TÉCNICO LISBOA

Attractor reachability estimation in logical models

Yu Cheng

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Pedro Tiago Gonçalves Monteiro
Prof. Claudine Chaouiya

Examination Committee

Chairperson: Prof. José Luís Brinquete Borbinha
Supervisor: Prof. Pedro Tiago Gonçalves Monteiro
Member of the Committee: Profª. Elisabeth Remy

November 2021

Acknowledgments

My deepest gratitude goes to Prof. Pedro and Prof.^{a} Claudine. They have always been available to share their knowledge, train my spirit of criticism, teach me the right approach to conduct a research, clarify all my doubts and provide any support that has made this Thesis possible. It was a pleasure to have worked under their supervision and make a contribution to the Colomoto community.

I also would like to thank my beloved parents, SuiMai and LiPing, for their caring, moral and financial supports over all these years.

Finally, my special gratitude to XiaoYi, my beloved sister, and XingLei, my best friend, for their availabilities and encouragements during the pandemic.

Abstract

Mathematical modeling of biological systems is often used as a tool to simulate and validate networks underlying biological processes. In particular, logical models are well-suited to capture dynamical properties of regulatory networks where biological phenotypes, *e.g.*, cell phenotypes, are associated with model attractors (stable states, or complex attractors). Identification of model attractors and quantification of their reachability are particularly relevant. However, as the size of a network model increases, the analysis of such asymptotical behavior turns particularly difficult. Indeed, the number of model states increases exponentially with the number of model components, hindering model analysis.

Several studies have been conducted to tackle the attractors determination and the quantification of their reachability. In particular, the algorithm Avatar was proposed as an adaptation of classical Monte Carlo simulations, offering a good performance for models with large and intertwined transient and terminal cycles. Nevertheless, Avatar and Monte Carlo only perform simulations considering the asynchronous updating scheme with uniform probabilities associated to concurrent transitions.

In this work, we aim to extend these algorithms to support novel updating schemes and calculate transition probabilities according to qualitative rates associated to component updates. We have provided convenient command line features to ease launching these algorithms, and developed sets of automatic tests to ensure their correctness. As a result, new functionalities have been integrated into bioLQM, a java library devoted to logical modeling, broadening the use of these new features. Finally, we further assessed the relevance of these developments by considering their application to real case studies.

Keywords

Logical modeling; Attractors; Reachability quantification; Markov process; Biological regulatory networks.

Resumo

A modelagem matemática de sistemas biológicos é frequentemente usada como uma ferramenta para simular e validar redes que descrevem processos biológicos. Em particular, os modelos lógicos são adequados para capturar propriedades dinâmicas de redes regulatórias onde fenótipos biológicos,*e.g.*, fenótipos celulares, estão associados a atratores de modelo. A identificação dos atratores e a quantificação de sua alcançabilidade são particularmente relevantes. No entanto, à medida que o tamanho do modelo aumenta, a análise de tal comportamento assintótico torna-se particularmente difícil. De facto, à medida que mais nós são introduzidos, o número de estados do modelo aumenta exponencialmente, o que dificulta a análise do modelo.

Vários estudos foram conduzidos para abordar a determinação de atratores e a quantificação de sua alcançabilidade. Em particular, o algoritmo Avatar foi proposto como uma adaptação da simulação clássica de Monte Carlo que oferece um bom desempenho para modelos com ciclos terminais e transientes grandes. No entanto, Avatar e Monte Carlo só realizam simulações considerando o esquema de atualização assíncrona com probabilidades uniformes associadas a transições concorrentes.

Para o propósito deste trabalho, pretendemos estender esses algoritmos para suportar um novos esquemas de atualização e calcular as probabilidades de transição de acordo com os rates associadas a cada atualização de componente. Desenvolvemos conjuntos de testes automáticos para garantir a sua correção. Os novos algoritmos poderão ser usados na linha de commando e foram integradas ao bioLQM, uma biblioteca java dedicada à modelagem lógica, o que divulga o uso dessas ferramentas. Finalmente, avaliamos ainda mais a relevância desses desenvolvimentos, considerando sua aplicação a estudos de caso reais.

