this command will open a web page with the Jupyter notebook interface, enabling loading and execution of notebooks. A new notebook can be created by using the "New/Python3" menu. In this environment, the user has access to all CoLoMoTo Python modules. A code cell is executed by typing "Shift+Enter". The menu and toolbar allow quick access to main Jupyter functionalities.

Warning: by default, the files within the Docker container are isolated from the running host computer, and are deleted when stopping the container. To have access to the files of the current directory of the host computer, the option `--bind` can be used:

[24] The notebook can be previewed and downloaded at https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/blob/2018-03-31/tutorials/Reproducibility%20-%20model%20checking.ipynb

[25] you may have to use `pip3` instead of `pip` depending on your configuration

[26] If using Docker Toolbox, the command should be executed within the Docker Terminal

```
colomoto-docker --bind .
```

The container can later be stopped by pressing Ctrl+C keys in the terminal. See `colomoto-docker --help` for other options. Additional documentation for running the CoLoMoto Docker image can be found at https://github.com/colomoto/colomoto-docker.

6 Discussion

Academic use cases

The prime aim of the CoLoMoTo Interactive Notebook is to assist with the production of accessible and reproducible computational analysis of biological models, with a focus on qualitative models, including Boolean and multi-valued networks. As demonstrated in supplementary file "*SnakeMake*", the CoLoMoTo Docker image can also be used in standard workflow systems, such as SnakeMake, to lighten the burden of installing the different software tools and make them accessible on different operating systems.

A notebook issued from the CoLoMoTo Docker image gives some guarantees of repeatibility, as it will contain references to the persistent Docker image to re-execute it in the same software environment. Therefore, the notebook file (with `.ipynb` extension) can be distributed as a supplementary file of the related scientific article with instruction to run the Docker image. The Jupyter interface allows to export the notebook in a static HTML file, which could also be joined to the supplementary file to provide a quick way to visualize it.

A notebook can also be distributed independently, for instance by publishing it on *Gist*[27] or *myExperiment*[28] [20], which allow to visualise download, and follow potential updates of the notebook. For instance, the tutorial notebook presented by Levy et al. [28] is hosted at https://gist.github.com/pauleve/a86717b0ae8750440dd589f778db428f. Services like Zenodo[29] also allow creating persistent DOI links to a specific version of a notebook file.

The CoLoMoTo Interactive Notebook is also relevant for teaching purposes. With Jupyter, students can straightforwardly execute, modify, and extend a template notebook to learn methods for analysing models of biological networks. Docker is a standard technology very often supported by local cloud infrastructures, which can therefore provide dedicated resources to execute remotely and privately the CoLoMoTo Jupyter web interface.

Extending the CoLoMoTo Interactive Notebook

The CoLoMoTo Docker image can be easily extended to include additional tools. The Docker architecture allows inheriting from an existing container, adding a new layer with additional executables. Contributions are welcome through GitHub[30]. The software tool must be usable from the Jupyter interface and should be able to connect with at least one other tool already included. Furthermore, a demonstration notebook should be provided to illustrate the tool usage and how it can be combined with other tools.

Currently, all the embedded tools require an already defined model. Nevertheless, once loaded, a model can be subsequently modified from the Python interface (see tool feature matrix in Figure 1). We are currently considering the development of a programmatic interface for model definition *ab initio*. One of the main challenge is to provide a decent visualisation of the programmatically-created model.

[27] https://gist.github.com
[28] https://www.myexperiment.org
[29] https://zenodo.org
[30] Guidelines available at https://github.com/colomoto/colomoto-docker/blob/master/CONTRIBUTING.md

A potential direction is to include a visual edition module in the Jupyter interface, which represents a substantial development effort.

The support for standard exchange formats is key to enable reproducibility of analyses with different tools. In that sense, BIOLQM plays an important role for the CoLoMoTo Interactive Notebook as it provides bridges between SBML-qual standard specifications and numerous software tools (Figure 2). The Tellurium Notebook system by [42] offers support for SED-ML to help reproduce quantitative simulation of biological networks. Future work should consider bringing this feature for qualitative models as well, in order to better meet FAIR (Findability, Accessibility, Interoperability, and Reusability) recommendations [53].

