University of Namur

MASTER IN COMPUTER SCIENCE

Reverse engineering of the CoLoMoTo software

BARVAUX, Aquilain

Award date:
2022

Awarding institution:
University of Namur

Link to publication

Université de Namur
Faculté d'informatique
Année académique 2021–2022

Reverse engineering of the CoLoMoTo software

Aquilain Barvaux

Promoteur : ______________________ (Signature pour approbation du dépôt - REE art. 40)
Jean-Marie Jacquet

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

1 Acknowledgements

I would like to thank my teacher and supervisor Jean-Marie Jacquet. His guidance and precious advice carried me through all the stages of my project, from the first article to this master thesis.

Thank you also to other members of the University of Namur for the knowledge they shared with me.

In addition, I am extremely thankful to my friend Benoît Notredame who shared with me precious advice and tools for the creation of this document.

I would also like to thank my mother Sophie Onderbeek, the members of my family and my friends who morally supported me during this master.

Finally, I would like to thank my colleagues from Key-performance which supported my project during those years.

2 Abstract

This document provides a retro-engineering of CoLoMoTo. The retro-engineering is motivated by a state of the art on system biology. This study showed that system biology has challenges due to the numerous models used to describe biological entities and due to the computational approach of system biology. From this statement, it was observed that the CoLoMoTo consortium proposed a solution called CoLoMoTo to address those challenges. However, this solution doesn't support yet all system biology models. It was then decided to do a retro engineering of CoLoMoTo to understand : how we can add more system biology tools in CoLoMoTo?

Ce document fournit une rétro-ingénierie de CoLoMoTo. La rétro-ingénierie est motivée par un état de l'art sur la biologie des systèmes. Cette étude a montré que la biologie des systèmes présente des défis en raison des nombreux modèles utilisés pour décrire les entités biologiques et à cause de l'approche informatique de la biologie des systèmes. À partir de cette démonstration, il a été observé que le consortium CoLoMoTo a proposé une solution appelée CoLoMoTo pour relever ces défis. Cependant, cette solution ne prend pas encore en charge tous les modèles de la biologie du système. Il a alors été décidé de faire une rétro-ingénierie de CoLoMoTo pour comprendre : comment est-il possible d'ajouter plus d'outils de biologie systémique dans CoLoMoTo ?

3 Table of content

4 Introduction

System biology is a branch of biology studying biological entities, their components and how these components interact.

To study those systems biologists rely on models which help them to get a global vision of the biological systems inner behavior. These models are simplified representation of biological bodies. They can be assimilated to engineering blueprints helping engineers to design their works.

There are many different models used to study gene regulatory system. This differentiation is explained by the numerous ways to study a model, one might want to have a basic understanding of a system while others want to cover all the possible way a system will evolve.

In this document, we are going to study the representation of these models, their outcomes and how they are completing each other.

With the rise of computer science, those models are now used in a set of software to help system biologists in their work. Indeed, the evolution of computer science is interestingly linked to the evolution of other sciences and is increasingly used in all scientific works.

Computer science helps scientists in various aspects:

- It helps them to share their work across the world faster than ever via emails, calls and webinars. These tools improve cross-country collaborations and have proven themselves even more useful during the coronavirus period.
- It fastens the experimentation by increasing the results gathering and by running simulations on their own. These simulations run in parallel and 24/7 providing tones of information to scientists.
- The work of all scientists can be gathered in databases shared by the scientific community. Therefore, building a stronger knowledge of the studied entities.

Nevertheless, computational science also brings some drawbacks:

- Biologists do not always have the knowledge to integrate data into computational models or do not want to spend time mastering computer science as they would lose time to run their experiments.

- Those who have built their own system does not consider some aspects required by their colleagues. Meaning they elaborate their own tools leading to a multiplication of software.
- Some experiment does not give the same result following the tool or methodology used. It induces some confusion when other scientists want to reproduce already made experiments and might not get the same results.

To suppress these drawbacks different solutions have been proposed. For example, MIRIAM, "*Minimum Information Requested in the Annotation of Biochemical Models*," as its name suggests force scientists to correctly annotate their models to guarantee their reproducibility and their validation by the community [2]. A more advance solution is the one proposed by "CoLoMoTo," "Consortium for Logical Models and Tools." Their aim is to ease the use of computer software by using simplified interfaces. They also provide a set of tools made by the community with the required format translator to ease cross software usage.

Colomoto proposes a collection of software tools for logical modeling: BoolNet, Logical Qualitative Modelleing toolkit, The Cell Collective, CellNetAnalyzer, CellNOPt, EPiLog, GINsim, Genetic Network Analyzer, JSMBL, LogicModelClassifier, MaBoSS, Pint, PyBoolNet, SQUAD and BoolSim, TemporalLogicTimeSeries and others [3].

All these pieces of software are combined to help system biologists to get a good understanding of a system. Each software has its own features, helping biologists in their study by providing dedicated simulations and methods. Indeed, following the type of study made, one might want to use a specific method such as asynchronous, synchronous, sequential or priority class in their simulation which can be traced, continuous, stochastic or state transition graphs. Their main purpose being to identify stable states and stable motifs [4].

5 State of the art

5.1 Global context of system biology

5.1.1 Historical Approach

System biology, is as its name suggests, the study of biology as systems. A system is a network of interdependent components, interconnected with each other creating a unified whole. Systems have an emergent behavior which is a unique property shown by the whole system and not found in its individual components. The study of systems is also found in other sciences such as medicine, psychology, sociology, and others. The advances in system biology are spreading in other branches such as biochemistry, ecology, genetics, and other domains.

Two concepts are believed to be the root of the system biology birth at the end of the 19th century, both of which started to be studied in the 17th century.

5.1.2 Reductionism

The first, reductionism, is the analysis of complex situation by reduction to manageable pieces which can be studied. The results can then be reassembled to give a global understanding of this system.

This concept is the result of the work of René Descartes (1596–1650). It was mainly used in physics and mathematics. Descartes works helped Newton (1642–1727) to mathematically describe the planetary movement and characterizing gravity increasing the belief that reductionism would provide answers to all scientific questions.

This believed spread to biology leading to the assumption that the higher level of a system can be understood from its lower components. [5]

5.1.3 Mechanistic Biology

The second, mechanistic biology, also developed in the 17th century, views the biological entities as mechanical systems. This concept arises with the construction of simple clockwork where every component can be disassembly and reassembly and has a deterministic behavior.

This idea was applied to the studied organism mainly with Jacques Loeb (1859–1924). He concludes that all biological behaviors were identical and predetermined between all members of a species meaning organisms were in fact complex machines. He also assumed there must be invariant physio-chemical mechanism ruling the organism behavior like the mechanisms in a clock. [5]

5.1.4 Limitation of the reductionist and mechanistic approach.

In the beginning of the 20th century, the reductionist and mechanistic approach started to be criticized by several biologists.

5.1.4.1 Two limitations of the reductionist approach:

One, Aristotle (384-322 B. C.) had stated, "The whole is something over and above its parts and not just the sum of them all." His approach of science was followed by many until the 17th century when it was replaced by the development of experimental physics and biology.

It was resurrected by Jan Smuts (1870–1950), which created the term holism. Holism being the fact that the characteristic of a whole or a body can be known only by studying a system as a whole and not only by studying is components. Therefore, it was necessary to develop new techniques to understand and define a system as the behavior.

Two, different investigations on living systems showed that their components were influenced and orchestrated by the upper-lying system. Therefore, the bottom-up approach would lose some key aspect of the top-down interactions.

This idea led to a change in the language used to describe systems coming from description of an organism by its action (growth, reproduction, excretion, …) to a description of its belonging in a group (organization, uniqueness, holism, …).

Reductionism and holism seem opposed by nature, but they can be regrouped by the needs to study a system. Indeed, we need to understand how the organisms and their components are assembled (reductionism) but we also need to understand the impact of the system on its components and why they have been assembled in that way. Therefore, both methods provide answers to a different question giving a global understanding of a body.

5.1.4.2 Limitation of the Mechanistic Approach

Pierre Weiss (1865–1940) observed the changes of light and gravity impact on insects. His observation showed that even if the final response to a change was the same as the behavioral steps to reach it differs among individuals.

This observation undermines Loeb's view on the body as a consistent mechanistic sequence. The same behaviors were observed with plants and how they adapt to variation in light and gravity.

Those studies opposed to the foundation of the mechanistic approach. Indeed, the final response of individuals to stimulus is the same from a statistical perspective but the trajectory they used to reach this final state varies following the individual.

Later, the publication of "Biochemical Individuality. The Key for the Genetotrophic concept by Williams, R. J. [6]" showed the variation in numerous parameters between normal, healthy and fertile human individuals.

Unlike machines, like clocks, which works only with specified components and a low error tolerance, the living cell has variable components and can tolerate variations.

The studied system showed that compensation mechanism exists to avoid turning huge molecular variation in phenotypic variation. For example, an individual with a smaller stomach might eat more frequently to compensate for their difference. This kind of compensation is typical of a system where the whole has an impact on its components.

A group of individuals can be described by a statistical approach, but each is a possible variant and will present a different characteristic and response to its environment.

5.1.5 Organization of biological system in hierarchy

In 1929, Woodger published a text pointing out the importance of the organization as a property of a living system. He showed that higher organism starts their life as a single cell and during their development, the cell multiply and the newly created cells have typical diversity, typical distribution of a typical temporal order.

As for the human body, which grows from a single cell to trillions of them. There are multiple types of human cells like skin cells which are in the outer layer of a human body and have a life span of two or three weeks. These skin cells can again be split in more subcategories.

The cells are dependent on their distribution and their behavior is restricted by it. If a cell is isolated, its behavior changes if it isn't any more under their organizational constraint. The organization is therefore an external property having a high control on the unique cell's behavior. In the same way, cellular molecules don't have the same behavior if they are separated from a cell compared to the one found in a regular cell.

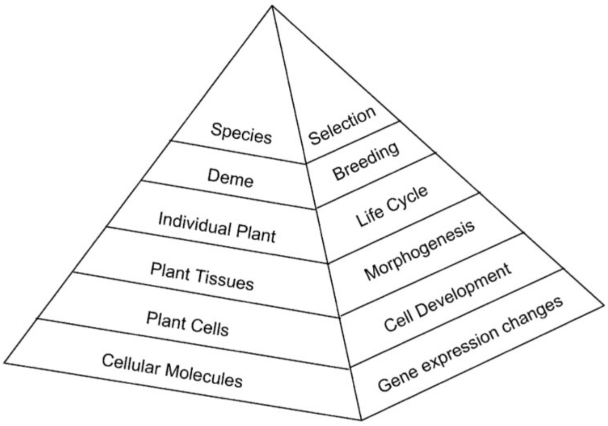


Figure 1: Biological Systems Arranged in a Hierarchy of Increasing Organizational Complexity. Source [2]

Figure 1 illustrates a simplified hierarchy; each level is property existing thanks to complex interaction in the underlying level. In reverse, each layer must follow the organizational constraints of the upper layers.

Later, Weiss tried to define the critical characteristic of biological systems using the theory of hierarchical structure from his work emerged two findings. First, the systems have greater variations at lower levels than at lower levels. Second, the output of individual behavior is more ordered than the one from an individual outside the system.

5.1.6 Communication and Control Within the System

The biological systems are networks of components working together in a system. This system is part of an organizational hierarchy and interact with the other layers. There is also a relation between each component of a system.

Indeed, there are countless observations where changing the level of a component has a direct impact on the others. They react and have their own levels changed. If the interaction between different levels known as upward and downward causation have been studied a lot, the interactions of a system inner components haven't been put aside.

5.1.6.1 Negative Feedback

Negative feedback allows the control of the reaction by monitoring them and interfering with them to match the desired outcome. It is an important element of control found in system biology but also in other domains. It allows the system to stabilize its reaction and therefore give him more resistance to external perturbations. The negative feedback is found on each layer of the hierarchy ranging from the elements in a cell to the whole population of individuals.

5.1.6.2 Feed-forward Activities

If negative feedback purpose is to gain stability, feed-forward tend to reach new states. This mechanism purpose is to adapt to external input and increase the robustness of a system. The analysis of the mechanism led to the discovery of three principles.

Draper's control timing

This principle says that the elements controlling components have to be able to change the system at least ten times faster than the reaction time of the components he has to control.

Law of requisite variety

This law is implied by the first and states that there should be as much elements to control than components to be controlled.

Changing Systems Structure and function

Given the multiple control elements and the lack of a single control factor, the manipulation of multiple linkages is necessary to imply change in the system.

5.1.6.3 Conclusion

The study of the biological systems has evolved through time. From the beginning of the 17th century the system was approached from components perspective where the observation of parts was supposed to explain the behavior of the whole. Later, the study also used a downward approach to understand the behavior from the components by the behavior of its upper layer. Both methodologies have proven useful to understand biological systems. In the meantime, observation of entities allowed the discovery of negative feedback designed to reach stability and feed-forward activities used to improve the systems to adapt to external stimulus.

5.2 Studied System : Gene Regulatory System

As we have seen in the previous section, biology studies can regroup living entities in systems which range from the population to their cellular molecules. It's the later, that will interest us in this paper. Indeed ColoMoto is designed to study the gene regulatory networks found in the cells. This section gives more context on the studied system.

Gene regulatory networks control most aspects of the cells levels of protein and RNA. They also control the genes and their expression following their spatial position and the evolution of the body in time.

5.2.1 How Does It Work

This controls system works thanks to the macromolecule Deoxyribonucleic Acid best known as DNA. It stores all the data required to create and maintain a cell and is inherited from the parents of the studied being, it's therefore the base of heredity [7].

The DNA is composed of thousands of sequences, called genes. The genes are sections of the DNA which commands the synthesis of proteins and as for the DNA is inherited from the parent. They are the bit of data constituting the DNA's global information.

They are characterized by a series of nucleotides. Nucleotides in the DNA are usually split in 4 types given their allotted base : C for cytosine base, G for Guanine, A for adenine and T for thymine.

An RNA polymerase will read the gene and produce a copy of the section in the form of messenger RNA, ribonucleic acid. The RNAs are composed of nucleotides as for the DNA with the same allotted base, except for uracil which replaces thymine.

The messenger RNA will finally be interpreted by a ribosome which will read the nucleotides by a set of 3 and produce an amino acid following the nucleotide triplet. After the whole RNA has been read, the created chain of amino acid will create a protein. Proteins are necessary to the metabolism of the cells and have many purposes in the cell, such as communication, production of energy, allowing movement, reproduction, …

5.2.2 Gene Regulatory Network

As its name suggests, a gene regulatory network, is a set of genes but also proteins that will control the development of the cell by regulating the expression of other genes and proteins. The network includes components from inside the cell but also external components. Via this process, the GRN can control the development of an entity, being plants or animals.

The inputs received by the sub-part of the network are processed by the regulatory modules and can be mathematically represented as we will see later in this document.

5.3 Objectives of System Biology

The main goal of system biology is the representation of biological entities. In this document the studied systems are mainly gene regulatory networks. In that system, the main components are proteins, RNA, genes, signaling molecules and metabolites [8].

In the second stage, the representations are used in simulations to provide information about the different stages of the system. In such an experiment, two parts are important : the stable states and the trajectories. Stable state are states without a successor or a loop of state where at some point the state repeat themselves. To find those stable state, trajectories are studied. Those trajectories are a set of state evolving through time. The simulation usually starts from an initial state, defined for the study, which follows one or more trajectories to reach one or more stable state [9].

In more advance studies, perturbations on the system are studied to understand the origins of disease such as Cancer and HIV. It's believed by the community that most disease originate from perturbations in the cellular systems. Other studies use the models and known perturbations to study the impact of drugs on the system [10].

It is therefore important for the models to first provide a clear representation of the system which will then be used to run simulation on the evolution of a system following degradation or medication.

5.4 Model Categorization based on representation

The first goal of the system biology and life scientist are the creation of biomolecular networks predictive models [11]. Such models are detailed to understand and predict the behavior of the observed biological system.

Through this document, we will review different models commonly used in the modeling and simulation of genetic regulation systems. These models can be divided into several categories. Some models may offer variation between these categories

5.4.1 Static And Dynamic Models:

Static models, as the name suggest, are representations of a system which doesn't allow the study of the evolution of the system through time or simulation. The advantages of such a system are, usually, their faster creation and lesser knowledge requirements. They are used as the basis for deeper study of a system.

Dynamic models are representations allowing the study of the evolution of different states encountered over time. They allow the deeper study of a system and its behavior across time following a set of input. But they require more knowledge of the system upon creation [9].

5.4.2 Discrete And Continuous Models

Discrete models are designed with discrete variation between their components which can be either positive, negative or null. These representations require less knowledge of the studied system but suffer from the lack of precision or realism due to their discrete range.

Continuous models have more extensive state variations between the components. They require advanced knowledge of the system to determine the scope of the various variations but offer more complete results [12].

5.4.3 Deterministic And Stochastic Models

Deterministic models only allow transitions from one state to another. In such models, each state has a unique successor except stable states which do not have successors. Starting at an initial state, the simulation evolves through different states called a trace. If there are enough steps, all the trace will end in an attractor. It can be a stable state or a cyclical attractor of length k where each k-th successor, of the state, is itself.

Stochastic models make it possible to integrate multiple evolution of a state into several states. They rely more on statistics and in some cases allow the identification of complementary attractors. As for deterministic simulations, all trajectories end in an attractor but the attractors may vary following the trajectories [9].

5.4.4 Updating Modes

The evolution of the system in time may vary following models.

There are 4 deterministic updating mode supported:

- Synchronous/Parallel: updates are applied all at the same time
- Sequential: updates are applied in a predetermined order
- Block Sequential: updates are applied in a predetermined order on the group (block) of components updated in Parallel
- Synchronous priority: updates are applied to component blocks and only the first block with updated components will be considered

There are 3 non-deterministic updating modes supported:

- Asynchronous: updates apply all logical rules independently and all successors of a state change only one component.
- Complete: updates consider all the possibilities at once.
- Priority: updates are applied to component blocks and allow some blocks (priority classes) to be updated asynchronously.

5.4.5 Qualitative Or Quantitative Results

Qualitative models are used to understand concepts, thoughts, or experiences. This type of mode allows the gathering of in-depth insights on topics that are not well understood. They are the most widespread models in actual system biology [12, 13].

Quantitative models are expressed in numbers and graphs. They are used to test or confirm theories and assumptions. This type of model can be used to establish generalizable facts about a system. The use of quantitative models allows the study of larger systems with a higher precision [12].

5.4.6 Common Models

This section regroups different models used in system biology. The listed systems aren't exhaustive. For example, Petri net models aren't explained. Nevertheless, one can use it to understand the numerous approaches to system biology and their implication

5.4.6.1 Directed Graphs

Directed graph is probably the easiest approach to modeling. A directed graph is defined as a tuple $\langle V, E\rangle$ where V is a set of vertices and E is a set of edges. E is defined as well as a tuple with $\langle i, j\rangle$ where i is the destination and j is the origin.

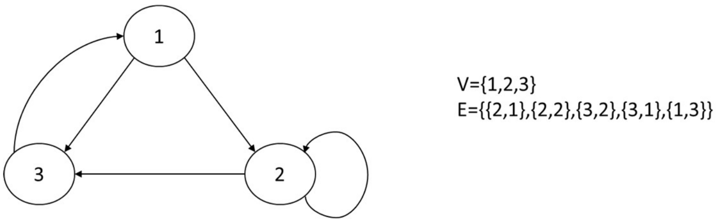


Figure 2 : Directed graph

More advance versions include activation and inhibition by adding a parameter s in the updated format $\langle i, j, s\rangle$ (e.g., figure 2 left).

In situations where multiple components cooperatively regulate another one, hypergraphs can be used. These hypergraphs use lists for the parameters j and s which are then noted J and S to represent the grouped interaction. For example, in the figure 1 on the right, the vertices 1 and 2 cooperatively activate the third one (noted $\langle 3, [1,2], [+,+]\rangle$).

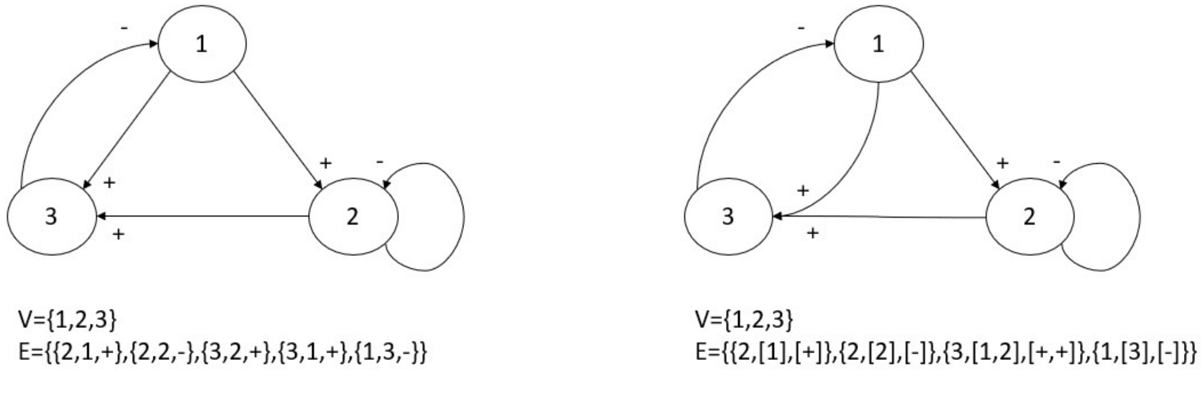

-
1
+
-
+
3
+
2
V={1,2,3}
E={{2,1,+},{2,2,-},{3,2,+},{3,1,+},{1,3,-}}
-
1
+
-
+
3
+
2
V={1,2,3}
E={{2,[1],[+]},{2,[2],[-]},{3,[1,2],[+,+]},{1,[3],[-]}}

Figure 3: Directed graphs and Directed hypergraph of 3 components interacting with each other. In the left graph, we assume 1 and 2 have a separate impact on 3 while on the left graph we assume they cooperate to act on 3.