Palavras Chave

Modelagem lógica; Atratores; Quantificação da alcançabilidade, Cadeia de Markov, Rede de transcrição genética.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

API	Application Program Interface
BDD	Boolean Decision Diagram
CTL	Computation Tree Logic
DFS	Depth First Search
HTG	Hierarchical Transition Graph
LRG	Logical Regulatory Graph
MDD	Multi-valued Decision Diagram
ODE	Ordinary Differential Equation
SAT	Boolean Satisfiability Problem
SCC	Strongly Connected Components
STG	State Transition Graph

1

Introduction

Contents

Nowadays, the number of studied biological systems is increasing at a very fast pace, and it gives the motivation and importance of modeling it to ease the understanding of complex biological phenomena, from biochemical pathways to cell differentiation. Mathematical modeling of biological regulatory networks [4] has proven to be a good approach as it describes networks underlying the control of biological processes, in a precise and unambiguous way. Specifically, modeling is effective to understand the role of individual components and interactions, to validate or question the current understanding and to predict behaviors upon perturbations.

There are two major types of modeling approaches [5]: quantitative modeling and qualitative modeling. In the first framework, the most commonly used formalism is Ordinary Differential Equation (ODE), where the model includes concentrations of chemical species represented by real numbers at a time t. In this context, regulatory interactions between the components, *i.e.,* synthesis or degradation, are defined by non linear functions, and the evolution of these concentrations are defined by ODE. Nevertheless, this approach has two main limitations: first, the approach is deterministic and thus focuses on an average behavior, therefore it is not adequate for heterogeneous systems; second, it is difficult to assign values to the kinetic parameters of the model.

In the second framework, the most used formalism is Boolean network. Here the model is represented by regulatory nodes, corresponding to components of the biological system, regulatory interactions, corresponding to arrows from source nodes to target nodes, and regulatory rules, which define the associated functions of regulatory nodes to reach their activity levels. As a result, model states are represented by the (Boolean) activity levels of the nodes, and the dynamics is defined by the set of transitions between the states, represented as a State Transition Graph (STG). The main drawback of this approach is the combinatorial explosion: the size of the state transition graph can reach up to $2^{\sharp nodes}$, turning most of algorithms for model analysis NP-complete.

Several tools have been developed, implementing these modeling frameworks for biological networks. In this work, the focus is on logical modeling (the extension of Boolean networks to account for multiple discrete values).

1.1 Objectives

This work focuses on logical models as they are appropriate to capture salient dynamical properties of regulatory networks. Much work has been conducted by several groups who contributed with various computer tools and methods for the definition and dynamical analyses of logical models.

The dynamics of a model is represented in terms of an STG. Since the number of states is finite, simulations always end up in a steady behavior, corresponding to a single state or not. We are particularly interested in such asymptotic behaviors, which, in the mathematical field of dynamical systems, are denoted *attractors*. They correspond to the terminal Strongly Connected Components (SCC) in the context of the STG, and they reveal states of relevant biological properties (*e.g.* long term oscillations).

Apart from the identification of attractors, we are also interested in the quantification of their reachability. This provides relevant predictions as attractors can reflect biological responses. For instance, when several attractors arise, we are able to assess the most probable attractors, thus predicting the most likely behavior feasible under a specific condition.

The identification and estimation of the reachability of the model attractors is not an easy challenge, though. As the number of model components increases, the number of states of the model grows exponentially, hindering the analysis and identification of the model properties.

The algorithm Avatar [1] was proposed to assess the reachability of such property. It is an adapted Monte-Carlo approach, which is proved to have good performance for models with large transient and terminal cycles. It explores iteratively the state space by performing random sampling and aggregates these results to estimate the probability of reaching attractors. Different from a classical Monte-Carlo simulation, it offers mechanisms to identify terminal cycles and to avoid getting trapped in transient cycles. Nevertheless, it has two limitations: first, the algorithm is performed for simulations using solely asynchronous updating scheme; second, it assumes that the probabilities of concurrent transitions are uniformly distributed.