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, CH, DT and LP designed the main principles of the CoLoMoTo Interactive notebook and its distribution. AN, CH, NL and LP implemented the necessary Python modules, their integration in the Jupyter interface, and the Docker image. AN and LP edited the notebook tutorials, while CH edited the SnakeMake workflow example. All authors contribute to the writing of the article under the supervision of DT and LP. All authors reviewed the content of this article and agreed to endorse it.

Funding

DT and CH acknowledge support from the French Plan Cancer (2014–2017), in the context of the projects CoMET and SYSTAIM. DT and AN acknowledge support from the French Agence Nationale pour la Recherche (ANR), in the context of the project SCAPIN [ANR-15-CE15-0006-01]. CC and PTM acknowledge support from the Fundação para a Ciência e a Tecnologia, through grants PTDC/BEX-BCB/0772/2014 and PTDC/EEI-CTP/2914/2014. TH acknowledges support from the National Institutes of Health (#5R35GM119770-02). SCB acknowledges support from CNRS (défi Mastodons). AZ, and LC acknowledge support from COLOSYS project in EU ERACoSysMed programme. AZ, LC, and LP acknowledge support from ANR in the context of the project ANR-FNR project AlgoReCell [ANR-16-CE12-0034]. LP and SCB acknowledge support from Paris Ile-de-France Region (DIM RFSI) and Labex DigiCosme [ANR-11-LABEX-0045-DIGICOSME] operated by ANR as part of the program "Investissement d'Avenir" Idex Paris-Saclay [ANR-11-IDEX-0003-02].

Acknowledgments

The authors would like to thank the attendees of the fourth CoLoMoTo meeting in Paris, July 17-19, for the insightful discussions which led to designing the CoLoMoTo Interactive notebook.

Notebook file name	Software tools involved
demo-cellcollective	CellCollective, bioLQM
demo-ginsim	GINsim, bioLQM
demo-biolqm	bioLQM
demo-interactive-fixpoints	bioLQM
demo-pint	Pint
demo-nusmv	GINsim, NuSMV
demo-maboss	MaBoSS
demo-pint+maboss	CellCollective, bioLQM, Pint, MaBoSS
demo-reproducibility-fixpoints	GINsim, bioLQM, Pint
demo-reproducibility-modelchecking	GINsim, Pint

Table 2: List of notebook files in supplemental data "*Notebooks*" demonstrating some features of the CoLoMoTo Interactive Notebook

Supplemental Data

The supplemental data "*SnakeMake*" contains an example of SnakeMake workflow which uses the CoLoMoTo Docker image to execute the different analyses.

The supplemental data "*Notebooks*" contains several short Jupyter notebooks which demonstrate different usage of the CoLoMoTo interactive notebook, listed in Table 2. The `.ipynb` files can be imported and executed within the Jupyter interface of the CoLoMoTo notebook, using the Docker image `colomoto/colomoto-docker:2018-03-31`. For each notebook file is included a static HTML file to quickly preview the Jupyter rendering of the notebook, without any requirement. These notebooks can also be previewed and downloaded at https://nbviewer.jupyter.org/github/colomoto/colomoto-docker/tree/2018-03-31/tutorials.

References

[1] Abou-Jaoudé, W., Monteiro, P. T., Naldi, A., Grandclaudon, M., Soumelis, V., Chaouiya, C., et al. (2015). Model checking to assess t-helper cell plasticity. *Front. Bioeng. Biotechnol.* 2, 86. doi:10.3389/fbioe.2014.00086

[2] Abou-Jaoudé, W., Traynard, P., Monteiro, P. T., Saez-Rodriguez, J., Helikar, T., Thieffry, D., et al. (2016). Logical Modeling and Dynamical Analysis of Cellular Networks. *Front. Genet.* 7, 94. doi:10.3389/fgene.2016.00094