These representations allow biologists to visualize a system or part of this system. From this visualization, they can search the path between two components. They can also isolate genes which have little interaction with others. Finally, they can compare two equivalent systems with each other rather quickly.

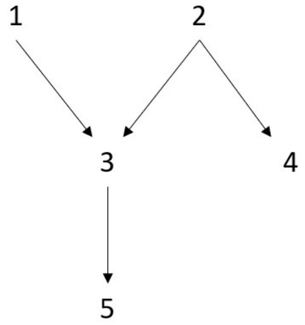
Nevertheless, these representations do not represent the interaction dynamic aspects. Therefore it isn't possible to determine stable states or traces.

5.4.6.2 Bayesian Networks

Bayesian networks are models of a genetic regulatory system by a directed acyclic graph. This graph is as well composed of vertices and edge $G = \langle V, E \rangle$.

The vertices represent components of the studied system. They are noted $i \epsilon V, 1 \leq i \leq n. X_i$ represents the expression level of i. For each expression a conditional distribution is defined $p(X_i \mid parents(X_i))$ where the parents are the direct regulator of i.

A conditional independency can be formed as well expressed by $i(Xi\,; nondescendants(Xi) |\, parents(Xi))$. With this conditional Independency. The graph encodes the Markov assumption allowing the joint probability distribution to be decomposed into $p(X) = \prod_{i=1}^{n} p(X_i | parents(X_i))$.



Conditional Distribution:
$p(X_1), p(X_2), p(X_3|X_1,X_2), p(X_3|X_2), p(X_5|X_3)$

Conditional Independency:
$i(X_1;X_2,X_4), i(X_2;X_1), i(X_3;X_4 |X_1,X_2), i(X_4;X_1,X_3,X_5 |X_2), i(X_5;X_1,X_2,X_4 |X_3)$

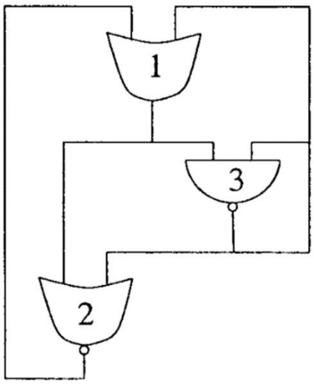
Joint probability distribution:
$p(X)=p(X_1)\, p(X_2)\, p(X_3|X_1,X_2)p(X_3|X_2)p(X_5|X_3)$

Figure 4 : Example of Bayesian Network

This technique relies on a matching score to evaluate the networks in regard to the data and search for the network with the optimal score. This model has the advantage to be based on statistics offering a good approach to deal with the stochastic aspect of gene expression and noisy measurement. Furthermore, this model can be used when there is a lack of knowledge on a particular system. It is also an intuitive representation of genetic regulatory network. Its disadvantages are the lack of dynamic aspects, the lack of currently available datasets and its optimization which is NP-hard.

5.4.6.3 Boolean models

Boolean models describe systems with Boolean variable expressing activity (1) or inactivity (0) of an item. The interaction between elements can be described with Boolean function which calculates the activation of a unit following other unit activation (Figure 4 (b)). Their representation is like those of electronic components [12, 14].



(a)

$$\hat{x}_1(t+1) = \hat{x}_2(t) \text{ or } \hat{x}_3(t)$$

$$\hat{x}_2(t+1) = \hat{x}_1(t) \text{ nor } \hat{x}_3(t)$$

$$\hat{x}_3(t+1) = \hat{x}_1(t) \text{ nand } \hat{x}_3(t)$$

(b)

Figure 5: Example of a Boolean network and the linked equation. Source : [10]

This model quickly provides information on the dynamic between the different components. The next state can be forecast, implying the Boolean model is deterministic. Another important point is that this model is synchronous meaning all transitions are applied at the same time. Therefore, sequence if state can be identified and will form trajectories.

These trajectories are determined by an initial state which will follow a set of transitions until it reaches a stable state or a state cycle which are respectively called point attractor or dynamic attractor. It provides information on the stable states and the intermediate states needed to reach stability.

Boolean models allow analysis of large regulatory networks in an efficient way by simplifying them. They present the same advantages as directed graphs and cover some parts of the dynamic aspect.

But they remain simplification of systems. Indeed, the activity level is represented as binary but in most systems the activity levels have a wider range of values.

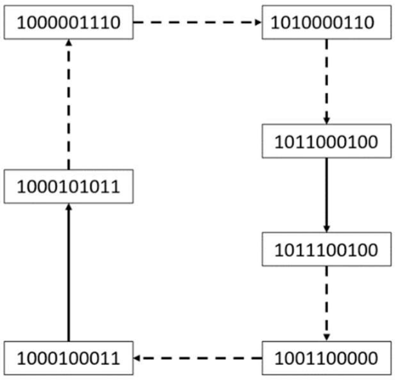


Figure 6: Visualization of a synchronous model. Source : [9].

Dashed lines represent multiple change while plain lines represent single modification.

The drawback is that kind of analysis cannot predict some behaviors as they occur on accurate activities. More complete range allows models to have more detailed results and to find new paths in a system [12, 15]. Also, systems are not updating all their components at once as an electrical system would do, meaning that the synchronous update hides some behavior. This system is deterministic, meaning the next state is always unique. In other words, they provide simplified information on the system but may hide information due to the simplification.

5.4.6.4 Asynchronous Boolean Models

An important component is the concept of time. Indeed, change in the systems can be triggered synchronously meaning all the changes are applied at once or sequential where updates occur in a predetermined order.

The synchronous version is easier to put in place but may not predict all the behaviors while asynchronous models are more precise but require more prior knowledge of the studied system and more computational resources [12].

Boolean models can be improved to provide information on asynchronous development meaning components are not updating all at once but separately.

There are also hybrid versions called “block sequential” where changes are applied in predetermined order on groups (blocks) of components updated in parallel resulting in hybrid version of the graph [16].

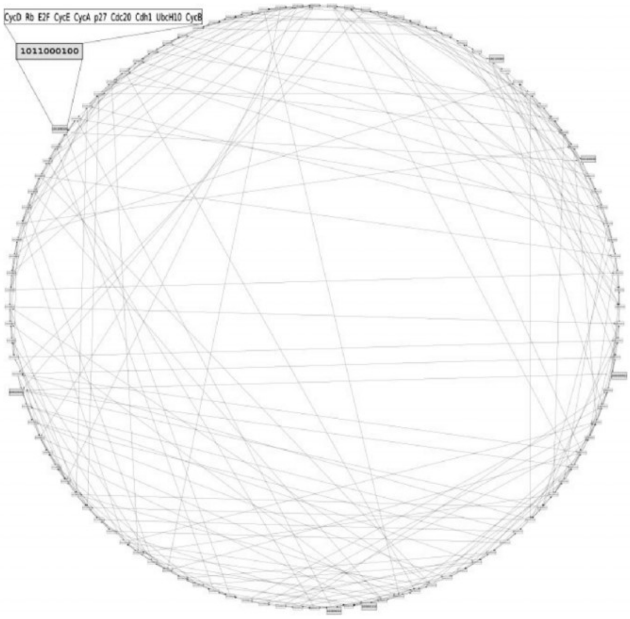


Figure 8: Visualization of an asynchronous model. Source : [9].

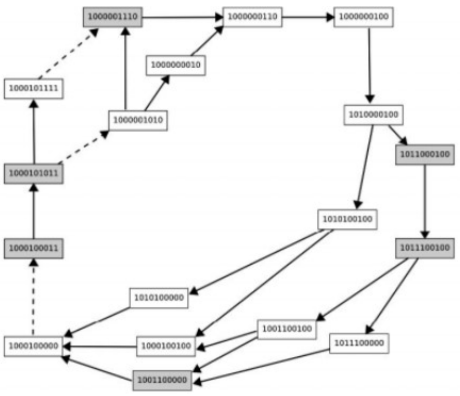


Figure 7: Visualization of a hybrid version. Source : [9].

Dashed lines represent multiple change while plain lines represent single modification.

Asynchronous and hybrid model provides more information and can show hidden paths in the system, but they are harder to read, and they grow the number of possibilities exponentially increasing the complexity and the time required to calculate and run simulations [12, 15]. Therefore they require more prior knowledge of the studied system and more computational resources while synchronous models are easier to create [12].

5.4.6.5 Generalized Logical Networks

This model, developed by Thomas, improved the Boolean model by increasing the range of the variables and use asynchronous update. This formalism uses discrete variables abstracted from the real concentration of variable. The possible values are determined by comparing the level of concentration of the variable and the number of elements it can influence [12].

Multi-valued models implement intermediate levels of activation. In these models the activity can range from 0 to n-value. These models describe in a more advance way the systems, but they require more previous knowledge of the system must maximum and intermediate values have to be defined for each interaction. Furthermore, it increases the complexity of the function defining such interaction. Indeed, the sums of logical terms have to be weighed as they don’t always have the same impact on the targeted components [15, 17]. These models are harder to run in the simulation as they require more resources, but they also provided more valuable information.

5.4.6.6 Automata Networks

Cellular Automata are widely used to study a lot of phenomena in different fields. They are flexible and simple models with a lot of capabilities. They can approximate continuous field with simpler discrete models which are easier to implement numerically, to understand and to compute [13, 18].

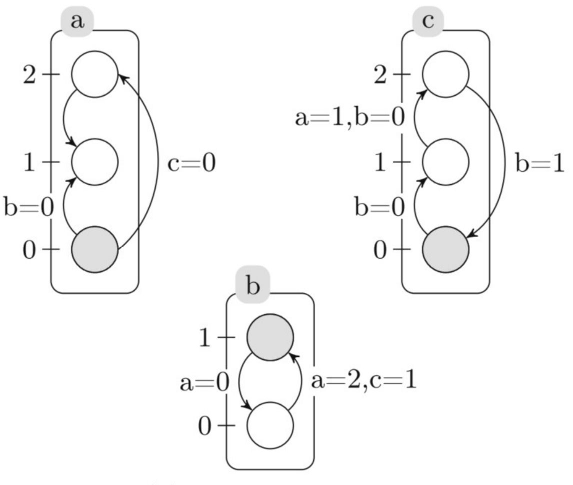
An automata network consists of a finite or infinite *d*-dimensional grid of cells. The value of each cell at *t*-time is a function of the value of neighborhood cells at *t*-1. The taken value is included in a finite set of value Σ. The cell updates can be either synchronous or asynchronous.

A cellular automaton A can be formally described as a quadruple :

$$A = (\Sigma, U, d, f)$$

Where Σ is a finite set of values, U is the cellular neighborhood, $d \in Z^{+}$ is the dimension of A and f is the transition function.

You can find below a graphical version of an automata network as found in PINT with 2 as the number of *d*-dimension :



(a) Automata network

```
a [0,1,2]
b [0,1]
c [0,1,2]

a 0 -> 1 when b=0
a 0 -> 2 when c=0
a 2 -> 1
b 1 -> 0 when a=0
b 0 -> 1 when a=2 and c=1
c 0 -> 1 when b=0
c 1 -> 2 when a=1 and b=0
c 2 -> 0 when b=1

initial_state a=0,b=1,c=0
```

(b) PINT textual format

Figure 9 : Representation of an automata network. Source : [10].

In this model, each cell is a box (a,b,c), the values or states of the cell are represented by a circle (the gray circle is the current state). The transitions are edges between the states. These transitions can be local and influenced by other cells' state. In improved automata networks, edges can go from one cell to another.

As for Boolean models, automata network can vary from simplified synchronous update to more advanced asynchronous models. They can as well include some randomness to tend to stochastic observation [18].

5.4.6.7 Nonlinear Ordinary Differential Equation

The ordinary differential equation is one of the most used systems to model dynamic systems in science and therefore in the analysis of genetic regulatory systems. This formalism models the concentration of the systems component (RNAs, protein, etc.) by time-dependent variables with values in a set of nonnegative real numbers. The regulatory interactions are represented with functional and differential relations between the concentration variables.

Gene regulation is modeled by the rate equation which expresses the rate of production of a component in regards of the concentration of the other components of the system.

$$\frac{dx_j}{dt} = f_j(x), 1 \leq i \leq n$$

Where $\boldsymbol{x} = [x_1, \dots, x_n] \geq 0$ is the vector of concentration of the system's component and $f_j\colon \mathbb{R}^n \to \mathbb{R}$ usually nonlinear function. The synthesis of j is dependent on the concentration of $\boldsymbol{x}$. f_j can be extended to include input of external components and discrete time delay can be represented to represent the time needed for events to take place.

A common example of regulation function is the Hill Curve:

$$h^+(x_j, \theta_{j,}m) = \frac{x_j^m}{x_j^m + \theta_j^m}$$

Where θ_j is the threshold of the influence of x_j on a gene and $m > 0$ a stepness parameter. The function ranges from 0 to 1 and increases as x_j increase. It expresses the activation of the gene in the function of x_j. Similarly, one can replace the function $h^+(x_j, \theta_{j,} m)$ by $h^-(x_j, \theta_{j,} m) = 1 - h^+(x_j, \theta_{j,} m)$ to express inhibition. For $m > 1$, the Hill curve has a sigmoid shape:

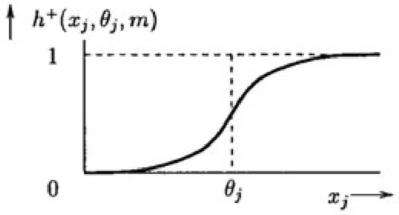


Figure 10 : Hill curve (m>1) source : [5]

Due to the nonlinearity of f_j, analytic solution is not normally possible. Another difficulty comes from the lack of measurement and therefore data on the kinetic parameters of most systems. On the other hand, this type of model provides a quantitative approach to system biology and has already been used to study well-known systems [12].

5.4.7 Piecwise-Linear Differential Equation

The piecewise-linear models are differential equation in which the variables denote concentration of genetic components. The rate of change of concentration at a particular time point is described as the balance between the rate of protein synthesis and degradation [8].

These syntheses and degradations are regulated by other components of the system by activation or inhibition of the transcription. The PL models capture the regulation effect by step function (fig7 b) that switch their value at once which approximates the sigmoidal (fig7 a) response function or Hill Curve observed in gene regulation [8, 12].

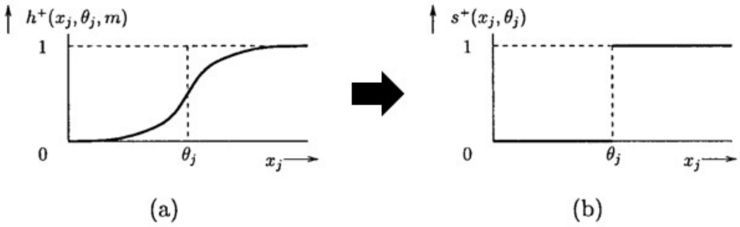


Figure 11 : regulation functions source : [5].

It allows the qualitative dynamics of the PL to be analyzed. The step usage allows the creation of an hyperrectangular partition of the state space where every region can be analyzed by a system of differential equation as long as it isn't located on a threshold. Furthermore, in these regions the concentration variable trends have a determinate sign [12] .

The use of steps to ease the representation and simulation of the system but threshold shouldn't be ignored because they can hide possible steady states or other important properties [8].

5.4.7.1 Stochastic master equation

The previously explored models provide a lot of information but suffer from two assumptions they make. First, they consider that concentration of substance varies continuously. However, systems with a low concentration of components compromise this continuity. Second, they consider a deterministic variation but there are fluctuations in the timing of cellular events.

Regarding those two issues, a discrete and stochastic model was proposed. It takes as state variable a discrete amount X of molecules and express the probability that at a certain time t the contain X of each molecule with $p(X,t)$ which is expressed as follows :

$$p(X,t+\Delta t) = p(X,t)\left(1-\sum_{j=1}^{m}\alpha_j\Delta t\right)+\sum_{j=1}^{m}\beta_j\Delta t$$

where m is the number of possible reactions, $\alpha_j\Delta t$ the probability that reaction j will occur in the interval of time given the state of the system at t and $\beta_j\Delta t$ the probability that the reaction j will set the system in state X from another state at the same time interval. Taking the limit as $\Delta t \to 0$ the formula can be rearranged to give the master equation :

$$\frac{\partial}{\partial \mathrm{t}}p(X,t) = \sum_{j=1}^{m}(\beta_j - \alpha_j p(X,t))$$

This equation provides a clear understanding of the stochastic process governing the dynamic of regulatory systems. However, its form containing n+1 variable (n variable of X and the variable t) makes it difficult to run in numerical simulations. Furthermore, this equation is difficult to solve by analytics approach.

5.4.7.2 Stochastic Simulation

A stochastic simulation is an alternative approach of the Stochastic master equation where the time evolution of the system is directly simulated. It determines what kind of reaction will happen and when given an original state X at t, evaluates the state following the reaction and goes to the next state. To guarantee that a large number of simulations will give a correct distribution of the components at a certain time, two stochastic variables (τ and ρ) are introduced and randomly chosen from a set of values with a density function derived from the same principle as the underlying master equation.

A stochastic simulation provides a more precise estimation of the gene regulation at a molecular level but it requires advance knowledge of the studied system and it requires a lot of simulations to get a proper estimation. Each of these simulations are themselves hard to run due to the high number of reactions to analyze [12, 19, 20].

5.4.7.3 Rule-base-formalism

Rule-based formalism consists of two main components, facts and rules stored in a knowledge database.

- Facts are the knowledge of a component of a regulatory system called objects. Those objects are described by properties which take their value from well-defined domains. Objects are most of the time structured in a hierarchy of classes and subclasses.
- Rules are subdivided in two parts, the condition part, and the action part. The former expresses conditions about the properties of an object while the second changes the properties of the object if the condition part is fulfilled.

Rule-based simulations are a chain of facts evaluated through rules which change the properties of the object following predefined actions. They can be forward or backward chaining mode.

In forward mode, an initial state is defined and follows the rules until it reaches one (or more) final fact.

In backward mode, the simulation process is done in reverse starting from a final state and recursively determines all its possible origin state.

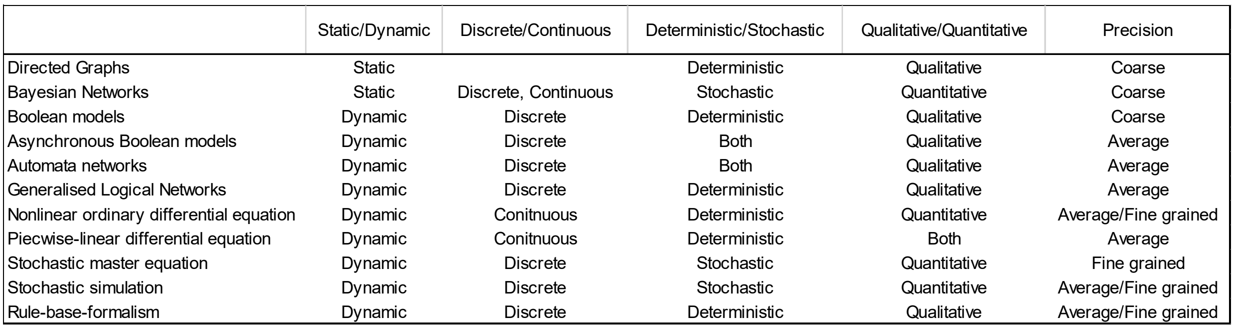
There is a control strategy to determine which rule will be applied if several conditions from different rules match at the same time. Control strategies can vary from “simple” randomness to more complex priority schemes.

The advantage of this model is its capacity to deal with numerous biological knowledge (such as temperature, physical structure of genes, …). And it implements this knowledge in an intuitive way. The disadvantages are the prerequisite knowledge needed to create such a model, the maintenance of the model (incorporate new knowledge, …) and the implementation of quantitative information.

5.4.8 Conclusion

There are many different models and simulation processes to study gene regulation systems. Simplified visualization of a system offers coarse representation of a system from which can be simulated basic reactions like Boolean models but they are required to increase the knowledge of a system.

Once enough knowledge has been acquired on a system more complex representation can be elaborate with continuous variables and stochastic simulations. Below, a table is given to summarize the characteristics of observed models.



	Static/Dynamic	Discrete/Continuous	Deterministic/Stochastic	Qualitative/Quantitative	Precision
Directed Graphs	Static		Deterministic	Qualitative	Coarse
Bayesian Networks	Static	Discrete, Continuous	Stochastic	Quantitative	Coarse
Boolean models	Dynamic	Discrete	Deterministic	Qualitative	Coarse
Asynchronous Boolean models	Dynamic	Discrete	Both	Qualitative	Average
Automata networks	Dynamic	Discrete	Both	Qualitative	Average
Generalised Logical Networks	Dynamic	Discrete	Deterministic	Qualitative	Average
Nonlinear ordinary differential equation	Dynamic	Conitnuous	Deterministic	Quantitative	Average/Fine grained
Piecwise-linear differential equation	Dynamic	Conitnuous	Deterministic	Both	Average
Stochastic master equation	Dynamic	Discrete	Stochastic	Quantitative	Fine grained
Stochastic simulation	Dynamic	Discrete	Stochastic	Quantitative	Average/Fine grained
Rule-base-formalism	Dynamic	Discrete	Deterministic	Qualitative	Average/Fine grained

Table 1 : Comparison of the observed models

5.5 Computational approach to biological models

To create those predictive models, sciences research relies more and more on computational solutions to do theoretical modeling and large-scale data analysis [21, 22].