The objective of this work is to improve Avatar and Monte Carlo algorithms and their implementations to overcome limitations mentioned above. Hence, new functionalities were developed and integrated into bioLQM [6], which is a Java toolkit for the conversion, transformation, and analysis of Logical Qualitative Models of biological regulatory networks.

1.2 Dissertation Outline

This document is organized in the following way.

Chapter 2 introduces notions and fundamentals concerning the logical modeling formalism.

Chapter 3 presents the related work developed by community for the identification and estimation of attractors reachability.

Chapter 4 presents a solution for Avatar and Monte Carlo to overcome aforementioned limitations and the computational works performed to integrate them in bioLQM.

Chapter 5 demonstrates the performance and the ability of the revised version of algorithms to identify attractors and to quantify their reachability, when evaluating on test model, synthetic model and biological models.

Chapter 6 summarizes all the works performed and proposes possible future works.

Appendix A includes a manual which provides additional information to use our tools in command line through bioLQM [6].

2

Background

Contents

This chapter presents the fundamental concepts concerning the logical modeling. It is organized in five parts: a briefly presentation of logical regulatory graphs; a description and representation of the dynamics that a model defines and its properties; a brief review of methods for model analysis; a brief introduction to Markov process; and finally, the application of Absorbing Markov chain to the dynamics of a model.

2.1 Logical regulatory graph

A Logical Regulatory Graph (LRG) can be interpreted as the composition of three components:

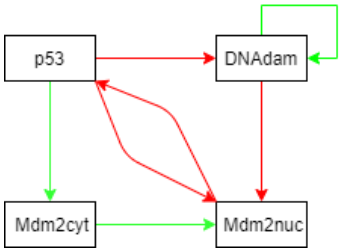
- Regulatory nodes, defining set of nodes representing the biological factors. Each node has a maximum functional level of activity or concentration.
- Regulatory interactions, defining the interaction arcs, *i.e.*, synthesis or degradation, between the nodes in the model. An interaction is active when the level of its source is equal or above its threshold.
- Regulatory rules, that specify the logical rules for each node to reach a given activity level.

Note that the definition of adequate regulatory rules is necessary to ensure the desired effects of each interaction on the target nodes, *e.g.*, when a negative interaction operates on the target node, the rule should reflect this negative effect, decreasing the level of the target node. In any case, the signs of interactions can be derived from the logical functions and an interaction is functional when it affects the focal value of its target [7].

Formally, a logical regulatory graph can be defined as:

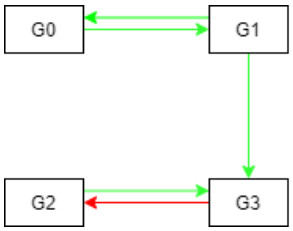
- A set of n regulatory components $G = \{g_1, g_2, \dots g_n\}$, where each g_i is associated with a variable s_i which takes its values in $0, \dots, max_i$. The (finite) state space S is defined as the Cartesian product $\prod_{i=1,\dots,n}\{0, \dots, max_i\}$;
- For each g_i, K_i denotes its logical regulatory function and defines its value, *i.e.*, $K_i : S \rightarrow \{0, \dots, max_i\}$ specifies the evolution of g_i; $\forall s \in S$, $K_i(s)$ is the target value of g_i that depends on the state s.

Figure 2.1 presents the topological representation of the regulatory network of LRG of p53-Mdm2 model [8] with its associated regulatory rules. Moreover, a toy model is presented and its representation and regulatory rules are illustrated in Figure 2.2.



Regulatory node	Target levels	Boolean rules
P53	2	!Mdm2nuc
Mdm2cyt	2	p53
	1	!p53
Mdm2nuc	1	Mdm2cyt:2 | (Mdm2cyt:1 & !p53 & !DNAdam)
DNAdam	1	DNAdam & !p53

Figure 2.1: LRG of p53-Mdm2 model



Regulatory node	Target levels	Boolean rules
G0	1	G1
G1	1	G0
G2	1	!G3
G3	1	G1 | G2

Figure 2.2: LRG of toy model

The regulatory components of the model are nodes with respective label. The edges between nodes denotes the regulatory interaction, green arrows denotes positive interaction (synthesis) and red arrows denotes negative interaction (degradation).