[3] Albert, I., Thakar, J., Li, S., Zhang, R., and Albert, R. (2008). Boolean network simulations for life scientists. *Source Code Biol. Med.* 3, 16. doi:10.1186/1751-0473-3-16

[4] Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature News* 533, 452. doi:10.1038/533452a

[5] Bartocci, E. and Lió, P. (2016). Computational modeling, formal analysis, and tools for systems biology. *PLOS Computational Biology* 12, 1–22. doi:10.1371/journal.pcbi.1004591

[6] Batt, G., Ropers, D., de Jong, H., Geiselmann, J., Mateescu, R., Page, M., et al. (2005). Validation of qualitative models of genetic regulatory networks by model checking: analysis of the nutritional stress response in escherichia coli. *Bioinformatics* 21, i19–i28. doi:10.1093/bioinformatics/bti1048

[7] Begley, C. G. and Ellis, L. M. (2012). Drug development: Raise standards for preclinical cancer research. *Nature* 483, 531–533. doi:10.1038/483531a

[8] Begley, C. G. and Ioannidis, J. P. (2015). Reproducibility in science improving the standard for basic and preclinical research. *Circ. Res.* 116, 116–126. doi:10.1161/CIRCRESAHA.114.303819

[9] Chaouiya, C., Bérenguier, D., Keating, S. M., Naldi, A., van Iersel, M. P., Rodriguez, N., et al. (2013). SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools. *BMC Syst. Biol.* 7, 135. doi:10.1186/1752-0509-7-135

[10] Chaouiya, C., Keating, S. M., Berenguier, D., Naldi, A., Thieffry, D., van Iersel, M. P., et al. (2015). The Systems Biology Markup Language (SBML) level 3 package: Qualitative models, version 1, release 1. *J. Integr. Bioinform.* 12, 270. doi:10.2390/biecoll-jib-2015-270

[11] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., et al. (2002). NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)* (Springer), vol. 2404 of *LNCS*. doi:10.1007/3-540-45657-0_29

[12] Cohen-Boulakia, S., Belhajjame, K., Collin, O., Chopard, J., Froidevaux, C., Gaignard, A., et al. (2017). Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. *Future Gener. Comput. Syst.* 75, 284–298. doi:10.1016/J.FUTURE.2017.01.012

[13] Collombet, S., van Oevelen, C., Sardina Ortega, J. L., Abou-Jaoudé, W., Di Stefano, B., Thomas-Chollier, M., et al. (2017). Logical modeling of lymphoid and myeloid cell specification and transdifferentiation. *Proc. Natl. Acad. Sci.* 114, 5792–5799. doi:10.1073/pnas.1610622114

[14] Drummond, C. (2009). Replicability is not reproducibility: nor is it good science. In *Proc. of the Evaluation Methods for Machine Learning Workshop at the 26th ICML*

[15] Errington, T. M., Iorns, E., Gunn, W., Tan, F. E., Lomax, J., and Nosek, B. A. (2014). An open investigation of the reproducibility of cancer biology research. *Elife* 3, e04333. doi:10.7554/eLife.04333

[16] Fauré, A., Naldi, A., Chaouiya, C., and Thieffry, D. (2006). Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. *Bioinformatics* 22, 124–31. doi:10.1093/bioinformatics/btl210

[17] Freire, J., Bonnet, P., and Shasha, D. (2012). Computational reproducibility: state-of-the-art, challenges, and database research opportunities. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data* (ACM), 593–596. doi:10.1145/2213836.2213908

[18] Freire, J., Fuhr, N., and Rauber, A. (2016). Reproducibility of data-oriented experiments in e-science. In *Dagstuhl Seminar 16041*. 108–159. doi:10.4230/DagRep.6.1.108

[19] Garg, A., Di Cara, A., Xenarios, I., Mendoza, L., and De Micheli, G. (2008). Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics* 24, 1917–1925. doi:10.1093/bioinformatics/btn336

[20] Goble, C. A., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D., Newman, D., et al. (2010). myExperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic acids research* 38, W677–W682