On the theoretical modeling part, integration of available information into a common formal framework helps researchers in many aspects.

First, it helps them to identify gaps in the current knowledge as one must “fill in” the required component to create the framework.

Second, it helps to identify key regulatory factors or interactions in the system. It also helps to compare data, adjust models, and test different effect before running experience to confirm the results [23].

Furthermore, computational modeling and simulation are already used in numerous fields and has helped to dramatically reduce their development costs [1].

On the large-scale data analysis part, the studies of individual networks can be integrated in bigger models faster thanks to the computational approach. Researchers can study their individual networks and share their experimental results through databases allowing the community to build a bigger picture thanks to each scientist work. [1]

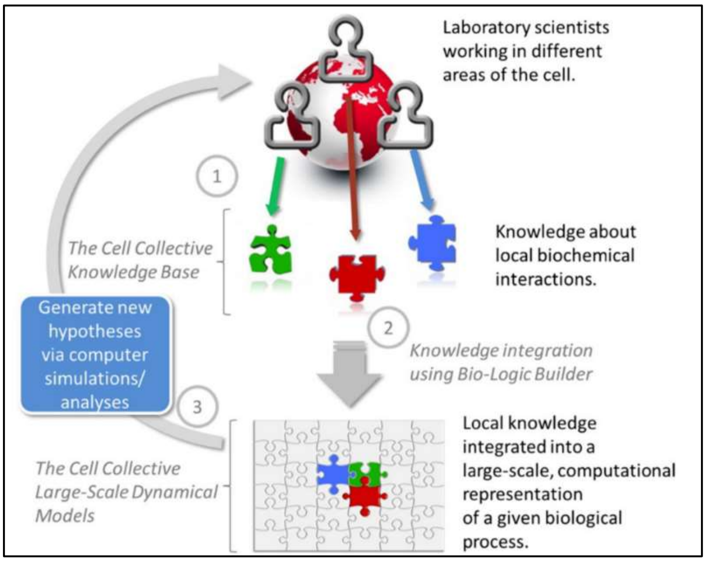

Laboratory scientists working in different areas of the cell.
1
The Cell Collective Knowledge Base
Knowledge about local biochemical interactions.
Generate new hypotheses via computer simulations/ analyses
2
Knowledge integration using Bio-Logic Builder
3
The Cell Collective Large-Scale Dynamical Models
Local knowledge integrated into a large-scale, computational representation of a given biological process.

Figure 12: Contribution of researchers to the global picture. Source : [1]

5.6 Different Models Lead to Different Software

Multiple software has been conceived to ease the development of those models. Such software has specific assets following the chosen model. Each software is developed to fulfill the requirements of the team creating them [24].

Some models are designed to allow perturbations of different natures such as fixed value, range restriction, removal of a regulator and a combination of the three. These perturbations succor the studies of cancer and other undesired behavior.

5.6.1 Multiple Tools Combination

As we saw that there are multiple models combined to gather more detailed data. So, most studies rely on a combination of different software's to get a global picture. By doing so, they face two issues. The first one is the difficulty to combine the software's. The second one is the challenge for scientists to reproduce the results of others as they must know which tools (and its version) were used and how they were assembled [25].

5.6.2 Combination Difficulties

It became necessary to combine programs to get a full picture of the studied system, but these combinations become more and more complex following the number of tools and their versions [24]. As these tools have been developed separately, they were created with different languages, increasing the need for computer knowledge to use these tools. In the same way, the difference in languages also lead to differences in the format of the files, meaning one has to convert the files before using them in another tool [1].

5.6.3 Reproducibility

Furthermore, to be able to reproduce the results, one needs to know which combination of those tools were used [22]. Being able to reproduce results is one of the sciences fundamentals. To validate studies' results the scientific community needs to be able to reproduce them.

To do so multiple groups of scientists have created tools and rules to validate quantitative models in biology to avoid their lost due to a lack of characterization. These rules will facilitate model re-use and integration in bigger models.

Another positive side effects are that they will facilitate the classification and therefore fasten the search for models, help to identify biological phenomena and, of course, increase the confidence that models are to have an accurate reference description [2].

5.7 CoLoMoTo

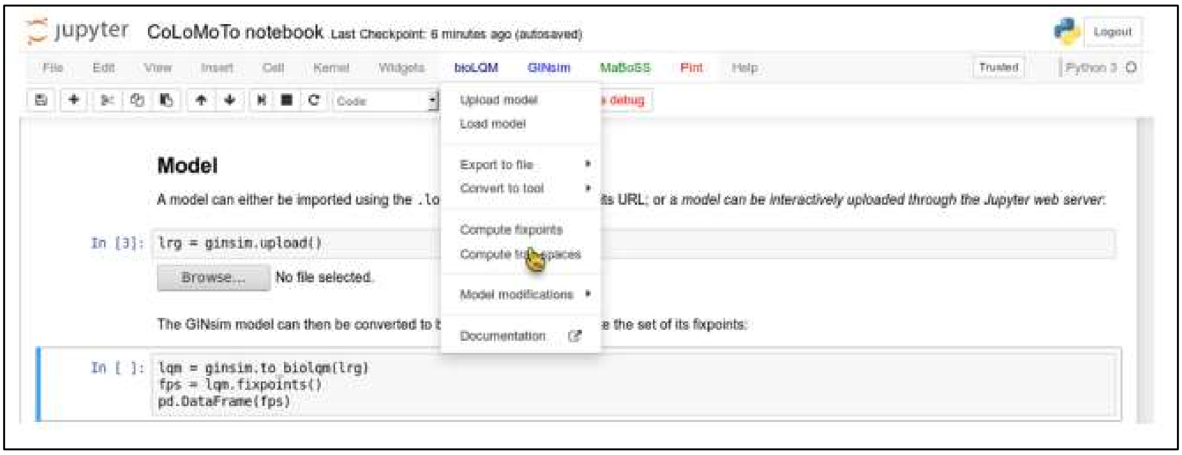
5.7.1 CoLoMoTo answers computational challenges

To tackle all these challenges, the Consortium for Logical Models and Tools was created.

The first challenge was the reproducibility. It is solved by using Docker. CoLoMoTo's tool is based on a dockerimage which embarks on a set of tools. The dockerimage is marked with a version number allowing reproduction of the results of previous experiments by picking the same build version. Once done, it is possible to rerun the experiments in similar conditions from any device. The only issue to take into account is the computational resources needed to run the experiments.

The second challenge is the ease of use for non-IT scientists. Three technologies are used to grant a simplified access to these tools.

- The Docker image already contains all the software preinstalled. It fastens the usage of the tools because users do not have to install them one by one. Moreover, Docker is compatible with GNU/Linux, MacOs and Microsoft windows. Meaning you can run each software on most computers.
- Python modules are integrated with the software to provide a unified interface to the users. These modules are designed to parameterize and execute analysis, fetch their results, and use them via other software.
- Finally, on top of the Python module, there is a Jupyter interactive notebook included in the image. It provides a web interface to create documents, called notebooks. They include codes, images, texts, tables. The code module can use the previously mentioned Python module to invoke CoLoMoTo software tools. Meaning the whole CoLoMoTo toolbox can be used via this interface as observed in Figure 7.



Model

A model can either be imported using the .lo its URL; or a *model can be interactively uploaded through the Jupyter web server.*

```
In [3]: lrg = ginsim.upload()
```

The GINsim model can then be converted to t e the set of its fixpoints:

```
In [ ]: lqm = ginsim.to_biolqm(lrg)
        fps = lqm.fixpoints()
        pd.DataFrame(fps)
```

Figure 13: Jupyter interface. Source : [14].

5.7.2 Tooling in CoLoMoTo

ColoMoto images embark most used tools in system biology. The complete list can be found on their website (http://www.colomoto.org/software/). Among many, GINsim is one of the best known as it is used to design models and as well by other software for model conversion. Below is a non-exhaustive list of tools found in ColoMoto :

5.7.2.1 GINsim

GINsim, standing for Gene Interaction Network simulations, is one of the commonest tools used in system biology. This tool is used to define, simulate, and analyze regulatory graph based on the logical formalism.

This kind of representation enrich Boolean models by including ranges in the nodes. Instead of an active or inactive state, the activity level can have intermediate levels. The number of intermediate values ranges from 0 to maximal value defined by the user. The nodes are linked by arcs from a source node to a target one. A threshold is set to indicate when the source regulates the target. The value of the threshold can vary between one and the maximal value of the source node. The arcs are represented with a “+” for activation, with a “-‘ for inhibition, and can be represented with a “?” when the interaction depends of cofactors. The target level of the destination node can be defined for each of the incoming interactions.

Moreover, this formalism is asynchronous providing deeper knowledge of the system. However, multi-valued variables and asynchronous updating come with their drawbacks as the exponentially increasing the possibilities and results, the time required to calculate and run simulations increase in the same manner.

Thus, GINsim is efficient to study the small-scale systems but ask for more resources to study large-scale systems.

GINsim is available for free for non-profit academic purposes. The latest version is 3.0.0 release in March 2018 and can be found on the official website in the download section (http://ginsim.org/downloads). GINsim is platform independent but it requires to have a Java Virtual Machine running on the host as it is developed and JAVA.

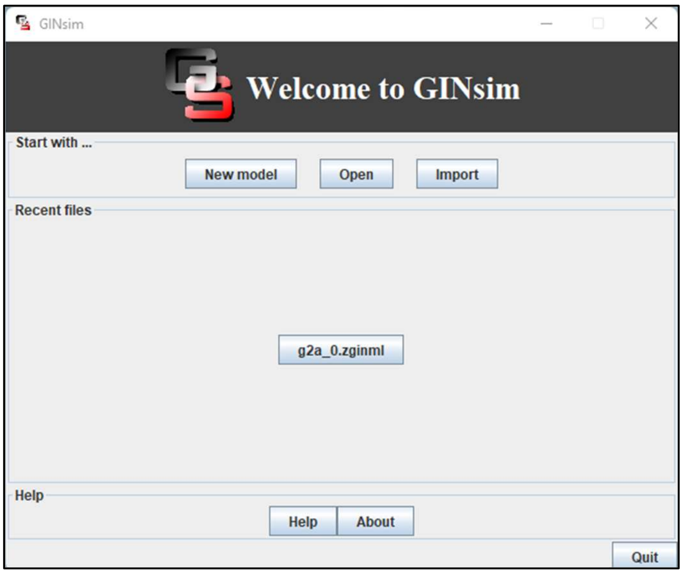
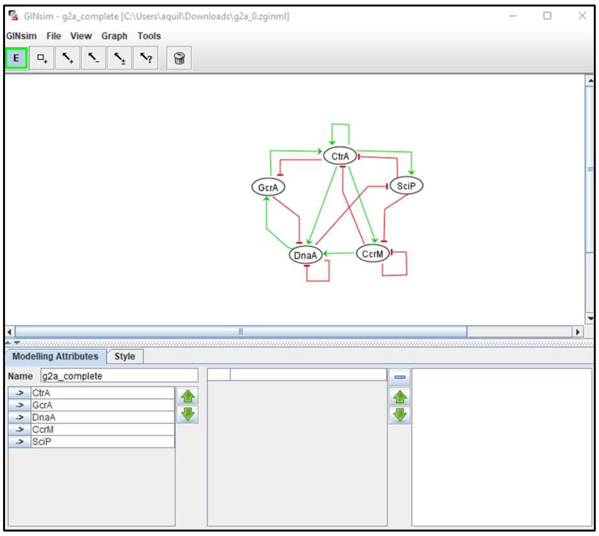


Figure 14 : GINsim Graphical User Interface

The tool allows the user to create a new model, open already created models, and import models from another format (BoolNet, Boolsim, TrutTable and SBML-qual). The storage of the models is based on an XML file format ".ginml" that can be zipped in the format ".zginml" with other files which includes simulation parameter definition.



Name g2a_complete

-> CtrA
-> GcrA
-> DnaA
-> CcrM
-> SciP

Figure 15 : Asymmetric Cell Division in Caulobacter Crescentus in GINsim

On the website, there is a model repository including already created models, a summary, and the linked paper. For example, the model “Asymmetric Cell Division in Caulobacter Crescentus” is available with a summary and the related paper “Exploring Asymmetric Cell Division in Caulobacter Crescentus Using Logic Modeling.” This model can be downloaded and opened into GINsim with the resulting Figure 15 [10, 13, 26].

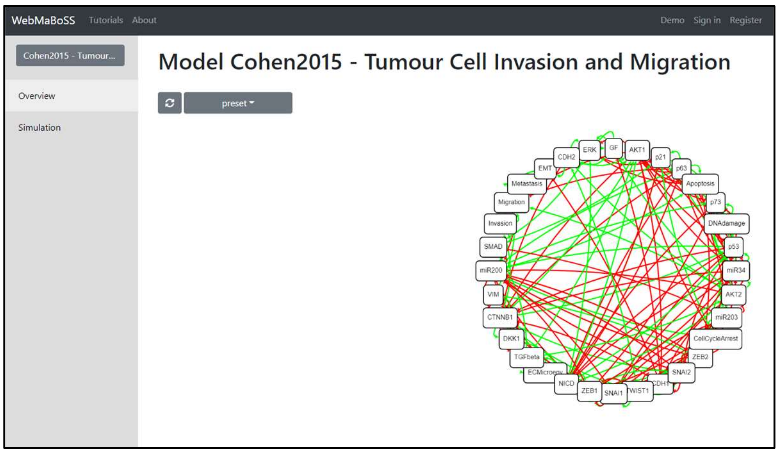
5.7.2.2 Pint

Pint is used to study large-scale networks and is based on automata network which encompasses Boolean and discrete models. The study of large-scale networks means that it doesn’t need to use reduced models like other tools. Therefore, it will provide more information on the studied system and avoid the loss of information linked to model reduction. It’s based on statistical analysis of the models and provide information on the correctness and the completeness of its results. It has three main features, the reachability analysis of a large system based on statistics, the prediction of mutation and the identification of the bifurcation in transitions. It’s worth mentioning that the reachability analysis process may be inconclusive if the correctness isn’t verified [13].

5.7.2.3 MaBoSS

MaBoSS, standing for Markovian Boolean Stochastic Simulator is designed to do define, simulate, and analyze regulatory networks with Boolean modeling. It is Markovian meaning that the probability to reach the next state is only depending on the state reached in the previous operation. It also handles physical time, instead of a sequence of events like for already introduced Boolean models. The initial state and the transaction can be stochastic meaning than a given network can have different trajectories. The results are then computed to obtain mean trajectories with probabilities per state over time.

MaBoSS latest version is 2.0. It can run on Unix-based OS and on windows if the core C++ script is installed. The tool can be downloaded from Curie's MaBoss website (https://maboss.curie.fr/) but there is also a web version available on https://webmaboss.vincent-noel.fr/model/overview/. The non-authentified users have access to two demo models while registered users can create their own models or import existing one from libraries.



Model Cohen2015 - Tumour Cell Invasion and Migration

In summary, MaBoss provides Boolean models with a description of the temporal behavior of all the biological entities of the system. It has proven its efficiency in the study of cancer-related models. Since version 2.0, it provides a set of tools to create models, visualizes solutions, and do simulate mutation and drug treatment. It was initially written in C++ but as new plugins written in Perl and Python [10, 27].

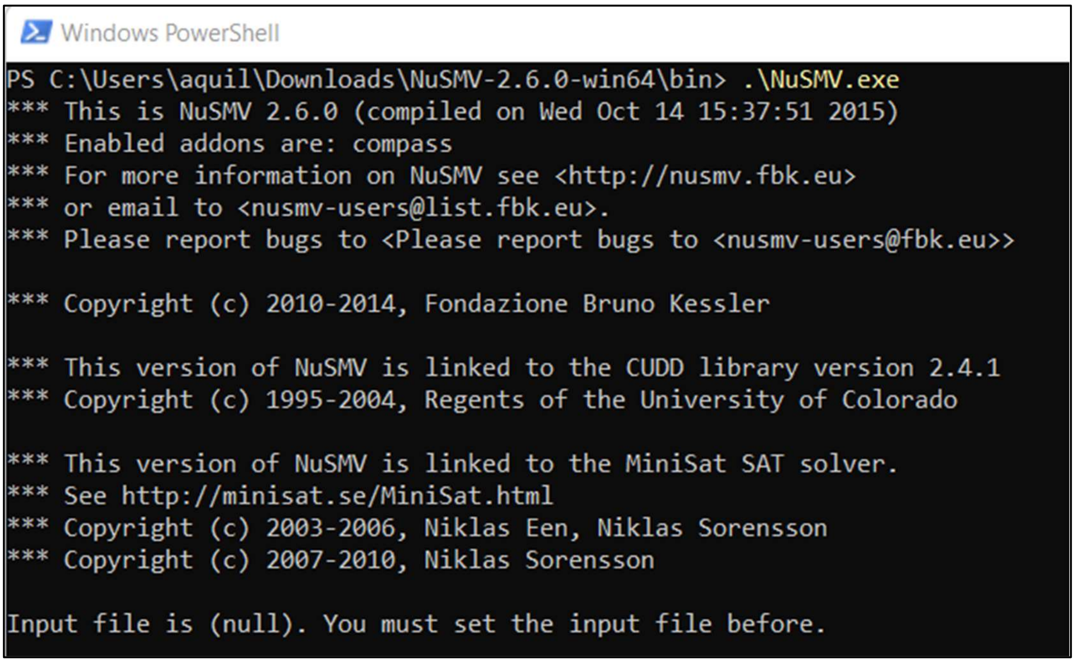
5.7.2.4 NuSMV

NuSMV is the results of the reengineering of SMV or Symbolic Model Verifirer. The reeingeeniring was done in a joint project between Carnegie Mellon University (CMU) in the United States of America and Istituto per la Ricerca Scientifica (IRST) in Italy. The purpose was to improve and extend SMV. The improvement was made in 3 dimensions. The functionalities were extended with a textual interaction shell, extended model portioning technique and checking and a graphical interface. The architecture was designed to be more modular and open to extension. The implementation was improved to increase the robustness and the maintainability.

At the beginning, NuSMV only implemented BBD-based symbolic model checking. Binary decision diagrams are used to symbolically represent the transition relations and the states of a studied system by Boolean functions. This representation allows the implementation of model checking algorithms such as fair CTL modeling checking and LTL model checking .

In its second version, NuSMV embarks also an SAT-based model checker. Satisfiability takes Boolean logic formula as input and has as an output a combination of variables that can satisfy it. Otherwise, it can demonstrate that no such combination exists.

NuSMV latest version is 2.6.0. It is open source and can be downloaded on NuSMV website (https://nusmv.fbk.eu/NuSMV/download/getting-v2.html). There is an available version for Windows, Linux and MacOS [28-31].



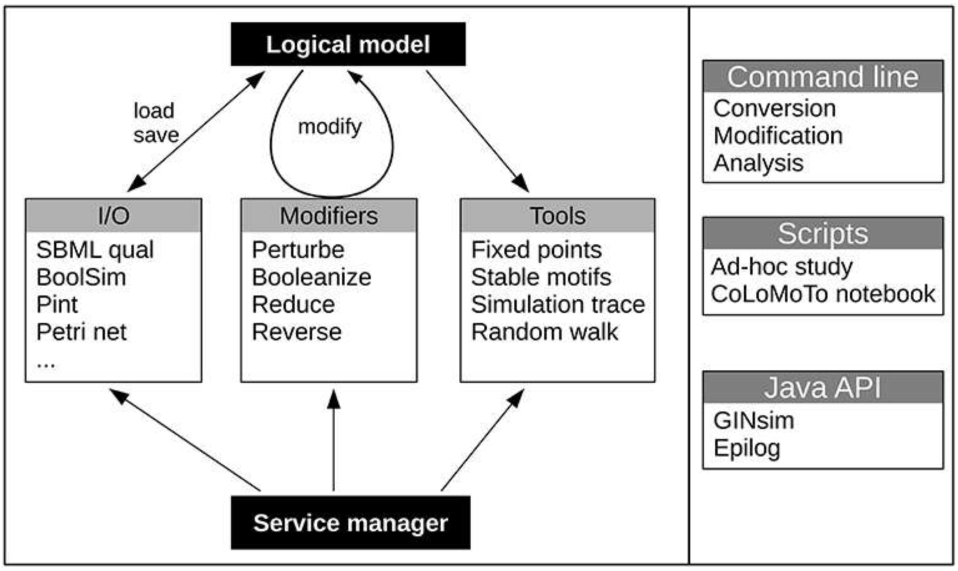
```
PS C:\Users\aquil\Downloads\NuSMV-2.6.0-win64\bin> .\NuSMV.exe
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:51 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

Input file is (null). You must set the input file before.
```

5.7.2.5 Tooling Combination

Each of those tools provides a different set of information on the studied system. Most of the time, they are used in a combined manner to first understand the standard behavior of a system and the analyze with more advance models. SBML is a common format which has been proposed to allow faster exchange of qualitative models between tools. But it has been integrated in all the tools included in the CoLoMoTo package [25, 32].

5.7.3 BioLQM

This part is focused on BioLQM which is a java toolkit used to manipulate and convert logical qualitative models of the studied networks. It implements a series of possibilities with format conversion, dynamic analysis and model modification.