2.2 Model dynamics and properties

From the definition of LRG, we define the model state as a vector $s = (s_1, \ldots, s_n)$, where s_i denotes the activity level of the corresponding node.

At a given state, the rules associated with each node define its target level. When the current level of a node is different from its target level, it is called to update towards this target level, resulting in a transition to another state, *i.e.*, when there is a change of the activity level of the model variables, transitions are created where the values of the variables s_i are increased or decreased.

The resulting state transitions define the State Transition Graph (STG), representing the dynamic behavior of the logical model. In this graph, nodes represent the states of the model variables, while directed arcs represent state transitions. Formally, the definition of the STG can be described as follows:

Given a LRG $R = (G, K)$, where G denotes the set of the components and K denotes the set of the logical regulatory functions, the corresponding STG $\mathcal{E} = (S, T)$ is a directed graph with:

- S the state space $R : S = \prod_{g_i \in G} D_i$, where $D_i = 0, \ldots, max_i$, which are the values that the level of the regulatory node g_i can assume.

- $T : S^2 \rightarrow \{0, 1\}$: the transition function. There is an arc connecting a state s to its successor s' whenever $T(s, s') = 1$. The transition function is defined according to an updating policy and the regulatory functions.

Here each state is represented by the vector s, which is composed by n values, where n is the number of the regulatory nodes, and its value is defined by the activity level of each regulatory component.

We further refine that, concerning the construction of the STG, several nodes can be called to update at a given state, thus, the modeler must specify how the update should be performed. The two most used strategies are the synchronous and asynchronous updates.

With the asynchronous update, each variable is modified independently, generating as many transitions as the number of updated variables, *i.e.*, if the current state involves k updating calls, it will thus have k successors. This updating scheme potentially generates non-deterministic dynamics. Given a model with n regulatory nodes, we denote $s(t)$ the state at iteration t and $s_i(t)$ the value of the activity level of the component g_i at iteration t. Its successor state $s(t + 1)$ can be defined according to the following equation:

$$\begin{aligned} s_i(t+1) &= s_i(t) + sign(K_i(s(t)) - s_i(t)), \\ s_j(t+1) &= s_j(t) \text{ for all } j \neq i. \end{aligned} \tag{2.1}$$

Note that a stable state s can be considered as its own successor.

The full STG generated from the LRG of the model of the mammalian p53-Mdm2 network using the asynchronous updating scheme can be visualized in Figure 2.3.

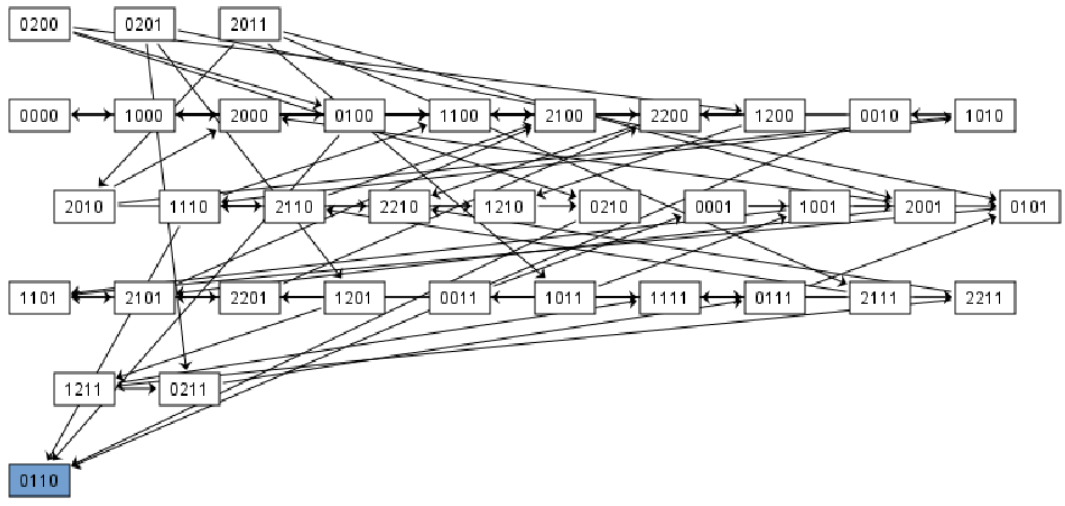


Figure 2.3: STG of p53-Mdm2 model using asynchronous update scheme

In contrast, in the synchronous updating scheme, all concerned nodes perform the updating si-