[21] Goodman, S. N., Fanelli, D., and Ioannidis, J. P. (2016). What does research reproducibility mean? *Sci. Transl. Med.* 8, 341ps12. doi:10.1126/scitranslmed.aaf5027

[22] Helikar, T., Kowal, B., McClenathan, S., Bruckner, M., Rowley, T., Madrahimov, A., et al. (2012). The Cell Collective: toward an open and collaborative approach to systems biology. *BMC Syst. Biol.* 6, 96. doi:10.1186/1752-0509-6-96

[23] Hucka, M., , Finney, A., , Sauro, H. M., , et al. (2003). The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* 19, 524–531. doi:10.1093/bioinformatics/btg015

[24] Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theor. Biol.* 22, 437–467. doi:10.1016/0022-5193(69)90015-0

[25] Klarner, H., Streck, A., and Siebert, H. (2017). PyBoolNet: a python package for the generation, analysis and visualization of Boolean networks. *Bioinformatics* 33, 770–772. doi:10.1093/bioinformatics/btw682

[26] Köster, J. and Rahmann, S. (2012). Snakemake - a scalable bioinformatics workflow engine. *Bioinformatics* 28, 2520–2522. doi:10.1093/bioinformatics/bts480

[27] Le Novère, N., Finney, A., Hucka, M., Bhalla, U. S., Campagne, F., Collado-Vides, J., et al. (2005). Minimum information requested in the annotation of biochemical models (MIRIAM). *Nat. Biotechnol.* 23, 1509–15. doi:10.1038/nbt1156

[28] Levy, N., Naldi, A., Hernandez, C., Stoll, G., Thieffry, D., Zinovyev, A., et al. (submitted). Prediction of Mutations to Control Pathways Enabling Tumour Cell Invasion with the CoLoMoTo Interactive Notebook (Tutorial)

[29] Lewis, J., Breeze, C. E., Charlesworth, J., Maclaren, O. J., and Cooper, J. (2016). Where next for the reproducibility agenda in computational biology? *BMC Systems Biology* 10, 52

[30] Müssel, C., Hopfensitz, M., and Kestler, H. (2010). BoolNet–an R package for generation, reconstruction and analysis of Boolean networks. *Bioinformatics* 26, 1378–1380. doi:10.1093/bioinformatics/btq124

[31] Naldi, A. (2018). bioLQM: a java library for the manipulation and conversion of Logical Qualitative Models of biological networks. *bioRxiv* Not peer-reviewed

[32] Naldi, A., Hernandez, C., Abou-Jaoudé, W., Monteiro, P. T., Chaouiya, C., and Thieffry, D. (2018). Logical modelling and analysis of cellular regulatory networks with GINsim 3.0. *bioRxiv* doi:10.1101/289298. Not peer-reviewed

[33] Naldi, A., Monteiro, P. T., Müssel, C., Consortium for Logical Models and Tools, Kestler, H. A., Thieffry, D., et al. (2015). Cooperative development of logical modelling standards and tools with CoLoMoTo. *Bioinformatics* 31, 1154–9. doi:10.1093/bioinformatics/btv013

[34] Naldi, A., Remy, E., Thieffry, D., and Chaouiya, C. (2011). Dynamically consistent reduction of logical regulatory graphs. *Theor. Comput. Sci.* 412, 2207–2218. doi:10.1016/j.tcs.2010.10.021

[35] Ostrowski, M., Paulevé, L., Schaub, T., Siegel, A., and Guziolowski, C. (2016). Boolean network identification from perturbation time series data combining dynamics abstraction and logic programming. *Biosystems* 149, 139 – 153. doi:10.1016/j.biosystems.2016.07.009

[36] Paulevé, L. (2017). Pint: A Static Analyzer for Transient Dynamics of Qualitative Networks with IPython Interface. In *CMSB 2017 - 15th conference on Computational Methods for Systems Biology* (Springer), vol. 10545 of *Lecture Notes in Computer Science*, 370 – 316. doi:10.1007/978-3-319-67471-1_20

[37] Paulevé, L. (2017). Reduction of Qualitative Models of Biological Networks for Transient Dynamics Analysis. *IEEE/ACM Trans. Comput. Biol. Bioinform.* doi:10.1109/TCBB.2017.2749225. In press