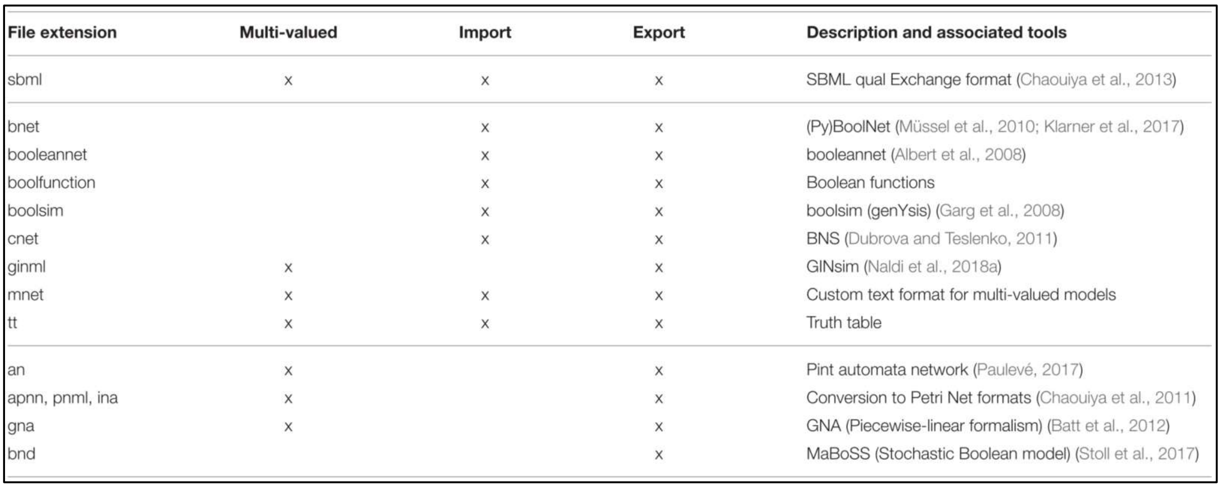


5.7.3.1 Format Conversion

As seen previously, many software tools were developed to study biological systems and most of them use their own file system leading to difficult workflows to combine different tools. SBML qual exchange format was proposed as standard but is not always implemented.

To ease model exchange with software not supporting SBML, BioLQM provides a list of the format handler connected to its internal model representation. Each format is described as a Java Class.

BioLQM can import and export the different formats which are listed in figure 16. Once the result have been formatted, it can be used with another software. This one of the important roles of BioLQM within CoLoMoTo [9].



File extension	Multi-valued	Import	Export	Description and associated tools
sbml	x	x	x	SBML qual Exchange format (Chaouiya et al., 2013)
bnet		x	x	(Py)BoolNet (Müssel et al., 2010; Klarner et al., 2017)
booleannet		x	x	booleannet (Albert et al., 2008)
boolfunction		x	x	Boolean functions
boolsim		x	x	boolsim (genYsis) (Garg et al., 2008)
cnet		x	x	BNS (Dubrova and Teslenko, 2011)
ginml	x		x	GINsim (Naldi et al., 2018a)
mnet	x	x	x	Custom text format for multi-valued models
tt	x	x	x	Truth table
an	x		x	Pint automata network (Paulevé, 2017)
apnn, pnml, ina	x		x	Conversion to Petri Net formats (Chaouiya et al., 2011)
gna	x		x	GNA (Piecewise-linear formalism) (Batt et al., 2012)
bnd			x	MaBoSS (Stochastic Boolean model) (Stoll et al., 2017)

Figure 16: list of tools supported by BioLQM [9]

Standard format :

SBML stands for system biology markup language. In this paper, it is extended by “qual” or qualitative model package. SBML qual is designed for the representation of qualitative model of biological system in computers. The purpose of this format is to have a standard representation of such systems. The usage of standards allows an easier usage of the same file and therefore the same model across multiple tools. SBML qual is based on XML and supports the representation of Boolean logical models, multi-valued systems and Petri nets.

Qualitative format

In this part, we find formats which are qualitative and therefore can be imported into BioLQM.

- Bnet is a format used by BoolNet and its Python version pyBoolNet .This software is used for Boolean models representation and simulations. It supports synchronous, asynchronous, probabilistic and temporal Boolean networks. The late version of BoolNet is 2.1.5 and 3 for pyBoolNet [33].
- Booleannet used by the software Booleannnet. This software is running in Python and specialized in discrete and hybrid models.
- Boolsim used to find attractors; it isrunning in C++
- Cnet used by BNS
- Ginml used by GINsim provides graphical visualization; it is running in JAVA
- Mnet used to describe multi-valued models
- Tt used to create true tables

Quantitative format

- An is used by Pint which is running in ASP and is used for large-scale analysis and to do interference simulations
- Apn, pnm and ina are used to create Petri net formats
- Gna used by GNA which is used to run simulation of gene regulatory networks based on piecewise-linear differential equation models
- Bnd used by MaBoSS which is used to run Stochastic simulations

As observed in Figure 16, BioLQM can export results in multiple tools but is not able to import from all of them.

5.7.3.2 Model creation and simulation

The state of a model is a vector giving the activity levels of all its components. As activity levels have a finite range, the state space is also finite but the number of state grows exponentially with the number of components. A component is called to update in a given state if the evolution of the associated logical rule is different from its current activity level.

The evolution of a model is given by the transition between the states of the model and is controlled by the updating calls but also the updating modes which define how the updating calls are made. Following the updating mode, reachability properties and cyclical attractors can vary a lot while stable states aren't impacted.

There are multiple updating modes with most software specialized in one of those modes. BioLQM has the goal to provide most of those updating mode in a single toolkit.

BioLQM supports deterministic simulation. In those simulations each state has a unique successor except stable state. Starting with an initial state, the simulation evolves through different states called a trace.

There are 4 deterministic updating mode supported in BioLQM. First, synchronous where updates are applied all at the same time. Second, sequential updates which are applied in a predetermined order. The third is a mix of the first and the second and is called bock sequential. The updates are applied in predetermined order for group of components and these groups are updated in Parallel. And finally synchronous priority where updates are applied to component blocks and only the first block with updated components will be considered.

BioLQM supports as well Non-Deterministic models. In those models each state can have multiple successors. The initial state is defined and can lead to multiple trajectories. This type of simulation is often represented by STG.

BioLQM supports the definition of the models but doesn't provide an engine for those simulations or a data structure for STG. But these features are integrated in GINsim which can be used with bioLQM as we will see in the next section about Jupyter.

There are 3 non-deterministic updating mode supported in the tool. The first are asynchronous updates which applies all logical rules independently and all successors of a state change only one component. The second is complete updates. They consider all the possibilities at once. The last one is priority updates which are applied to component blocks and allow some blocks to be updated asynchronously.

Finally, BioLQM support partially stochastic updaters. They allow the computation of a single successor selected randomly across multiple possibilities and can therefore change between each call. They can be derived from a deterministic updater by giving equal probabilities to all transitions defined in the original updater or a custom updater can be made by defining individual properties. BioLQM provides a randomization tool to run stochastic simulation but is limited to the construction of single trajectories and doesn't provide a complete analysis. The integration of BioLQM with other tools is advised to allow the complete analysis of such simulation results.

The analysis of large networks leads to combinatorial explosion, even more in a non-deterministic case. BioLQM implements two methods based on constraints solving to identify attractors without explicitly enumerating all states.

The first method enables the identification of stable states by extracting and combining stability conditions from the logical rules. BioLQM includes this method and also propose an alternative implementation based on "clingo ASP solvers" which is slower for small models but can scale faster sometimes. Similar methods are also provided by GNA and Pint.

The second method is based on stable patterns which helps to identify complex attractors (which, has their name suggest, are more complex to identify) and doesn't depend on the updating mode. A pattern is partially defined state where part of the component have fixed activity levels while others are undefined. These patterns represent all states with matching activity level for a set of components. It is considered as stable if the images of all the states included are belonging to the pattern. This method is included in BioLQM using ASP solvers and introduce an alternative implementation based on decision diagrams. But as already mentioned complex attractors exact identification requires further analysis using other tools.

5.7.3.3 Model Modifications

BioLQM also proposes model modification tools which create alternative models from existing ones. The modified models can be described by a keyword and other parameters within the tool. There is a model modifier API allowing multiple modifications chaining before conversion or analysis.

There are 3 kinds of model modification proposed by BioLQM :

1. Perturbations and Mutations : enables the change of some logical rules of a model. BioLQM provides : 3 types of "atomic perturbation" : fixed value, range restriction, removal of a regulator. These atomic perturbations can be combined to create multiple perturbations.
2. Model Reduction : enable the reduction of a model while keeping similar dynamic properties.
3. Boolean mapping of multi-valued Models

With its 3 functions BioLQM offer a lot of possibilities to study systems. It is also designed to facilitate the addition of new components and it provides an API. Hardware requirements depends on the size and the structure of a model and also how it is analyzed.

5.7.4 Jupyter Notebook

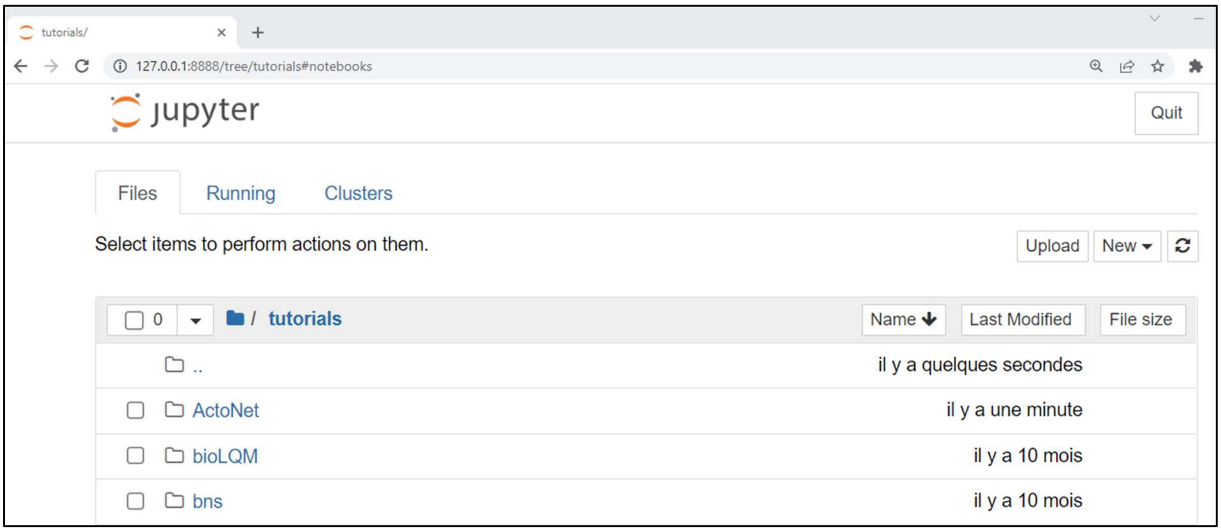
Another key software used by CoLoMoTo is the Python web interface called Jupyter. It allows the creations of notebooks which include text, graphs, tables as well as code and equations. These notebooks are used, within CoLoMoTo to describe analysis workflows.

These workflows include the code ran to study the desired regulatory network with the parameters used in the tools and explanation of the methodologies and results.

The notebooks are stored in a single file which can be shared to allow other members of the community to visualize, re-execute and modify them.

Jupyter is also used in CoLoMoTo as the interface to command the tool included into the container. Indeed, the user can create a sequence of cells with each cell contains code to run the tools. The cells can be executed and will return the tool's results in the desired format (text, graphs, images, etc.).

The container of CoLoMoTo includes an installed version of Jupyter notebooks and can be accessed on localhost on port 8888. During the installation, tutorials for the tools included are also created. However if the user wants to familiarize with interface, a demo version is available on colomoto website (http://tmpnb.colomoto.org/).



Select items to perform actions on them.

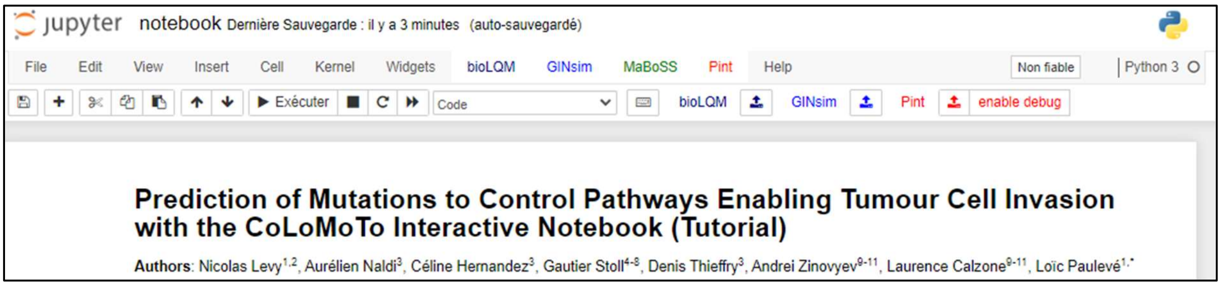
/ tutorials

..	il y a quelques secondes
ActoNet	il y a une minute
bioLQM	il y a 10 mois
bns	il y a 10 mois

Three kinds of modules have been added within the tool to ease the creation of workflows by users. First, menus have been enhanced to provide predefined code to use the main features of the tools. Second, a module has been created to upload model files on the server. This module is mainly useful if the user doesn't have a direct access to the server file system. The last kind of modules are Python modules which ease the generation of code for the tools by providing JavaScript widget which generate the code interactively.

An example of the usage of Jupyter Notebook within CoLoMoTo can be found in the publication "*Prediction of Mutations to Control Pathways Enabling Tumor Cell Invasion With the CoLoMoTo Interactive Notebook (Tutorial)*" [24]. This example illustrates the installation and the usage of the tool. The installation is done with the dockerimage of May 2018 which open the webpage where the tools are used. The author first use GINsim to import an existing model from the GINsim model repository. Then they used bioLQM to convert the GINSim file into a format usable by BioLQM and to find the list of stable state of the studied model. One of the stable states found with BioLQM is then pinpointed within GINSim. Afterwards, the author uses MaBoss to do a stochastic simulation with the same model find the probability in the aim of finding alternative attractors. To do so he re-use BioLQM to convert the file from BioLQM format to MaBoSS format and enhance the model by defining an initial state, simulation parameters and the desired output nodes. Once done, he uses Pint to make predictions on mutation combination and NuSMV to check the model. In the end MaBoSS is re-used to assess the result.

This complete process was made within Jupyter and is publicly available into the supplementary materiel of the article. It can be reproduced by uploading the notebook into a Jupyter installed with the same version of CoLoMoTo's docker image.



Prediction of Mutations to Control Pathways Enabling Tumour Cell Invasion with the CoLoMoTo Interactive Notebook (Tutorial)

Authors: Nicolas Levy[1,2], Aurélien Naldi[3], Céline Hernandez[3], Gautier Stoll[4-8], Denis Thieffry[3], Andrei Zinovyev[9-11], Laurence Calzone[9-11], Loïc Paulevé[1,*]

5.8 External tools

Other tools like GNA are part of the CoLoMoTo consortium but not implemented with the docker image.

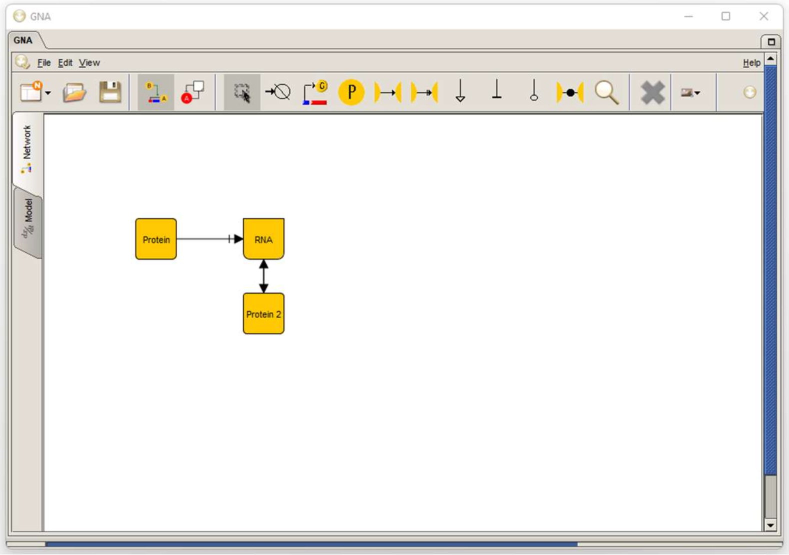
GNA, standing for Genetic Network Analyzer, is used to define, stimulants, and analyze regulatory networks based on piecewise-linear (PL) differential equation models.

PL models are based on the differential equation. They describe the rate of change of genetic materials in time based on other components of the system. They simplify the differential equation by using steps function instead of sigmoidal function as shown in Figure 5.

This approach simplify the creation of models because it doesn't rely on quantitative data unlike differential equations. Instead, they can be generated from qualitative observation. However, the results of this approach are approximation of the behavior. Even more when the results are coming from the area around the threshold they should be handled with extra care as the sigmoid isn't represented.

With this approach, the resulting graph is a state transition graph where each node is qualitative state representing the regions in the space of state and the edges between each state are the switches created by the transition from the differential equations to the picewise-linear models.

As for Ginsim, GNA is free for non-profit academic purpose. The latest version is 8.7 and can be found on the team website of the INRIA "National Institute for Research in Digital Science and Technology" (https://team.inria.fr/microcosme/genetic-network-analyzer-gna/). It's available on Windows, MacOS and Linux.



In short, GNA allows scientists to define regulatory networks build models of the network, find the steady states of the studied system, perform qualitative simulations and verify them with a model checking tools. To counter that lack of visibility linked to PL's threshold, GNA allows ones to use differential inclusion which allows for steady states in which one or more variables have a threshold value. It can export format in GNA and SBML [8].

5.9 Conclusion

The number of representations previously mentioned finds its origin in the funnel effect necessary to study a system.

At first, a system biologist is going to use a simple model to roughly describe a system and begin his studies.

Once he has acquired enough knowledge of the system thanks to its experience or by sharing with its pairs, he can use more complex models which will provide more fine-grained results.

From these models follows numerous software specialized in the representation of one or more specific models. The multiplication of these tools complexifies the share of knowledge on systems due to the specificity of each tool.

CoLoMoTo addresses this challenge by offering a solution including a set of software. This solution is based on 3 main components : Docker, BioLQM and Jupyter Notebooks.

The usage of docker deserves two goals. The first goal is to simplify the installation of the tools. The second goal is to keep track of all the versions used within the tools to increase the reproducibility.

The model transfers between the installed tools are managed by the usage of BioLQM. However, BioLQM doesn't allow yet the exchange between all tools. This format exchange is key because the implication of all these models was important to fully understand the system and its dependency.

The last component, Jupyter Notebooks, is used to create notebooks which are used as instruction to follow a methodology allowing one to reproduce all the step of a study to understand the obtain results.

One question arises from that information : it is important to improve the coverage of as much software as possible to get a better understanding of the systems. Therefore, how can we add more system biology tools in CoLoMoTo?

6 Development

6.1 Introduction

This chapter will study the possibility to add more tools in CoLoMoTo. To understand how to add tools in CoLoMoTo it is required to deep dive into the tool installation and into its components. The tool relies on Python to be installed. The analysis starts with the prerequisite to install CoLoMoTo. Then, we will deep dive in the Python code to understand how it works. The analysis ends with docker files which contains all the instructions required to properly create the container.

6.2 Prerequisite

The installation of CoLoMoTo is done with pip, a packet manager developed in Python. Therefore the user needs to have a running version of Python on his computer. Moreover, it must be a Python version 3 installation.

Python version 3.10 is the latest version released. It is available on Python website (https://www.Python.org/downloads/). Python is available for most platforms including Windows, Linux and MacOS. Users can run the command “Python -- version” to check if Python is correctly installed in version 3. If the user as version 2 installed in parallel of version 3, he can force the usage of version 3 by running “Python 3” instead of just “Python.”

Pip is a package installer for Python. It is usually installed by default with Python. If it isn’t the case, pip can be downloaded and installed manually.

Once Pip is installed, the user can run the command, “pip install -U colomoto-docker.” The flag “-U” stands for upgrades meaning it will automatically look for the latest version and install it. If you have an older version of ’colomoto-docker” it will be replaced by the newest version. Pip will look for the package in the Python package index or “pype” and install it. The package can be seen on pype’s website (https://pypi.org/project/colomoto-docker/).

This package contains the script “colomot_docker.py.” This Python script is designed to launch the docker container of CoLoMoTo.

6.3 Colomoto_docker.py

“Colomoto_docker.py,” command the launch of the docker container. It also supports a couple of inputs like for example selecting the desired version of the container.