multaneously, resulting in an unique transition toward a single successor state. Thus, it is defined by increasing or decreasing by 1 all the variables whose current values differ from the values specified by their logical functions. Hence, this updating generates deterministic dynamics and using the same formalism described in the asynchronous updating scheme, its successor state $s(t+1)$ can be defined according to the following equation:

$$s(t+1) = (s_i(t) + sign(K_i(s(t)) - s_i(t)))_{i=1,...n} \tag{2.2}$$

According to this equation, the successor state $s(t+1)$ can be obtained by increasing, decreasing or maintaining the value of of activity level of each component of the model taking into account their corresponding logical regulatory function. Note that $sign(p)$ is equal to 1 if $p > 0, -1$ if $p < 0$, and 0 otherwise, where p is the difference between the value of the target activity level and the value of current activity level of the corresponding variable.

The STG of the p53-Mdm2 model generated using synchronous updating scheme is presented in Figure 2.4. Note that, compared to the STG generated in the asynchronous updating scheme in Figure 2.3, the graph is now divided in three disconnected sub-graphs. And some trajectories are grouped in a single transition.

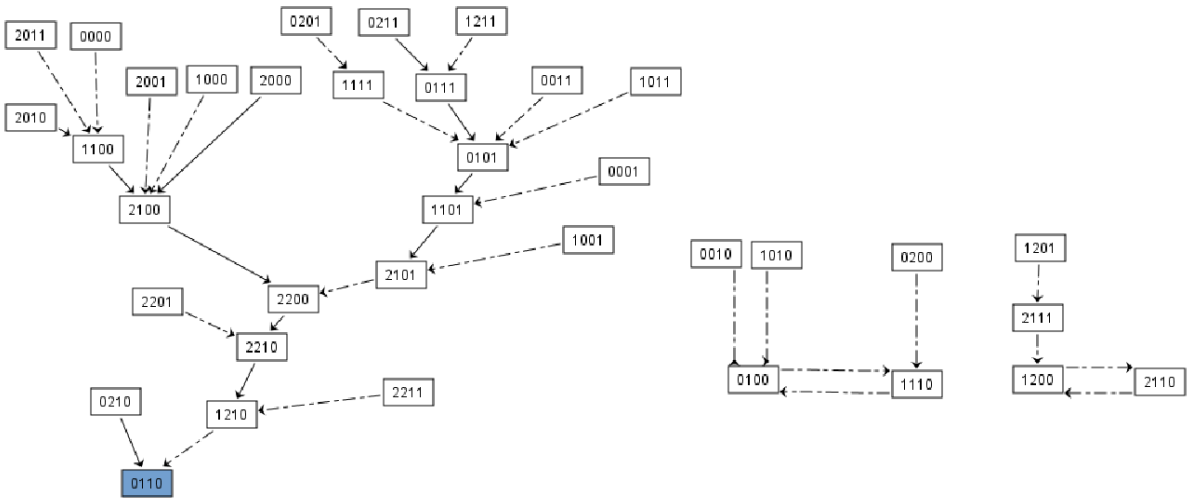


Figure 2.4: STG of p53-Mdm2 model using synchronous update scheme

Since the state space of graph is finite, there is always at least one state without successors or a cyclic trajectory denoted as terminal Strongly Connected Components (SCC), *i.e.*, a sub-graph with a maximum number of states that are mutually reachable and no leaving transition.