[38] Peng, R. D. (2009). Reproducible research and biostatistics. *Biostatistics* 10, 405–408. doi:10.1093/biostatistics/kxp014

[39] Ragan-Kelley, M., Perez, F., Granger, B., Kluyver, T., Ivanov, P., Frederic, J., et al. (2014). The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication. In *AGU Fall Meeting Abstracts*. vol. 1, 07

[40] Richter, S. H., Garner, J. P., Auer, C., Kunert, J., and Würbel, H. (2010). Systematic variation improves reproducibility of animal experiments. *Nat. Methods* 7, 167–168. doi:10.1038/nmeth0310-167

[41] Santori, G. (2016). Journals should drive data reproducibility. *Nature* 535, 355–355. doi:10.1038/535355b

[42] Sauro, H. M., Medley, K., Choi, K., Konig, M., Smith, L., Hellerstein, J., et al. (2017). Tellurium notebooks - an environment for dynamical model development, reproducibility, and reuse. *bioRxiv* doi:10.1101/239004. Not peer-reviewed

[43] Smith, M. A. and Houghton, P. (2013). A proposal regarding reporting of in vitro testing results. *Clin. Cancer Res.* 19, 2828–2833. doi:10.1158/1078-0432.CCR-13-0043

[44] Stodden, V., Guo, P., and Ma, Z. (2013). Toward reproducible computational research: an empirical analysis of data and code policy adoption by journals. *PLOS ONE* 8, e67111. doi:10.1371/journal.pone.0067111

[45] Stodden, V., Leisch, F., and Peng, R. D. (2014). *Implementing reproducible research* (CRC Press)

[46] Stoll, G., Caron, B., Viara, E., Dugourd, A., Zinovyev, A., Naldi, A., et al. (2017). MaBoSS 2.0: an environment for stochastic Boolean modeling. *Bioinformatics* 33, 2226–2228. doi:10.1093/bioinformatics/btx123

[47] Stoll, G., Viara, E., Barillot, E., and Calzone, L. (2012). Continuous time boolean modeling for biological signaling: application of gillespie algorithm. *BMC Systems Biology* 6, 116. doi:10.1186/1752-0509-6-116

[48] Terfve, C., Cokelaer, T., Henriques, D., MacNamara, A., Goncalves, E., Morris, M. K., et al. (2012). CellNOptR: a flexible toolkit to train protein signaling networks to data using multiple logic formalisms. *BMC Syst. Biol.* 6, 133. doi:10.1186/1752-0509-6-133

[49] Thomas, R. (1973). Boolean formalization of genetic control circuits. *J. Theor. Biol.* 42, 563 – 585. doi:10.1016/0022-5193(73)90247-6

[50] Todd, R. G. and Helikar, T. (2012). Ergodic sets as cell phenotype of budding yeast cell cycle. *PLOS ONE* 7, 1–10. doi:10.1371/journal.pone.0045780

[51] Waltemath, D., Adams, R., Beard, D. A., Bergmann, F. T., Bhalla, U. S., Britten, R., et al. (2011). Minimum Information About a Simulation Experiment (MIASE). *PLOS Comput. Biol.* 7, e1001122. doi:10.1371/journal.pcbi.1001122

[52] Waltemath, D., Adams, R., Bergmann, F. T., Hucka, M., Kolpakov, F., Miller, A. K., et al. (2011). Reproducible computational biology experiments with SED-ML – the Simulation Experiment Description Markup Language. *BMC Syst. Biol.* 5, 198. doi:10.1186/1752-0509-5-198

[53] Wittig, U., Rey, M., Weidemann, A., and Müller, W. (2017). Data management and data enrichment for systems biology projects. *J. Biotechnol.* 261, 229 – 237. doi:10.1016/j.jbiotec.2017.06.007

[54] Yaffe, M. B. (2015). Reproducibility in science. *Sci. Signal.* 8, eg5. doi:10.1126/scisignal.aaa5764