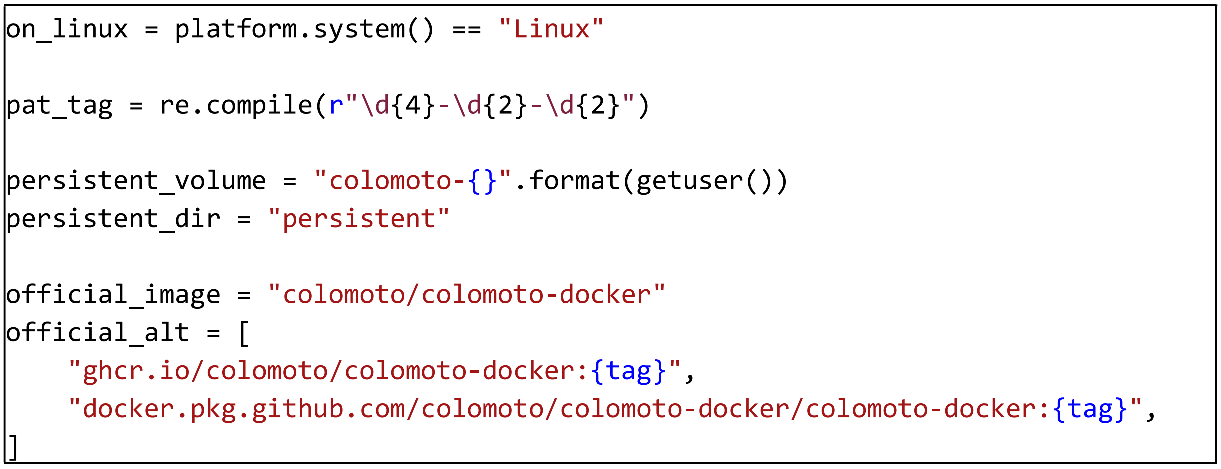
The files (Appendices 9.1) start with a couple of imports. The first one is “from __future__ import print_function.” This “future” statement needs to be handled at the beginning of the file because it is handled in a particular way by the compiler. The future statement means that the package syntax or semantic have to be handled with a newer version of Python. This may involve that some keyword doesn’t have the same usage in the package.

```
from __future__ import print_function

from argparse import ArgumentParser, REMAINDER
import os
from contextlib import closing
from getpass import getuser
import platform
import re
import socket
import subprocess
import sys
import webbrowser
```

Figure 17: Python imports

The other imports are “ArgumentParser,” “REMAINDER” from “argparse,” “OS,” “closing” from “contextlib,” “getuser” from “getpass,” “platform,” “re,” “socket,” “subprocess,” “sys” and “webbrowser.”

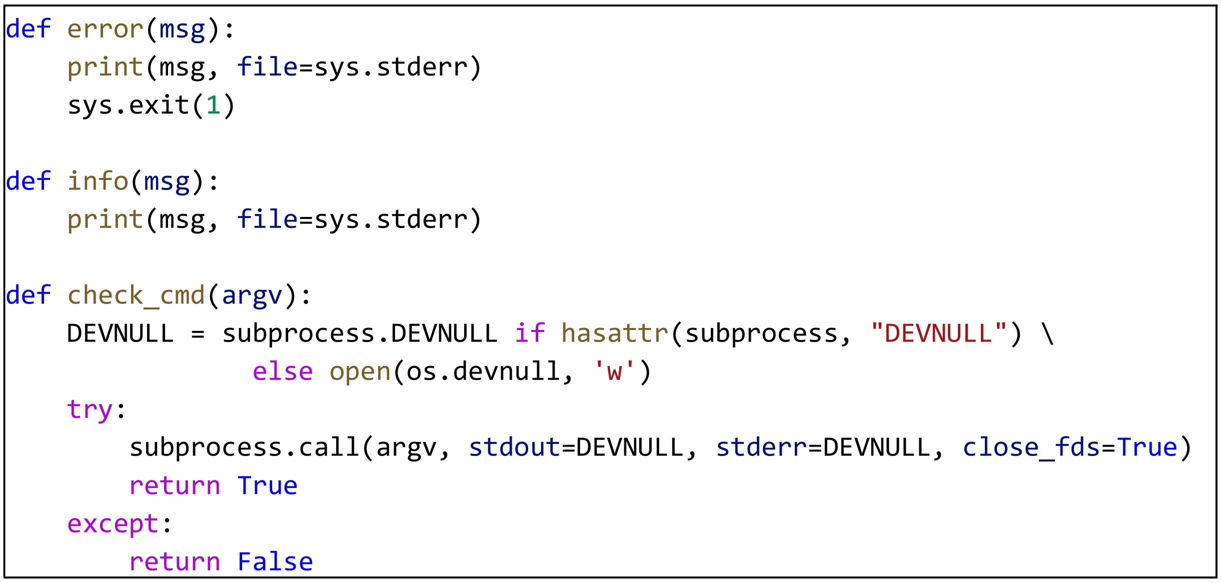
A couple of variables used later in the script are created for example a true or false is created to check if it is running on a Linux machine or not and stored in a variable on_linux.



```
on_linux = platform.system() == "Linux"

pat_tag = re.compile(r"\d{4}-\d{2}-\d{2}")

persistent_volume = "colomoto-{}".format(getuser())
persistent_dir = "persistent"

official_image = "colomoto/colomoto-docker"
official_alt = [
    "ghcr.io/colomoto/colomoto-docker:{tag}",
    "docker.pkg.github.com/colomoto/colomoto-docker/colomoto-docker:{tag}",
]
```

Figure 18: variables

After the variable definition, some functions are created to pass error messages and information and also to check if the user is able to execute docker. Following the OS of the user, the software will return an error if docker toolbox isn't available or if the user doesn't have the proper rights to use docker on Linux.



```
def error(msg):
    print(msg, file=sys.stderr)
    sys.exit(1)

def info(msg):
    print(msg, file=sys.stderr)

def check_cmd(argv):
    DEVNULL = subprocess.DEVNULL if hasattr(subprocess, "DEVNULL") \
                else open(os.devnull, 'w')
    try:
        subprocess.call(argv, stdout=DEVNULL, stderr=DEVNULL, close_fds=True)
        return True
    except:
        return False
```

Figure 19 : error handling

```
def check_sudo():
    return check_cmd(["sudo", "true"])

def docker_call():
    direct_docker = ["docker"]
    sudo_docker = ["sudo", "docker"]
    if on_linux:
        import grp
        try:
            docker_grp = grp.getgrnam("docker")
            if docker_grp.gr_gid in os.getgroups():
                return direct_docker
        except KeyError:
            raise
        if not check_sudo():
            error("""Error: 'sudo' is not installed and you are not in the 
'docker' group.
Either install sudo, or add your user to the docker group by doing
   su -c "usermod -aG docker $USER" """)
        return sudo_docker
    return direct_docker

def check_docker():
    if not check_cmd(["docker", "version"]):
        if not on_linux:
            error("""Error: Docker not found.
If you are using Docker Toolbox, make sure you are running 'colomoto-docker'
within the 'Docker quickstart Terminal'.""")
        else:
            error("Error: Docker not found.")
    docker_argv = docker_call()
    if subprocess.call(docker_argv + ["version"], stdout=2):
        error("Error: cannot connect to Docker. Make sure it is running.")
    return docker_argv
```

Figure 20 : handling docker error

The last part of the code is the main section. This section, we have the list of all the arguments handled by the script.

- Bind is used to bind a path to the working directory of the container. The default value is none.
- No-selinux is a flag to disable SElinux (Security-Enhanced Linux) in the container. The default value is "False."
- Workdir is used to change the working directory inside the docker image and is by default "/notebook"
- Shell is a flag to start a shell instead of the notebook. The default value is false. It allows the user to run shell command before running Jupyter inside the container.
- Version allows the user to select a specific version he wants to use. The default value is "same." Same means the script will look for the most recently fetched image. This flag can have the value "latest" to download the latest tag on GitHub (where the images are stored). It can also take a specific tag value to get a specific image. In the GitHub context, the tags are the dates of the image release in the "Y-mm-dd" format.
- Port is used to change the local port. The default value is 0. Even if this argument takes 0 as default value, the value used by default in the script will be 8888. If this port isn't available and there aren't specified ports, the Python will look for the next available port and use it instead.
- Image defines the image to use in the container. By default, the value is "official image."
- No browser if the user doesn't want to automatically start the browser at the start of the container. The default value is false.
- Unsafe-SSL to avoid connections to look for a certificate. The default value is false.
- No update to disable the automatic update of the image The default value is false.
- Clean up to remove the old images. The default value is false.

Other arguments exist to give more control to the user on the docker run command.

- Env to set environment variables
- Name to give a specific name to the container
- Volume to select a specific volume where the data of the persistent file are going to be stored

```
def main():
    parser = ArgumentParser()
    parser.add_argument("--bind", default=None, type=str,
        help="Bind specified path to the docker working directory")
    parser.add_argument("--no-selinux", default=False, action="store_true",
        help="Disable SElinux for this container")
    parser.add_argument("-w", "--workdir", default="/notebook", type=str,
        help="Workdir within the docker image")
    parser.add_argument("--shell", default=False, action="store_true",
        help="Start interactive shell instead of notebook service")
    parser.add_argument("-V", "--version", type=str, default="same",
        help="""Version of docker image ('latest' to fetch the latest tag;
        'same' for most recently fetched image)""")
    parser.add_argument("--port", default=0, type=int,
        help="Local port")
    parser.add_argument("--image", default=official_image,
        help="Docker image")
    parser.add_argument("--no-browser", default=False, action="store_true",
        help="Do not start the browser")
    parser.add_argument("--unsafe-ssl", default=False, action="store_true",
        help="Do not check for SSL certificates")
    parser.add_argument("--no-update", default=False, action="store_true",
        help="Do not check for image update")
    parser.add_argument("--cleanup", default=False, action="store_true",
        help="Cleanup old images")

    group = parser.add_argument_group("docker run options")
    group.add_argument("-e", "--env", action="append",
        help="Set environment variables")
    group.add_argument("--name", help="Name of the container")
    group.add_argument("-v", "--volume", action="append",
        help="Bind mount a volume")
    docker_run_opts = ["env", "name", "volume"]

    parser.add_argument("command", nargs=REMAINDER, help="Command to run
instead of colomoto-nb")
    args = parser.parse_args()
```

Figure 21: Argument handler

Then the main section handles the arguments and prevents the issues like troubles to connect to the official GitHub or handling if the user has inserted a wrong version.

If the script is run without argument, it will first look for existing images or download the docker images if it has been specified by the user or if no images have been already downloaded.

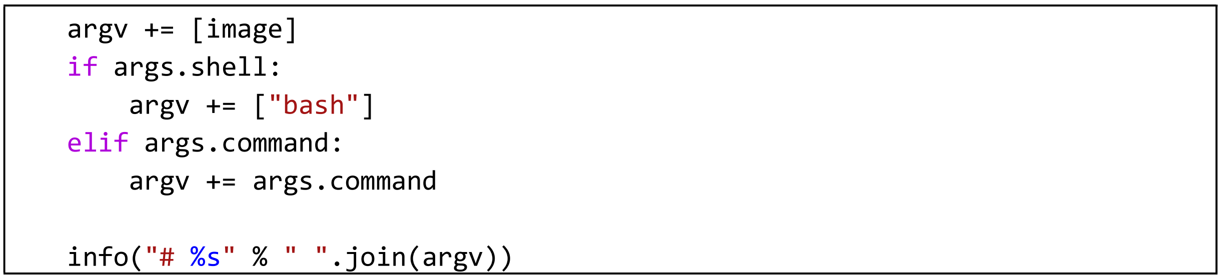
```
# querying for latest tag of colomoto/colomoto-docker...

# using colomoto/colomoto-docker:2022-07-01

2022-07-01: Pulling from colomoto/colomoto-docker

b10d9a43e982: Pull complete

774334e15edb: Pull complete
```

Figure 22: example of image pull

Once done, the end of the main section will run the following command, ‘docker run -it --rm --volume colomoto-[username]:/notebook/persistent -w /notebook -p 8888:8888 colomoto/colomoto-docker:2022-07-01.’ This command will start a new container based on the colomoto images with the version used in this study which is the one from July 2022 called 2022-07-01.

```
    argv = docker_argv + ["run", "-it", "--rm"]
    if args.no_selinux:
        argv += ["--security-opt", "label:disable"]
    if args.bind:
        argv += ["--volume", "%s:%s" % (os.path.abspath(args.bind),
args.workdir)]
    else:
        persistent_mount = "%s/%s" % (args.workdir, persistent_dir)
        argv += ["--volume", "%s:%s" % (persistent_volume, persistent_mount)]
    argv += ["-w", args.workdir]
    if not args.shell and not args.command:
        container_ip = "127.0.0.1"
        docker_machine = os.getenv("DOCKER_MACHINE_NAME")
        if docker_machine:
            container_ip = subprocess.check_output(["docker-machine", "ip",
docker_machine])
            container_ip = container_ip.decode().strip().split("%")[0]
        if args.port == 0:
            # find next available
            for port in range(8888, 65535):
                with closing(socket.socket(socket.AF_INET,
socket.SOCK_STREAM)) as s:
                    dest_addr = (container_ip, port)
                    if s.connect_ex(dest_addr):
                        break
        else:
            port = args.port
        argv += ["-p", "%s:8888" % port]
```

Figure 23 : docker run command creation (part 1/2)



```
    argv += [image]
    if args.shell:
        argv += ["bash"]
    elif args.command:
        argv += args.command

    info("# %s" % " ".join(argv))
```

Figure 24 : docker run command creation (part 2/2)

```
Status: Downloaded newer image for colomoto/colomoto-docker:2022-07-01

docker.io/colomoto/colomoto-docker:2022-07-01

# docker run -it --rm --volume colomoto-aquil:/notebook/persistent -w /notebook -p 8888:8888
colomoto/colomoto-docker:2022-07-01
```

Figure 25: example of container startup

Finally, the main browser of the user will be open to go to "http://localhost" on port 8888 which will open the Jupyter notebook inside the container.

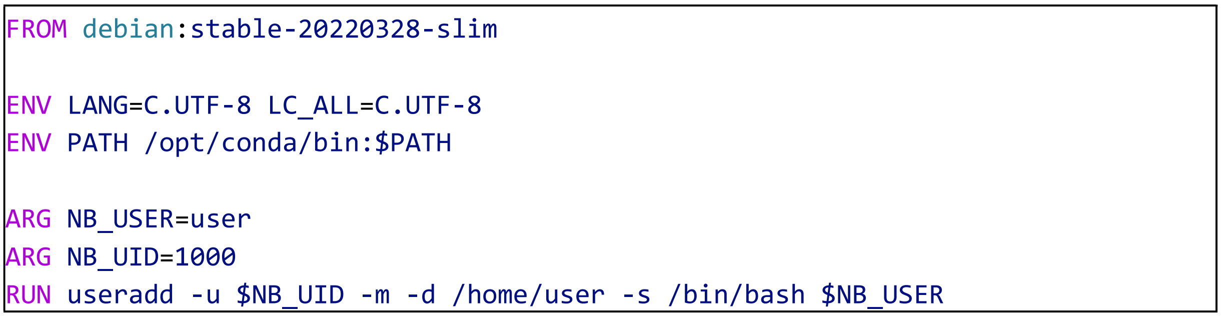
6.4 The dockerfile

The last step to understand CoLoMoTo structure is the analysis of its dockerfiles. It includes all the steps required to create the image. This system comes with a versioning of the dockerfile. To retrieve an older version of the docker images, it is possible to invoke an older version by specifying, "-V *the_older_version*."

The dockerfile begins with a "FROM" to select the image used as a basis for the container in this case the image is "debian:stable-20220328-slim." This is an official image released by Debian. The slim version has a reduced weight by excluding a handful of packages and removing docks and localization files. This image is about 40% lighter than the standard one. As for many other parts of the file, this keeps the container as light as possible.

Then, the environment language is set to C.UTF-8 and the path is set to "/opt/conda/bin."

The user: "user" is created with the uid 1000.



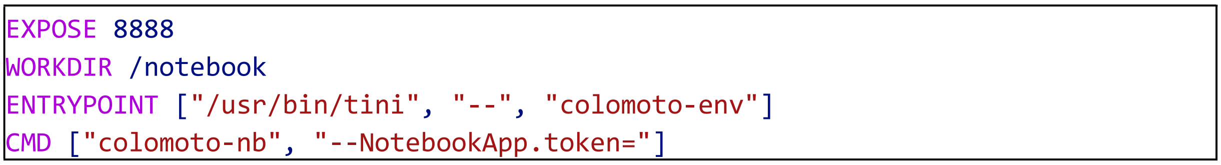
```
FROM debian:stable-20220328-slim

ENV LANG=C.UTF-8 LC_ALL=C.UTF-8
ENV PATH /opt/conda/bin:$PATH

ARG NB_USER=user
ARG NB_UID=1000
RUN useradd -u $NB_UID -m -d /home/user -s /bin/bash $NB_USER
```

Figure 26: dockerfile part 1

The port 8888 is exposed, this will be the port to use to access the Jupyter interface which is the main interface for colomoto. "/notebooks" is set as a working directory meaning the "RUN," "CMD," "ENTRYPOINT," "COPY" and "ADD" will be run in this directory.

Further the entry point set the executable to "/usr/bin/tini" and specify the '--' as the first parameter and "colomoto-env" as the second.

The CMD will specify the default parameters of the entry point previously created.



```
EXPOSE 8888
WORKDIR /notebook
ENTRYPOINT ["/usr/bin/tini", "--", "colomoto-env"]
CMD ["colomoto-nb", "--NotebookApp.token="]
```

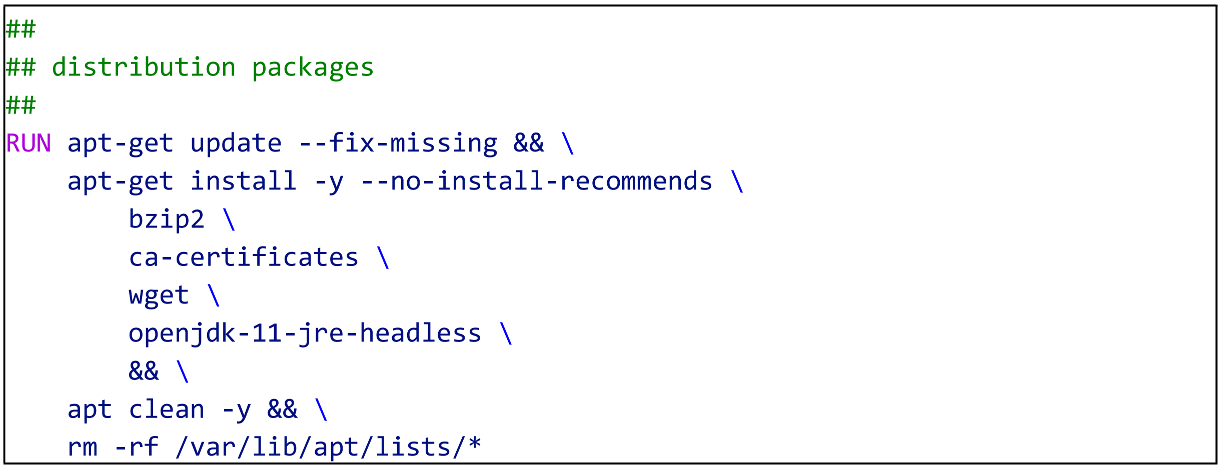
Figure 27 : dockerfile part 2

The next part of the installation includes the update of "APT." It is used to manage Debian packages. In this case it will be used to install additional packages in the slim Debian.The packages are installed silently with the flag "-y" and with the flag "--no-install-recommends" to avoid installing complementary dependencies.

The package installed is

- Bzip2: an open-source file compression software using the Burrow-Wheeler algorithm
- Ca-certificats: this program allows the generation of Ca-certificates and to hold SSL certificates. Ca-certificates are used to verify the identity of a third party and encrypt the data between a client and the third party.
- Wget: software to download files from web servers via HTTP, HTTPS and FTP
- Openjdk-11-jre-headless : a JAVA execution environment

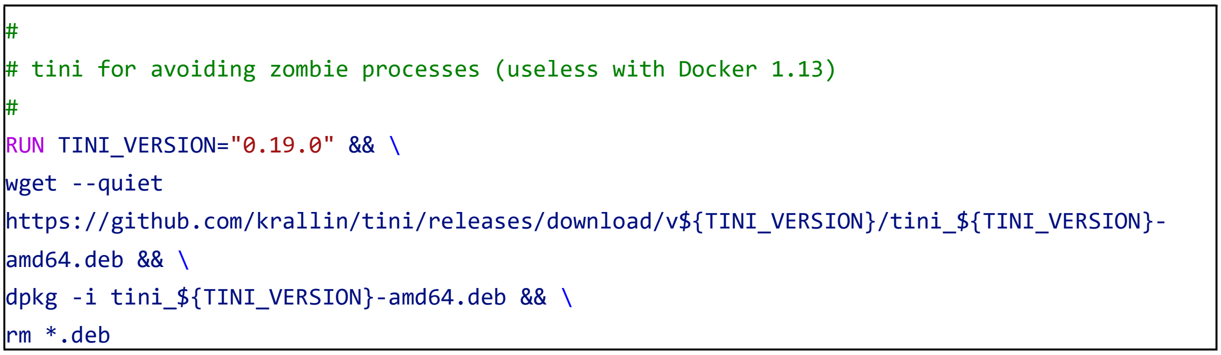
Apt is cleaned and the files are removed from /var/lib/apt/lists



```
##
## distribution packages
##
RUN apt-get update --fix-missing && \
    apt-get install -y --no-install-recommends \
        bzip2 \
        ca-certificates \
        wget \
        openjdk-11-jre-headless \
        && \
    apt clean -y && \
    rm -rf /var/lib/apt/lists/*
```

Figure 28: dockerfile part 3

Tini is installed to avoid zombie process. The version used is 0.19.0. It is downloaded quietly with the wget previously mentioned, it's installed and the installer is removed.



```
#
# tini for avoiding zombie processes (useless with Docker 1.13)
#
RUN TINI_VERSION="0.19.0" && \
wget --quiet
https://github.com/krallin/tini/releases/download/v${TINI_VERSION}/tini_${TINI_VERSION}-
amd64.deb && \
dpkg -i tini_${TINI_VERSION}-amd64.deb && \
rm *.deb
```

Figure 29: dockerfile part 4

In this part, Conda is installed in its miniconda version, as its name suggest miniconda is a lightweight version which installs less dependency than the other version named Anaconda. The minified version is used to keep the container light.