For the dynamical analysis of the biological network, we are particularly interested in the sub-graphs of the STG as they reveal properties relevant for the identification of such asymptotic behaviors [9]. To

formally define an attractor, we introduce the following notations [10]:

- Scc is the set of all SCCs of the $\mathcal{E}$.
- $\mathscr{S}$ is the subset of Scc and represent the set of trivial SCCs ($\forall C \in \mathscr{S}, |C| = 1$).
- $\mathscr{C}$ represents the set of complex SCC: ($\forall C \in \mathscr{C}, |C| >= 2$).
- $\forall s, s' \in S$, $s \rightsquigarrow s'$ means that for every state s and s' of S, if $s \rightsquigarrow s'$, then exist a path from s to s'.
- $\forall s \in S$, $\forall C \in Scc$, $s \rightsquigarrow C$ means there exists a path from every state s of S to any state s' of C.
- * denotes terminal elements of Scc. The non-terminal components are transient, *i.e.*, with leaving transitions.
- C^* is an attractor if $\forall s \in C^*$, $\forall s' \in S$, $s \rightsquigarrow s' \rightarrow s' \in C^*$, *i.e.*, for every state s of SCC C, if exist a path from s to any state s' of S, then s' belongs to C.

Note that $\mathscr{C}^*$ denotes the set of complex attractors and $\mathscr{S}^*$ denotes the set of stable state.

Figure 2.5 presents the STG of the toy model (in an asynchronous dynamics) when grouping states in SCCs. And we can observe that:

- States without color are transient trivial SCCs that lead to other transient SCC or attractors;
- States in green form a transient complex SCC;
- States in light red form a terminal complex SCC (complex attractor);
- State in dark red is a terminal trivial SCC (stable state).

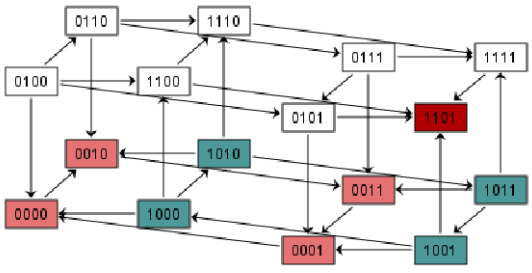


Figure 2.5: Asynchronous STG of toy model.

From the biological point of view, the asynchronous scheme is more realistic than the synchronous scheme as it accounts for different delays associated to the updates, generating all potential trajectories.

It is also important to mention that different updating schemes may lead to different dynamics. Indeed, while the stable states are preserved, the STG of the p53-Mdm2 model generated from the synchronous updating presented in Figure 2.4, generates two more cyclic attractors, which can be reached only from two or three states.

Furthermore, asynchronous scheme generates as many successors as updating calls, thus some of these trajectories are potentially unfeasible. In case of concurrent events, some choices can be prioritized as long as the choice is well-grounded on the nature of the processes involved [11].

Indeed, in a biological context, defining the *priority classes* can be seen as a refinement of the updating scheme by defining the different time scales associated with biochemical processes. Taking into account additional knowledge, we can classify component updates into slow or fast processes. Hence, when we considering an asynchronous scheme refined with priority class, some transient SCCs with oscillatory behaviors may turn into cyclic attractors.

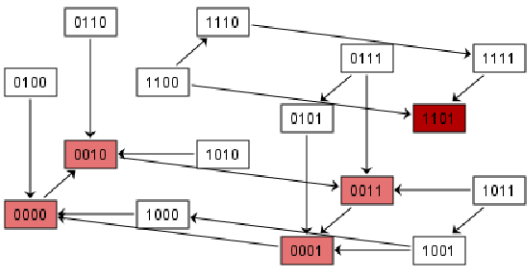


Figure 2.6: Asynchronous STG of toy model with priority classes.

Figure 2.6 shows the STG generated by the LRG of the toy model, when considering that the degradation of regulatory factors is faster than the synthesis.

As a result, transitions involving increasing component levels (i.e., corresponding to synthesis) are being eliminated when they are in concurrence with transitions decreasing some component levels. Note that, trajectories that are considered unfeasible as discarded, which may lead to the loss of some transient SCCs, to new (complex) attractors, and to modified reachability properties.

2.3 Model analysis

Attractors are always present in the model whether to be a stable state or a cyclic attractor. Note that the identification of attractors is especially important for the analysis of biological model. Indeed, model attractor are relevant to understand biological processes in the system. Cyclic attractors are associated with oscillatory behaviors that are observed in the case of the cell cycle or circadian rhythms [11]. Stable states, instead, are usually associated with cellular responses to external factors or perturbation or cell

lineages [12].

However, the identification of such property brings hard challenges. As the number of component of the model increases, the size of the corresponding state space (and hence STG) grows exponentially.

Indeed, in small models (up to dozen components), it is relatively easy to recover attractors by analyzing directly the STG constructed. However, for larger models, such an approach to identify the attractors becomes unrealistic.

The identification of the stable states is relatively easier, as it can be identified by reaching to a state without successors (except itself). By analyzing the logical regulatory function of each component and represent it by a decision diagram, it is possible to identify all the stable states of a model up to hundred components [7].

The identification of the complex attractors is harder, as they might differentiate from the selection of different updating scheme. They are cycles composed by p states (without knowing p in advance) Thus, it is hard to detect without exploring the whole STG.

Moreover, one can compact the full STG by grouping states that belong to the same complex SCC into a single node. The resulting graph is denoted as Hierarchical Transition Graph (HTG) [10] and it eases the analysis of the dynamical properties. In specific, it reveals properties related with the identification of attractors and the basins of attraction that is described in Subsection 3.1.2.

A method based on model checking applied in biological systems [13] allows to verify the reachability of attractors. The model checker takes a set of properties expressed as temporal logic formulas and checks whether they are satisfied by the model.

The estimation of reachability of attractors is even harder. This is because, apart from the difficulties for the identification of the attractors mentioned above, we are performing simulations with several runs in order to assess the transition probability to each attractor. In practice, we are exploring the whole STG and calculating the percentage of runs which reach to a determinate attractor, and as the size of the STG grows exponentially, the process becomes computationally intensive.

The algorithms Firefront and Avatar [1] have been proposed to quantify the attractor reachability. Both approaches allow to estimate the distribution of the reachability of attractors under the assumption of equiprobability of concurrent transitions, differing in the performance (the second one performs well with STG with large transient SCCs) and the requirements (the first one requires an initial condition). On the other hand, MaBoSS [14] is proposed to estimate the state probabilities over time and compute the stationary distribution of the resulting Markov process of the model.

2.4 Markov process

A Markov process is a stochastic process that satisfies the *Markov property* which is characterized as memory-less. Indeed, in a Markov process, the probability of transitioning to any particular state only depends on the current state, *i.e.*, predictions can be made regardless the history of the process or the future states[1]. A Markov process can be characterized as time continuous or time discrete. For this work, we focused on Markov processes with discrete time and state space, denoted as *Markov chain*.

To describe a Markov chain, let us assume wthat e have a set of states $\sum = \{s_1, s_2, \ldots, s_n\}$. Given a stochastic process, it starts in one of these states and transitions from one state to another. Each transition is denoted as $step$. The probability of a state s_i transiting to another state s_j is denoted by $P_{i,j}$ and is called *transition probability*. Such probability does not depend on which of the previous states are visited by the process before the current state. This property is denoted *Markov Property* and can be described as follows [15].

Let X be a Markov chain, and let $X_t = s$ denote that the chain is in the state s at time t. The probability of transiting from the state s_i at time t to a successor state s_j is:

$$P(X_{t+1} = s_j | X_t = s_i) = P(X_{t+1} = s_j | X_0 = s_0, X_1 = s_1, \ldots, X_t = s_i) \tag{2.3}$$