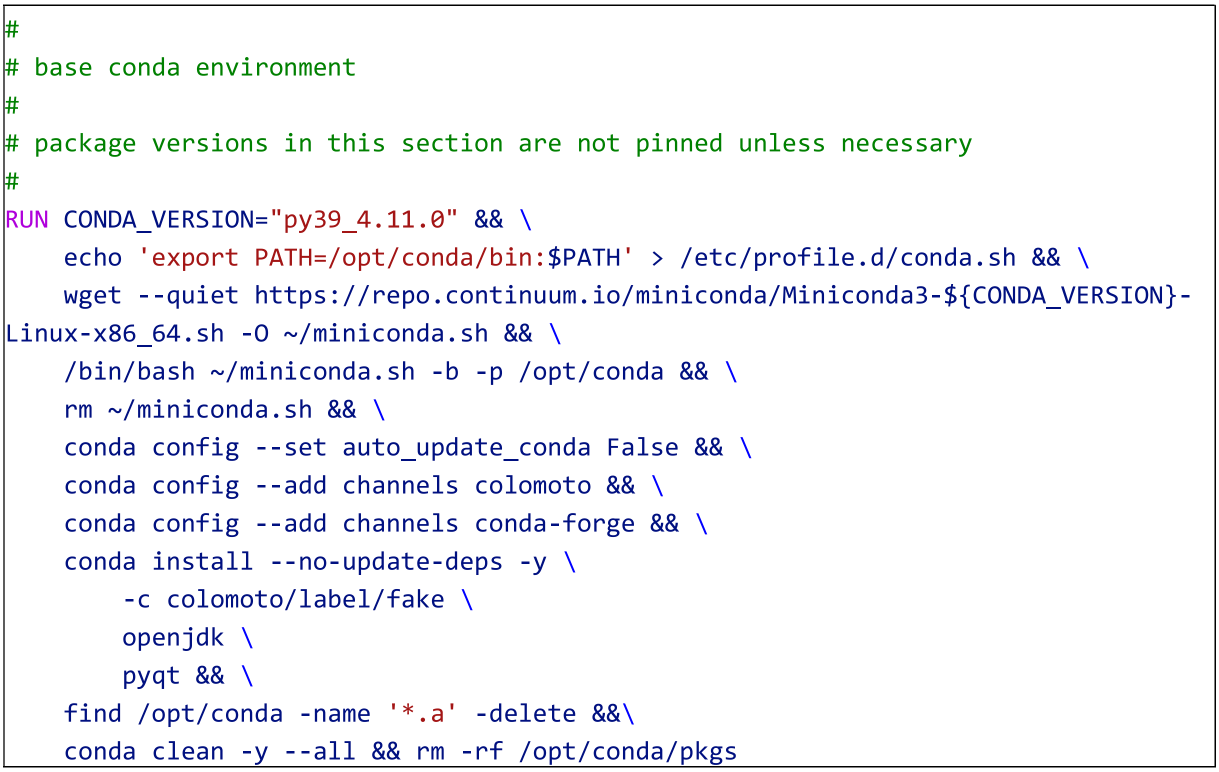
Conda has 3 purposes. First, it is used as a package manager. We will see later in the dockerfiles that it is used to install all the software needed. Second, it can be used to create a virtual environment. Virtual environment allows for example a user to use multiple Python version on the same device depending on the project they are working on. In the same way, it can be used as well to install specific package for the projects without including them in all the project. This second purpose isn't used in this case as the purpose of the container is only to be used for CoLoMoTo. Third, Conda includes a lot of packages required for data analysis and data science.

Like for Tini, the software is downloaded in a specific version and the installer is removed after the installation. Then the automatic update is disabled, and two channels are added.

The channels are used in Conda to download package not included by default in Conda. The channel colomoto included the tools we mentioned earlier in this paper plus a couple more. The channel conda forge included other packages such as R and others that will be installed further in the dockerfile.

Conda is then used to install silently ("-y") package from the channel "colomoto/label/fake" without updating the installed packages.

The package installed are openjdk and pqyt. The first is an open-source JAVA development kit and the latest is the integration of QT in Python. QT is a crossplatform program mainly designed to create graphical user interfaces. It embarks a lot of modules to allow the handling of widgets, multimedia, svg, XML but also components to handle Bluetooth, positioning, web sockets and other network interfaces. The installers are removed.



```
#
# base conda environment
#
# package versions in this section are not pinned unless necessary
#
RUN CONDA_VERSION="py39_4.11.0" && \
    echo 'export PATH=/opt/conda/bin:$PATH' > /etc/profile.d/conda.sh && \
    wget --quiet https://repo.continuum.io/miniconda/Miniconda3-${CONDA_VERSION}-
Linux-x86_64.sh -O ~/miniconda.sh && \
    /bin/bash ~/miniconda.sh -b -p /opt/conda && \
    rm ~/miniconda.sh && \
    conda config --set auto_update_conda False && \
    conda config --add channels colomoto && \
    conda config --add channels conda-forge && \
    conda install --no-update-deps -y \
        -c colomoto/label/fake \
        openjdk \
        pyqt && \
    find /opt/conda -name '*.a' -delete &&\
    conda clean -y --all && rm -rf /opt/conda/pkgs
```

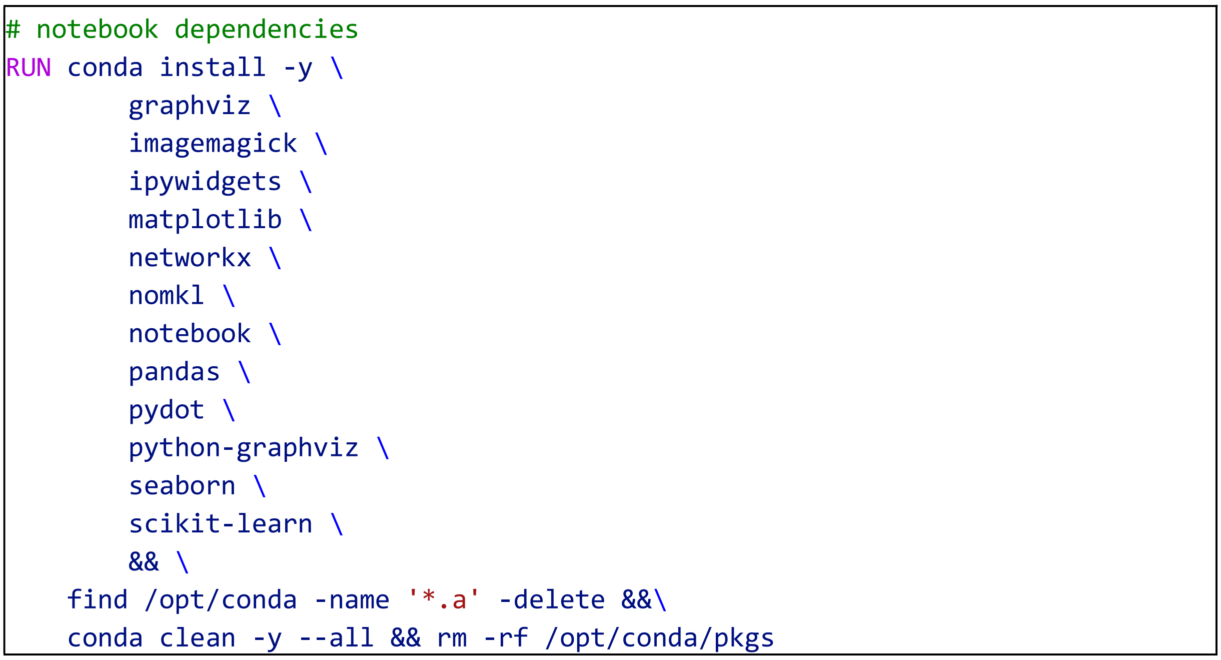
Figure 30: dockerfile part 5

In this part of the dockerfile, the different packages needed for the notebook are installed with Conda as well and the installer are erased afterwards.

The package installed are:

- Graphviz: an open-source graph visualization program using DOT language

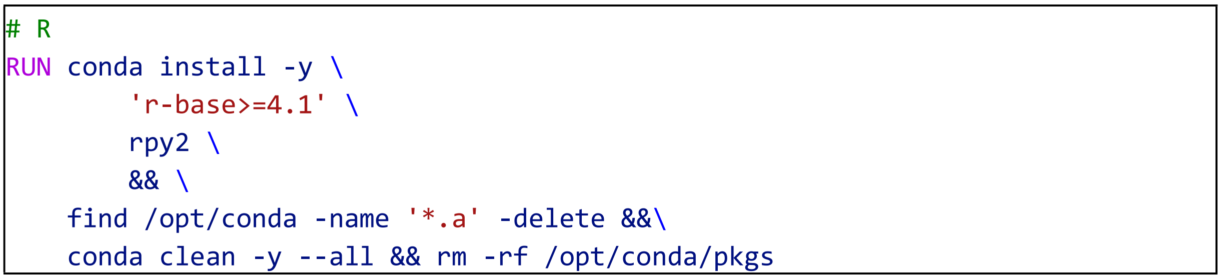
- Imagemagick: an open-source software used to create, modify and convert raster images. Raster images are matrices of cells where each cell have a value representing information.
- Ipywidgets (Jupyter Widgets): this package includes interactive browser control for JupyterNotebooks ranging from basic controls such as sliders and checkboxes to more advance controls such as maps and data grid
- Matplotlib: a package used in Python to plot visualization with an API to include plots in GUI toolkits.
- Networkx: it allows the creation and manipulation of complex networks to study them.
- Nomkl : Used to toggle the usage of different build variant configurations in an environment.
- Notebook (Jupyter Notebook): a web-based open-source software used for interactive development of notebooks, code, and data. It is used to share documents containing equation, visualization, live code, and text
- Pandas : software for data manipulation and analysis.
- Pydot : a Python library used for handling DOT language files used by Graphviz
- Python-graphviz : a simplified interface for Graphviz
- Seaborn : a data visualization library based on previously installed matplotlib
- Scikit-learn : a machine learning tool for predictive data analysis



```
# notebook dependencies
RUN conda install -y \
        graphviz \
        imagemagick \
        ipywidgets \
        matplotlib \
        networkx \
        nomkl \
        notebook \
        pandas \
        pydot \
        python-graphviz \
        seaborn \
        scikit-learn \
        && \
    find /opt/conda -name '*.a' -delete &&\
    conda clean -y --all && rm -rf /opt/conda/pkgs
```

Figure 31: dockerfile part 6

Still with Conda we install R a cross platform free software for logical statistics and graphics.



```
# R
RUN conda install -y \
        'r-base>=4.1' \
        rpy2 \
        && \
    find /opt/conda -name '*.a' -delete &&\
    conda clean -y --all && rm -rf /opt/conda/pkgs
```

Figure 32: dockerfile part 7

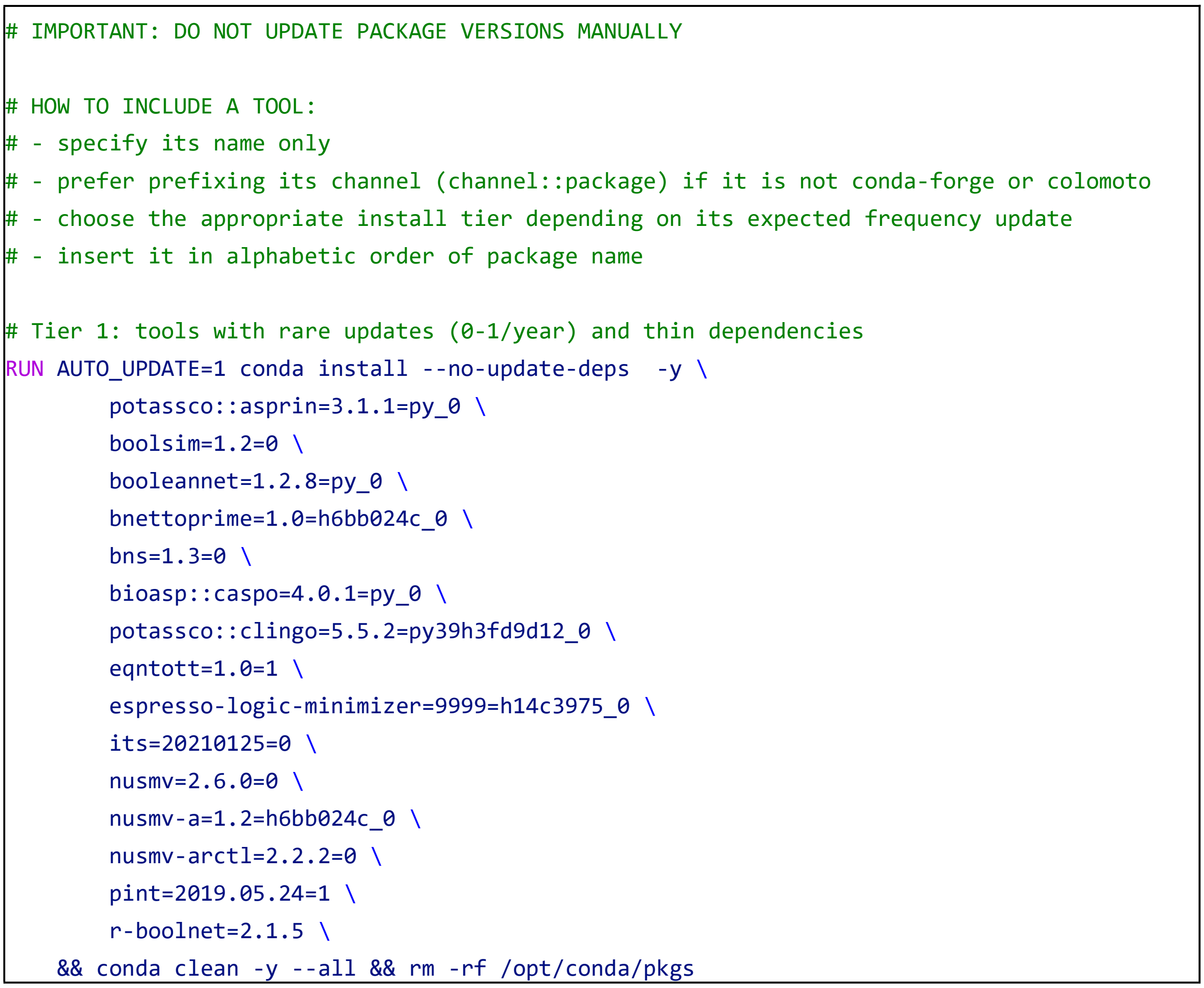
As we can see from the dockerfile, this part is the one where we install all the system biology tools.

They are split in 3 tiers. The first tier includes tools with infrequent updates which occurs once a year or less and do not have many dependencies. The second tier includes tools with more frequent updates from 2 to 4 times a year. The latest regroup tools with frequent updates or light tools.

Tier one includes:

- Aspirin: a framework for qualitative and quantitative optimization in ASP (Answer Set Programing) offering a simple and powerful modeling language to describe combinatorial problems as logic programs.
- Boolsim: used to find attractors and running in C++
- Booleannet: used for Boolean network simulation
- Bnettoprime: used to convert boolNet format to a list of all prime implicants for each regulatory function.
- BNS: used to compute attractors in Boolean Network with Synchronous update
- Caspo: used to automate the inference of logical network from experimental data to help design new experiments and look for intervention strategies to force components in a desired state.
- Clingo: used to compute answer sets for ASP to represent the solution for a given problem.
- Eqntott: used to create truth tables for PLA (Programmable Logic Array) from Boolean equations.
- Espresso heuristic logic minimizer: used to reduce the complexity of Boolean system using heuristics and algorithms.

- ITS : a model checker
- NuSMV: used to check symbolic models for the analysis of finite and infinite state systems
- Pint: used to integrate and annotate SBML files and to study large-scale networks. Pint is based on automata network which encompasses Boolean and discrete models.
- Boolnet: used to create, simulate, and analyze Boolean networks



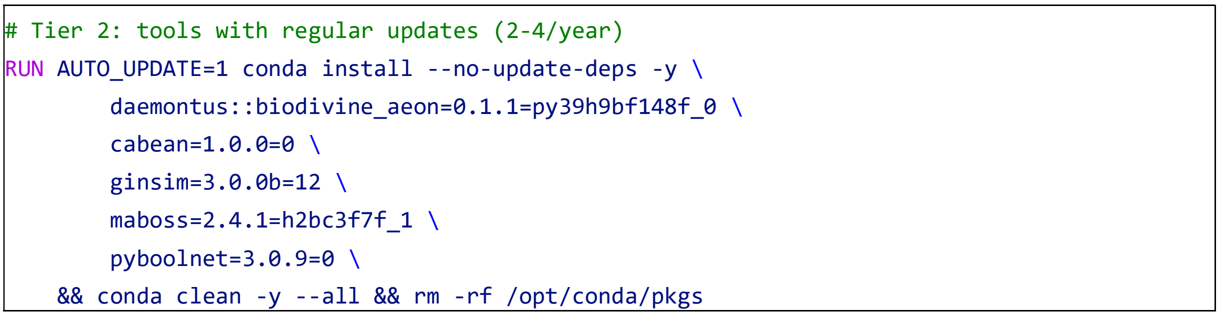
```
# IMPORTANT: DO NOT UPDATE PACKAGE VERSIONS MANUALLY

# HOW TO INCLUDE A TOOL:
# - specify its name only
# - prefer prefixing its channel (channel::package) if it is not conda-forge or colomoto
# - choose the appropriate install tier depending on its expected frequency update
# - insert it in alphabetic order of package name

# Tier 1: tools with rare updates (0-1/year) and thin dependencies
RUN AUTO_UPDATE=1 conda install --no-update-deps  -y \
        potassco::asprin=3.1.1=py_0 \
        boolsim=1.2=0 \
        booleannet=1.2.8=py_0 \
        bnettoprime=1.0=h6bb024c_0 \
        bns=1.3=0 \
        bioasp::caspo=4.0.1=py_0 \
        potassco::clingo=5.5.2=py39h3fd9d12_0 \
        eqntott=1.0=1 \
        espresso-logic-minimizer=9999=h14c3975_0 \
        its=20210125=0 \
        nusmv=2.6.0=0 \
        nusmv-a=1.2=h6bb024c_0 \
        nusmv-arctl=2.2.2=0 \
        pint=2019.05.24=1 \
        r-boolnet=2.1.5 \
    && conda clean -y --all && rm -rf /opt/conda/pkgs
```

Figure 33 : dockerfile part 8

Tier 2 includes :

- Biodivine: used to analyze bifurcation properties of Boolean networks by modifying attributes or parameters of a Boolean network
- Cabean: used to control asynchronous Boolean network
- GINSIM: used for graphical visualization of systems. It can handle synchronous and asynchronous updating.

- Maboss: used to do stochastic Boolean modeling
- Pyboolnet : used to generate, edit and analyze Boolean models

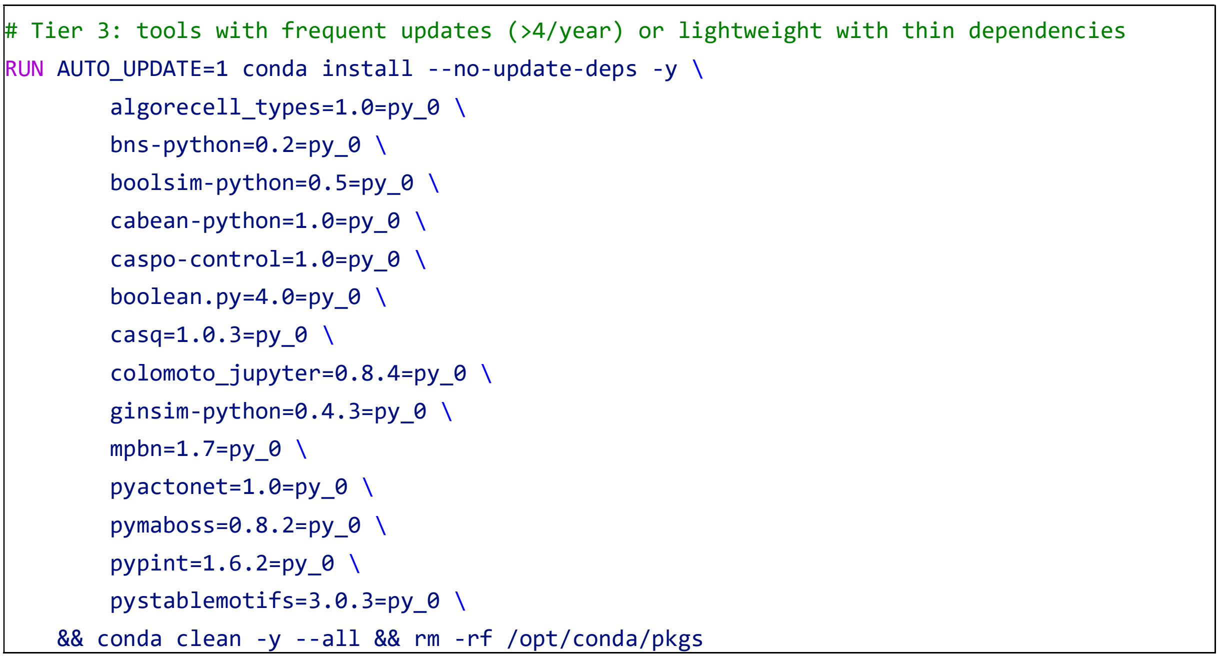


```
# Tier 2: tools with regular updates (2-4/year)
RUN AUTO_UPDATE=1 conda install --no-update-deps -y \
        daemontus::biodivine_aeon=0.1.1=py39h9bf148f_0 \
        cabean=1.0.0=0 \
        ginsim=3.0.0b=12 \
        maboss=2.4.1=h2bc3f7f_1 \
        pyboolnet=3.0.9=0 \
    && conda clean -y --all && rm -rf /opt/conda/pkgs
```

Figure 34: dockerfile part 9

Tier 3 includes :

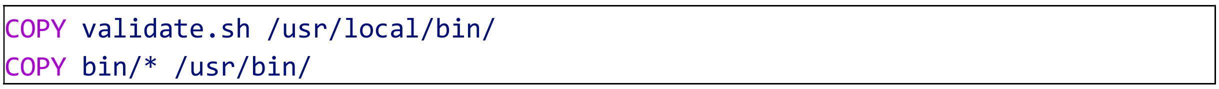
- Algorecell_types: used to do prediction for the control of attractors in Boolean and multi-valued models. It considers perturbations and sequential reprogramming strategies.
- bns-Python: a Python interface designed by colomoto for the tool BNS
- boolsim-Python: a Python interface designed by colomoto for Boolsim
- cabean-Python: a Python interface designed by colomoto for Cabean
- caspo-control: a Python interface to control capso
- boolean.py: a Python library to handle Boolean expression's
- casq: convert cellDesignes model in SBML format
- colomoto_jupyter : a toolset to integrate with CoLoMoTo Jupyter notebook including menus, upload and integration with the network, CellCollectivean NuSMV
- GINsim-Python : a Python interface and an integration with Jupyter Notebook for GINsim and bioLQM
- mpbn : a Python module to analyze Boolean networks
- pyactonet : used to control Boolean network base on abduction of fixed points
- pymaboss : a Python interface designed by colomoto for maboss
- pypint : a Python interface for pint
- pystablemotifs : used to identify and control attractor in Boolean models



```
# Tier 3: tools with frequent updates (>4/year) or lightweight with thin dependencies
RUN AUTO_UPDATE=1 conda install --no-update-deps -y \
        algorecell_types=1.0=py_0 \
        bns-python=0.2=py_0 \
        boolsim-python=0.5=py_0 \
        cabean-python=1.0=py_0 \
        caspo-control=1.0=py_0 \
        boolean.py=4.0=py_0 \
        casq=1.0.3=py_0 \
        colomoto_jupyter=0.8.4=py_0 \
        ginsim-python=0.4.3=py_0 \
        mpbn=1.7=py_0 \
        pyactonet=1.0=py_0 \
        pymaboss=0.8.2=py_0 \
        pypint=1.6.2=py_0 \
        pystablemotifs=3.0.3=py_0 \
    && conda clean -y --all && rm -rf /opt/conda/pkgs
```

Figure 35: dockerfile part 10

The file validate.sh is copied into "/usr/local/bin" and the content of "bin/*" is copied into "/usr/bin."



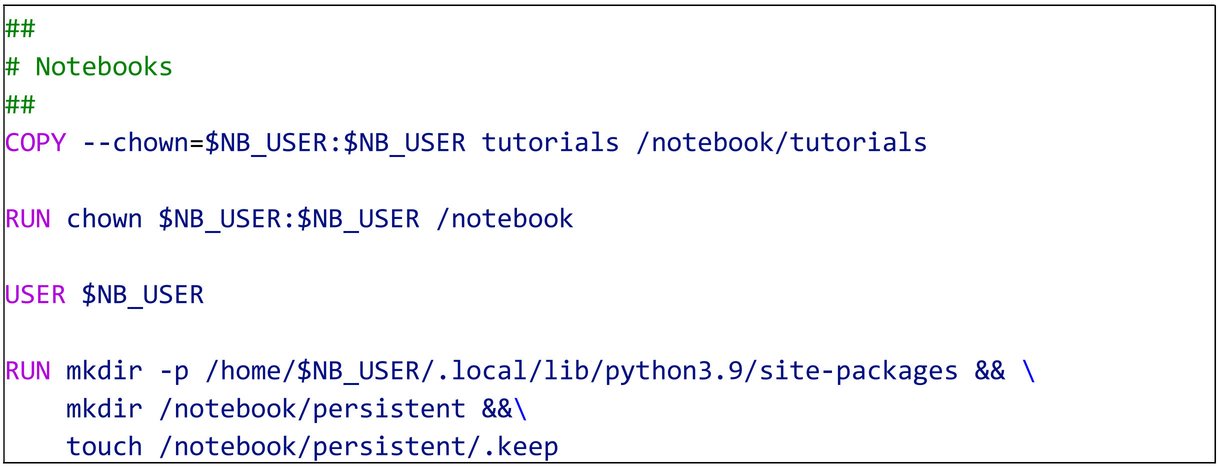
```
COPY validate.sh /usr/local/bin/
COPY bin/* /usr/bin/
```

Figure 36: dockerfile part 11

For the notebooks, the content of tutorials is copied into "/notebook/tutorials" this command is run with the flag "--chown" to give the ownership of this file to our created users.

The owner of "/notebook" and its owner group are set to the user we created, and we switch to the created user with the command: "USER $NB_USER."

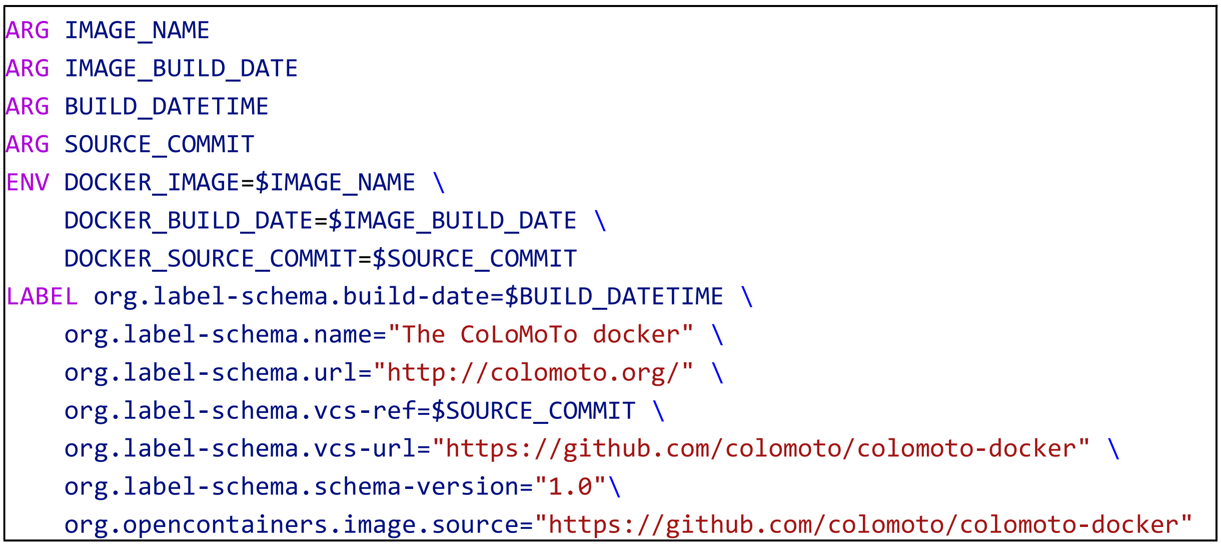
The docker file create subdirectories of the directory until "/home/user/.local/lib/Python3.9/sites-package" and create the directory "/notebook/persistent" and it creates the file ".keep"



```
##
# Notebooks
##
COPY --chown=$NB_USER:$NB_USER tutorials /notebook/tutorials

RUN chown $NB_USER:$NB_USER /notebook

USER $NB_USER

RUN mkdir -p /home/$NB_USER/.local/lib/python3.9/site-packages && \
    mkdir /notebook/persistent &&\
    touch /notebook/persistent/.keep
```

Figure 37: dockerfile part 12

The final part of the docker is designed to allow the user to create a variable that the user can pass during the build of the container. The user can pass a name for the image, the image build date, the build datetime and the source commit. The image is also labeled with the information on the build date time, the source commit and CoLomoTo's information.



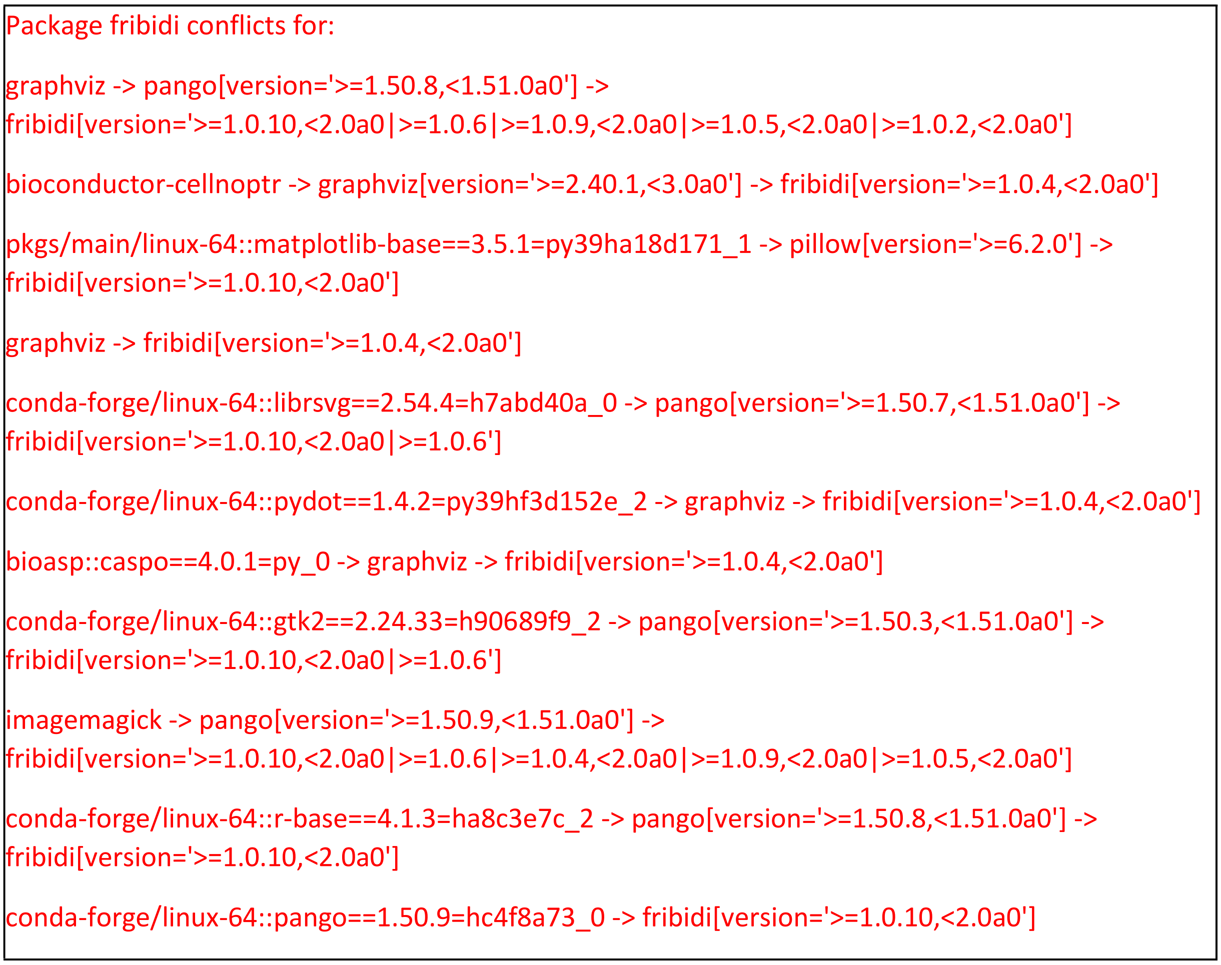
```
ARG IMAGE_NAME
ARG IMAGE_BUILD_DATE
ARG BUILD_DATETIME
ARG SOURCE_COMMIT
ENV DOCKER_IMAGE=$IMAGE_NAME \
    DOCKER_BUILD_DATE=$IMAGE_BUILD_DATE \
    DOCKER_SOURCE_COMMIT=$SOURCE_COMMIT
LABEL org.label-schema.build-date=$BUILD_DATETIME \
    org.label-schema.name="The CoLoMoTo docker" \
    org.label-schema.url="http://colomoto.org/" \
    org.label-schema.vcs-ref=$SOURCE_COMMIT \
    org.label-schema.vcs-url="https://github.com/colomoto/colomoto-docker" \
    org.label-schema.schema-version="1.0"\
    org.opencontainers.image.source="https://github.com/colomoto/colomoto-docker"
```

Figure 38: dockerfile part 13

From this observation of the dockerfile, two necessary steps seems to be required to add new tools in CoLoMoTo.

First they need to be available in Conda, as it is recommended to add new tools with Conda. If the desired tool isn't available, it is possible to add a new package in Conda and make them available for everyone. The documentation to do so is available in Conda's documentation (https://docs.conda.io/projects/conda-build/en/stable/user-guide/tutorials/build-pkgs.html).

Another issue that may arise with adding tools with Conda is the conflict between packages. Indeed if two tools use the same package but with different versions, there will be a conflict between the two versions.



```
Package fribidi conflicts for:

graphviz -> pango[version='>=1.50.8,<1.51.0a0'] ->
fribidi[version='>=1.0.10,<2.0a0|>=1.0.6|>=1.0.9,<2.0a0|>=1.0.5,<2.0a0|>=1.0.2,<2.0a0']

bioconductor-cellnoptr -> graphviz[version='>=2.40.1,<3.0a0'] -> fribidi[version='>=1.0.4,<2.0a0']

pkgs/main/linux-64::matplotlib-base==3.5.1=py39ha18d171_1 -> pillow[version='>=6.2.0'] ->
fribidi[version='>=1.0.10,<2.0a0']

graphviz -> fribidi[version='>=1.0.4,<2.0a0']

conda-forge/linux-64::librsvg==2.54.4=h7abd40a_0 -> pango[version='>=1.50.7,<1.51.0a0'] ->
fribidi[version='>=1.0.10,<2.0a0|>=1.0.6']

conda-forge/linux-64::pydot==1.4.2=py39hf3d152e_2 -> graphviz -> fribidi[version='>=1.0.4,<2.0a0']

bioasp::caspo==4.0.1=py_0 -> graphviz -> fribidi[version='>=1.0.4,<2.0a0']

conda-forge/linux-64::gtk2==2.24.33=h90689f9_2 -> pango[version='>=1.50.3,<1.51.0a0'] ->
fribidi[version='>=1.0.10,<2.0a0|>=1.0.6']

imagemagick -> pango[version='>=1.50.9,<1.51.0a0'] ->
fribidi[version='>=1.0.10,<2.0a0|>=1.0.6|>=1.0.4,<2.0a0|>=1.0.9,<2.0a0|>=1.0.5,<2.0a0']

conda-forge/linux-64::r-base==4.1.3=ha8c3e7c_2 -> pango[version='>=1.50.8,<1.51.0a0'] ->
fribidi[version='>=1.0.10,<2.0a0']

conda-forge/linux-64::pango==1.50.9=hc4f8a73_0 -> fribidi[version='>=1.0.10,<2.0a0']
```

Figure 39: Example of Conflict

Second, it might be required to create a Python interface to allow the usage of the tool within Jupyter Notebooks. In some case, the interface should be able to handle graphical visualization if the implemented tool returns images.

7 Conclusion

At the beginning, this document introduces the notion of system biology. Starting with an historical approach, the root of biology can be observed in the reductionism and mechanistic biology founded in the 17th century. In the beginning of the 20th century criticism were addressed on those methodologies and two new approaches arise: organization of systems in hierarchy and observation of the communication and control within the system. These new ideas were merged with old ones to form the actual science of system biology.

After the historical review, the object of studies themselves are described getting a basic knowledge of what are gene regulatory systems, their components, and the relation between those components.

Later, the introduction of the basic concept of gene network representation shows the different approach used to study a system from static model to dynamic models with discrete or continuous variables. Moreover, different mechanisms were introduced to understand the evolution of the dynamic models in time.

Following the introduction of basic concepts, we observed the different models proposed by scientists to represent in a mathematical way and later in a computational way the organisms. Each representation uses the basic concepts observed earlier and use them to create their models. Each model takes inputs and transforms them in a particular way to give outputs that may vary following the transformation.

The next step into the evolution of system biology was the rise of computer science which based on the previous models allow the scientists to run experiments on computers instead of in their labs and allow them to share their results in a faster way.

However, the computational approach and the diversification of models bring difficulties in the handling of the software because software is developed specifically for specific models and by different teams therefore impacting the combination of models and the reproducibility of experiments.

In this context, the consortium CoLoMoTo introduced a solution named CoLoMoTo to solve these drawbacks. They answer those challenges by proposing a solution based on Docker which embarks multiple tools, allows conversions from one tool to another and improves the reproducibility with notebooks keeping tracks of the different steps of the experiments.

However, the consortium solution doesn't embark all the existing tools used in system biology. Therefore the question is how can we add more system biology tools in CoLoMoTo?

To answer this question, we decided to do a retro-engineering of CoLoMoTo to find out how to integrate new technologies into it.

The retro-engineering shows that CoLoMoTo heavily relies on Python. The script to launch the docker container is based on Python and within the container most technologies are Python based. The ones without a Python base needs interfaces to be usable within the Jupyter notebooks.

To add a new technology, the first step is to upload the package into Conda platform and if needed include a Python interface to use the tool with Jupyter. The second step is to include it in the dockerfile. Then the user needs to build the image based on the dockerfile. Finally, he can create a new container with the new tool and CoLoMoTo existing tools.

The next step to improve the results would be to create a proper example of the introduction of a tool within CoLoMoTo. A tentative was done with CellNOptR but failed due to a conflict into the packages.

8 Bibliography

1. Helikar, T., et al., *The Cell Collective: toward an open and collaborative approach to systems biology.* BMC Syst Biol, 2012. **6**: p. 96.
2. Le Novere, N., et al., *Minimum information requested in the annotation of biochemical models (MIRIAM).* Nat Biotechnol, 2005. **23**(12): p. 1509-15.
3. Consortium, C. *Software tools for logical modelling*. 2018; Available from: http://www.colomoto.org/software/.
4. Consortium, C. *Methods for Logical Models*. 2018; Available from: http://www.colomoto.org/methods/.
5. Trewavas, A., *A brief history of systems biology. "Every object that biology studies is a system of systems." Francois Jacob (1974).* Plant Cell, 2006. **18**(10): p. 2420-30.
6. Williams, R.J., *Biochemical Individuality The Basis for the Genetotrophic Concept*. 1959.
7. Silverthorn, D.U., *Pysiologie Humaine Une approche intégrée*. 4 ed. 2007.
8. Batt, G., et al., *Genetic network analyzer: a tool for the qualitative modeling and simulation of bacterial regulatory networks.* Methods Mol Biol, 2012. **804**: p. 439-62.
9. Naldi, A., *BioLQM: A Java Toolkit for the Manipulation and Conversion of Logical Qualitative Models of Biological Networks.* Front Physiol, 2018. **9**: p. 1605.
10. Stoll, G., et al., *MaBoSS 2.0: an environment for stochastic Boolean modeling.* Bioinformatics, 2017. **33**(14): p. 2226-2228.
11. Samaga, R. and S. Klamt, *Modeling approaches for qualitative and semi-quantitative analysis of cellular signaling networks.* Cell Commun Signal, 2013. **11**(1): p. 43.
12. de Jong, H., *Modeling and simulation of genetic regulatory systems: a literature review.* J Comput Biol, 2002. **9**(1): p. 67-103.
13. Paulevé, L., *PINT: A Static Analyzer for Transient Dynamics of Qualitative Networks with IPython Interface.*
14. Wang, R.S., A. Saadatpour, and R. Albert, *Boolean modeling in systems biology: an overview of methodology and applications.* Phys Biol, 2012. **9**(5): p. 055001.
15. Sedghamiz, H., Chen, W., Rice, M., Whitley, D., & Broderick, G., *Selecting Optimal Models Based on Efficiency and Robustness in Multi-valued Biological Networks.* 2017.
16. Faure, A., et al., *Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle.* Bioinformatics, 2006. **22**(14): p. e124-31.
17. Rene Thomas, R.D.A., *Biological Feedback*. 1990.
18. Tomassini, M., *Generalized Automata Networks.* 2006.
19. Holehouse, J., Z. Cao, and R. Grima, *Stochastic Modeling of Autoregulatory Genetic Feedback Loops: A Review and Comparative Study.* Biophys J, 2020. **118**(7): p. 1517-1525.
20. Szekely, T., Jr. and K. Burrage, *Stochastic simulation in systems biology.* Comput Struct Biotechnol J, 2014. **12**(20-21): p. 14-25.
21. Albert, I., et al., *Boolean network simulations for life scientists.* Source Code Biol Med, 2008. **3**: p. 16.
22. Abou-Jaoude, W., et al., *Logical Modeling and Dynamical Analysis of Cellular Networks.* Front Genet, 2016. **7**: p. 94.
23. Collombet, S., et al., *Logical modeling of lymphoid and myeloid cell specification and transdifferentiation.* Proc Natl Acad Sci U S A, 2017. **114**(23): p. 5792-5799.
24. Levy, N., et al., *Prediction of Mutations to Control Pathways Enabling Tumor Cell Invasion With the CoLoMoTo Interactive Notebook (Tutorial).* Front Physiol, 2018. **9**: p. 787.
25. Naldi, A., et al., *The CoLoMoTo Interactive Notebook: Accessible and Reproducible Computational Analyses for Qualitative Biological Networks.* Front Physiol, 2018. **9**: p. 680.
26. Naldi, A., et al., *Logical Modeling and Analysis of Cellular Regulatory Networks With GINsim 3.0.* Front Physiol, 2018. **9**: p. 646.

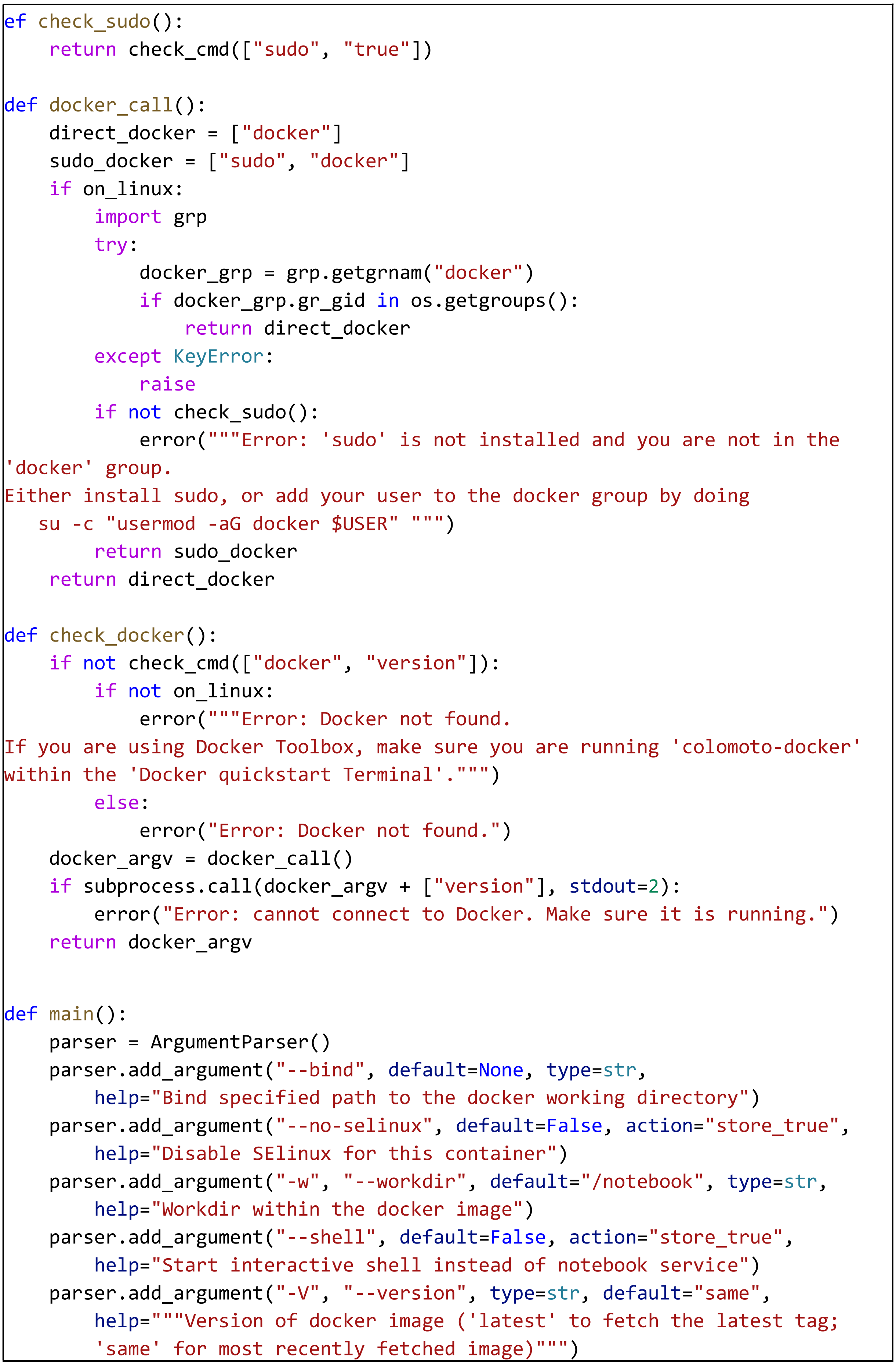
27. Noel, V., et al., *WebMaBoSS: A Web Interface for Simulating Boolean Models Stochastically.* Front Mol Biosci, 2021. **8**: p. 754444.
28. Biere, A. and D. Kröning, *SAT-Based Model Checking*, in *Handbook of Model Checking*. 2018. p. 277-303.
29. Cimatti, A., et al., *NuSMV: A New Symbolic Model Verifier*, in *Computer Aided Verification*. 1999. p. 495-499.
30. Cimatti, A., et al., *NuSMV 2: An OpenSource Tool for Symbolic Model Checking*, in *Computer Aided Verification*. 2002. p. 359-364.
31. Chaki, S. and A. Gurfinkel, *BDD-Based Symbolic Model Checking*, in *Handbook of Model Checking*. 2018. p. 219-245.
32. Chaouiya, C., et al., *SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools.* BMC Syst Biol, 2013. **7**: p. 135.
33. Klarner, H., A. Streck, and H. Siebert, *PyBoolNet: a Python package for the generation, analysis and visualization of boolean networks.* Bioinformatics, 2017. **33**(5): p. 770-772.

9 Appendices

9.1 Colomoto_docker.py code:

Source : https://pypi.org/project/colomoto-docker/

```python
#!/usr/bin/env python

from __future__ import print_function

from argparse import ArgumentParser, REMAINDER
import os
from contextlib import closing
from getpass import getuser
import platform
import re
import socket
import subprocess
import sys
import webbrowser

on_linux = platform.system() == "Linux"

pat_tag = re.compile(r"\d{4}-\d{2}-\d{2}")

persistent_volume = "colomoto-{}".format(getuser())
persistent_dir = "persistent"

official_image = "colomoto/colomoto-docker"
official_alt = [
    "ghcr.io/colomoto/colomoto-docker:{tag}",
    "docker.pkg.github.com/colomoto/colomoto-docker/colomoto-docker:{tag}",]
def error(msg):
    print(msg, file=sys.stderr)
    sys.exit(1)

def info(msg):
    print(msg, file=sys.stderr)

def check_cmd(argv):
    DEVNULL = subprocess.DEVNULL if hasattr(subprocess, "DEVNULL") \
            else open(os.devnull, 'w')
    try:
        subprocess.call(argv, stdout=DEVNULL, stderr=DEVNULL, close_fds=True)
        return True
    except:
        return False
```



```
ef check_sudo():
    return check_cmd(["sudo", "true"])

def docker_call():
    direct_docker = ["docker"]
    sudo_docker = ["sudo", "docker"]
    if on_linux:
        import grp
        try:
            docker_grp = grp.getgrnam("docker")
            if docker_grp.gr_gid in os.getgroups():
                return direct_docker
        except KeyError:
            raise
        if not check_sudo():
            error("""Error: 'sudo' is not installed and you are not in the
'docker' group.
Either install sudo, or add your user to the docker group by doing
   su -c "usermod -aG docker $USER" """)
        return sudo_docker
    return direct_docker

def check_docker():
    if not check_cmd(["docker", "version"]):
        if not on_linux:
            error("""Error: Docker not found.
If you are using Docker Toolbox, make sure you are running 'colomoto-docker'
within the 'Docker quickstart Terminal'.""")
        else:
            error("Error: Docker not found.")
    docker_argv = docker_call()
    if subprocess.call(docker_argv + ["version"], stdout=2):
        error("Error: cannot connect to Docker. Make sure it is running.")
    return docker_argv


def main():
    parser = ArgumentParser()
    parser.add_argument("--bind", default=None, type=str,
        help="Bind specified path to the docker working directory")
    parser.add_argument("--no-selinux", default=False, action="store_true",
        help="Disable SElinux for this container")
    parser.add_argument("-w", "--workdir", default="/notebook", type=str,
        help="Workdir within the docker image")
    parser.add_argument("--shell", default=False, action="store_true",
        help="Start interactive shell instead of notebook service")
    parser.add_argument("-V", "--version", type=str, default="same",
        help="""Version of docker image ('latest' to fetch the latest tag;
        'same' for most recently fetched image)""")
    parser.add_argument("--port", default=0, type=int,
        help="Local port")
    parser.add_argument("--image", default=official_image,
        help="Docker image")
    parser.add_argument("--no-browser", default=False, action="store_true",
        help="Do not start the browser")
    parser.add_argument("--unsafe-ssl", default=False, action="store_true",
        help="Do not check for SSL certificates")
    parser.add_argument("--no-update", default=False, action="store_true",
        help="Do not check for image update")
    parser.add_argument("--cleanup", default=False, action="store_true",
        help="Cleanup old images")

    group = parser.add_argument_group("docker run options")
    group.add_argument("-e", "--env", action="append",
        help="Set environment variables")
    group.add_argument("--name", help="Name of the container")
    group.add_argument("-v", "--volume", action="append",
        help="Bind mount a volume")
    docker_run_opts = ["env", "name", "volume"]

    parser.add_argument("command", nargs=REMAINDER, help="Command to run
instead of colomoto-nb")
    args = parser.parse_args()

    image_tag = args.version

    if args.version == "same":
        output = subprocess.check_output(["docker", "images", "-f",
                                    "reference=colomoto/colomoto-docker",
                                    "--format", "{{.Tag}}"])
        output = output.decode()
        if not output:
            args.version = "latest"
        else:
            image_tag = output.split("\n")[0]

    docker_argv = check_docker()

    if args.version == "latest" and not args.no_update:
        import json
        try:
            from urllib.request import urlopen
        except ImportError:
            from urllib2 import urlopen
        if args.unsafe_ssl or not on_linux:
            # disable SSL verification...
            import ssl
            ssl._create_default_https_context = ssl._create_unverified_context

        info("# querying for latest tag of {}...".format(args.image))
        url_api =
"https://registry.hub.docker.com/v1/repositories/{}/tags".format(args.image)
        tags = []
        q = urlopen(url_api)
        data = q.read().decode("utf-8")
        r = json.loads(data)
        q.close()
        tags = [t["name"] for t in r if pat_tag.match(t["name"])]
        if not tags:
            info("# ... none found! use 'latest'")
            image_tag = "latest"
        else:
            image_tag = max(tags)

    image = "%s:%s" % (args.image, image_tag)
    info("# using {}".format(image))

    if not args.no_update \
        and (image_tag.startswith("next") \
            or not subprocess.check_output(docker_argv + ["images", "-q",
image])):
        if args.image == official_image:
            pull = subprocess.run(docker_argv + ["pull", image])
            if pull.returncode != 0:
                info(f"The image {image} does not exists on hub.docker.com,
falling back to mirrors..")
                for ref in official_alt:
                    altimage = ref.format(tag=image_tag)
                    pull = subprocess.run(docker_argv + ["pull", altimage])
                    if pull.returncode == 0:
                        info(f".. using {altimage}")
                        subprocess.check_call(docker_argv + ["tag", altimage,
image])
                        subprocess.check_call(docker_argv + ["rmi", altimage])
                        break
                raise Exception("Docker image not found, maybe wrong
version?")
        else:
            subprocess.check_call(docker_argv + ["pull", image])
```

```
if args.cleanup:
        output = subprocess.check_output(docker_argv + ["images", "-f",
                                    "reference=colomoto/colomoto-docker",
                                    "--format", "{{.Tag}} {{.ID}}"])
        todel = []
        for line in output.decode().split("\n"):
            if not line:
                continue
            tag, iid = line.split()
            if tag == image_tag:
                continue
            if tag == "<none>":
                todel.append(iid)
            else:
                todel.append("{}:{}".format(args.image, tag))
        if todel:
            argv = docker_argv + ["rmi"] + todel
            info("# {}".format(" ".join(argv)))
            subprocess.call(argv)

    argv = docker_argv + ["run", "-it", "--rm"]
    if args.no_selinux:
        argv += ["--security-opt", "label:disable"]

    if args.bind:
        argv += ["--volume", "%s:%s" % (os.path.abspath(args.bind),
args.workdir)]
    else:
        persistent_mount = "%s/%s" % (args.workdir, persistent_dir)
        argv += ["--volume", "%s:%s" % (persistent_volume, persistent_mount)]

    argv += ["-w", args.workdir]
    if not args.shell and not args.command:
        container_ip = "127.0.0.1"
        docker_machine = os.getenv("DOCKER_MACHINE_NAME")
        if docker_machine:
            container_ip = subprocess.check_output(["docker-machine", "ip",
docker_machine])
            container_ip = container_ip.decode().strip().split("%")[0]
        if args.port == 0:
            # find next available
            for port in range(8888, 65535):
                with closing(socket.socket(socket.AF_INET,
socket.SOCK_STREAM)) as s:
                    dest_addr = (container_ip, port)
                    if s.connect_ex(dest_addr):
                        break
        else:
            port = args.port

        argv += ["-p", "%s:8888" % port]


    # forward proxy configuration
    for env in ["HTTP_PROXY", "HTTPS_PROXY", "FTP_PROXY", "NO_PROXY"]:
        if env in os.environ:
            argv += ["-e", env]

    def easy_volume(val):
        orig, dest = val.split(":")
        if dest[0] != "/":
            dest = os.path.abspath(os.path.join(args.workdir, dest))
        if orig[0] != "/" and os.path.isdir(orig):
            orig = os.path.abspath(orig)
        return "%s:%s" % (orig, dest)

    for opt in docker_run_opts:
        if getattr(args, opt) is not None:
            val = getattr(args, opt)
            if isinstance(val, list):
                for v in val:
                    if opt == "volume":
                        v = easy_volume(v)
                    argv += ["--%s"%opt, v]
            else:
                argv += ["--%s" % opt, val]

    argv += [image]
    if args.shell:
        argv += ["bash"]
    elif args.command:
        argv += args.command

    info("# %s" % " ".join(argv))

    if not args.shell and not args.command and not args.no_browser:

        p = subprocess.Popen(argv, stdout=subprocess.PIPE)

        launched = False
        while True:
            line = os.read(p.stdout.fileno(), 1024)
            if line:
                os.write(sys.stdout.fileno(), line)
                line = line.decode()
                if not launched and " is running at:" in line:
                    launched = True
                    try:
                        webbrowser.open("http://{}:{}".format(container_ip,
port))
                    except:
                        info("""
Please open your web-browser to the following address:

    http://{}:{}

""".format(container_ip, port))
            elif p.poll() is not None:
                break

    else:
        os.execvp(argv[0], argv)

if __name__ == "__main__":
    main()
```

9.2 Colomoto-docker / Dockerfile:

Source : https://github.com/colomoto/colomoto-docker/blob/master/Dockerfile

```
FROM debian:stable-20220328-slim

ENV LANG=C.UTF-8 LC_ALL=C.UTF-8
ENV PATH /opt/conda/bin:$PATH

ARG NB_USER=user
ARG NB_UID=1000
RUN useradd -u $NB_UID -m -d /home/user -s /bin/bash $NB_USER

EXPOSE 8888
WORKDIR /notebook
ENTRYPOINT ["/usr/bin/tini", "--", "colomoto-env"]
CMD ["colomoto-nb", "--NotebookApp.token="]

##
## distribution packages
##
RUN apt-get update --fix-missing && \
    apt-get install -y --no-install-recommends \
        bzip2 \
        ca-certificates \
        wget \
        openjdk-11-jre-headless \
        && \
    apt clean -y && \
    rm -rf /var/lib/apt/lists/*

#
# tini for avoiding zombie processes (useless with Docker 1.13)
#
RUN TINI_VERSION="0.19.0" && \
    wget --quiet
https://github.com/krallin/tini/releases/download/v${TINI_VERSION}/tini_${TINI
_VERSION}-amd64.deb && \
    dpkg -i tini_${TINI_VERSION}-amd64.deb && \
    rm *.deb

#
# base conda environment
#
# package versions in this section are not pinned unless necessary
#
```

```
RUN CONDA_VERSION="py39_4.11.0" && \
    echo 'export PATH=/opt/conda/bin:$PATH' > /etc/profile.d/conda.sh && \
    wget --quiet https://repo.continuum.io/miniconda/Miniconda3-
${CONDA_VERSION}-Linux-x86_64.sh -O ~/miniconda.sh && \
    /bin/bash ~/miniconda.sh -b -p /opt/conda && \
    rm ~/miniconda.sh && \
    conda config --set auto_update_conda False && \
    conda config --add channels colomoto && \
    conda config --add channels conda-forge && \
    conda install --no-update-deps -y \
        -c colomoto/label/fake \
        openjdk \
        pyqt && \
    find /opt/conda -name '*.a' -delete &&\
    conda clean -y --all && rm -rf /opt/conda/pkgs

# notebook dependencies
RUN conda install -y \
        graphviz \
        imagemagick \
        ipywidgets \
        matplotlib \
        networkx \
        nomkl \
        notebook \
        pandas \
        pydot \
        python-graphviz \
        seaborn \
        scikit-learn \
        && \
    find /opt/conda -name '*.a' -delete &&\
    conda clean -y --all && rm -rf /opt/conda/pkgs

# R
RUN conda install -y \
        'r-base>=4.1' \
        rpy2 \
        && \
    find /opt/conda -name '*.a' -delete &&\
    conda clean -y --all && rm -rf /opt/conda/pkgs
```

```
# IMPORTANT: DO NOT UPDATE PACKAGE VERSIONS MANUALLY

# HOW TO INCLUDE A TOOL:
# - specify its name only
# - prefer prefixing its channel (channel::package) if it is not conda-forge or colomoto
# - choose the appropriate install tier depending on its expected frequency update
# - insert it in alphabetic order of package name

# Tier 1: tools with rare updates (0-1/year) and thin dependencies
RUN AUTO_UPDATE=1 conda install --no-update-deps  -y \
        potassco::asprin=3.1.1=py_0 \
        boolsim=1.2=0 \
        booleannet=1.2.8=py_0 \
        bnettoprime=1.0=h6bb024c_0 \
        bns=1.3=0 \
        bioasp::caspo=4.0.1=py_0 \
        potassco::clingo=5.5.2=py39h3fd9d12_0 \
        eqntott=1.0=1 \
        espresso-logic-minimizer=9999=h14c3975_0 \
        its=20210125=0 \
        nusmv=2.6.0=0 \
        nusmv-a=1.2=h6bb024c_0 \
        nusmv-arctl=2.2.2=0 \
        pint=2019.05.24=1 \
        r-boolnet=2.1.5 \
    && conda clean -y --all && rm -rf /opt/conda/pkgs

# Tier 2: tools with regular updates (2-4/year)
RUN AUTO_UPDATE=1 conda install --no-update-deps -y \
        daemontus::biodivine_aeon=0.1.1=py39h9bf148f_0 \
        cabean=1.0.0=0 \
        ginsim=3.0.0b=12 \
        maboss=2.4.1=h2bc3f7f_1 \
        pyboolnet=3.0.9=0 \
    && conda clean -y --all && rm -rf /opt/conda/pkgs
```

```
# Tier 3: tools with frequent updates (>4/year) or lightweight with thin
dependencies
RUN AUTO_UPDATE=1 conda install --no-update-deps -y \
        algorecell_types=1.0=py_0 \
        bns-python=0.2=py_0 \
        boolsim-python=0.5=py_0 \
        cabean-python=1.0=py_0 \
        caspo-control=1.0=py_0 \
        boolean.py=4.0=py_0 \
        casq=1.0.3=py_0 \
        colomoto_jupyter=0.8.4=py_0 \
        ginsim-python=0.4.3=py_0 \
        mpbn=1.7=py_0 \
        pyactonet=1.0=py_0 \
        pymaboss=0.8.2=py_0 \
        pypint=1.6.2=py_0 \
        pystablemotifs=3.0.3=py_0 \
    && conda clean -y --all && rm -rf /opt/conda/pkgs

COPY validate.sh /usr/local/bin/
COPY bin/* /usr/bin/
##
# Notebooks
##
COPY --chown=$NB_USER:$NB_USER tutorials /notebook/tutorials

RUN chown $NB_USER:$NB_USER /notebook

USER $NB_USER

RUN mkdir -p /home/$NB_USER/.local/lib/python3.9/site-packages && \
    mkdir /notebook/persistent &&\
    touch /notebook/persistent/.keep

ARG IMAGE_NAME
ARG IMAGE_BUILD_DATE
ARG BUILD_DATETIME
ARG SOURCE_COMMIT
ENV DOCKER_IMAGE=$IMAGE_NAME \
    DOCKER_BUILD_DATE=$IMAGE_BUILD_DATE \
    DOCKER_SOURCE_COMMIT=$SOURCE_COMMIT
LABEL org.label-schema.build-date=$BUILD_DATETIME \
    org.label-schema.name="The CoLoMoTo docker" \
    org.label-schema.url="http://colomoto.org/" \
    org.label-schema.vcs-ref=$SOURCE_COMMIT \
    org.label-schema.vcs-url="https://github.com/colomoto/colomoto-docker" \
    org.label-schema.schema-version="1.0"\
    org.opencontainers.image.source="https://github.com/colomoto/colomoto-
docker"
```