The *transition matrix* P of a Markov chain X is a matrix providing the probability of transitioning between states. In particular, given an ordering of the matrix's rows and columns by the state space $\sum$, the $(i, j)^{th}$ element of the matrix P is given by

$$P_{i,j} = P(X_{t+1} = s_j | X_t = s_i) \tag{2.4}$$

P as defined by Equation 2.4 is a probability matrix: for all i, the i^{th} row gives the probabilities to reach each state of the state space from state s_i, the sum of these probabilities being 1. Furthermore, by calculating the k-step transition matrix of the chain, we can obtain the transition probabilities of each state after k steps, calculated according to the following equation:

$$P^k = \Pi_{i=1\ldots k} P \tag{2.5}$$

By assigning the initial probabilities of the chain starting in each of the states, we define u to be the probability vector which represents the starting distribution and then the probability that the chain is in state s_i after n steps is the ith entry in the vector:

[1] https://en.wikipedia.org/wiki/Markov_chain

$$u^n = uP^n \tag{2.6}$$

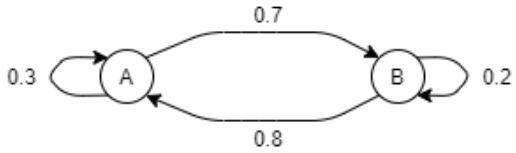


Figure 2.7: Example of a Markov chain

Figure 2.7 provides an example of a Markov chain. According to the probabilities assigned to the arcs between the nodes, the transition matrix of the Markov chain built on states A and B is $P = \begin{pmatrix} 0.3 & 0.7 \\ 0.8 & 0.2 \end{pmatrix}$. To obtain the transition probabilities of each state after 2 steps, we calculate the 2-step transition matrix: $P^2 = \begin{pmatrix} 0.65 & 0.35 \\ 0.4 & 0.6 \end{pmatrix}$.

We further introduce the notion of an *Absorbing Markov chain*, which is a special type of Markov chain. A state s_i of a Markov chain is called absorbing if it has no successor but it self ($p_{i,i} = 1$). A Markov chain is absorbing if it has at least one absorbing state and if, from every state, an absorbing state is reached. A state that is not absorbing is called *transient*.

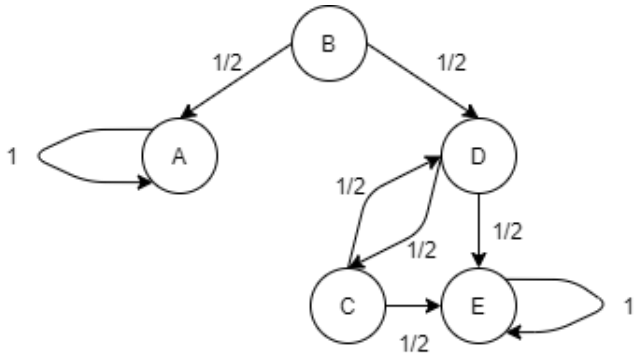


Figure 2.8: Absorbing Markov chain

Figure 2.8 presents an example of an absorbing Markov chain, which has two absorbing states (A and E), where from any transient state, the probability of hitting an absorbing state as time progresses is almost 1.

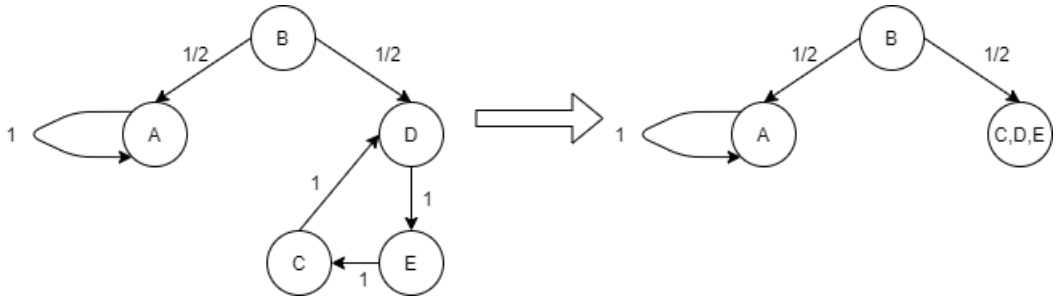


Figure 2.9: Absorbing Markov chain by considering complex attractor as an absorbing class

2.5 STG seen as Absorbing Markov Chains

The incidence matrix of a STG can naturally be translated into an $|S|$ x $|S|$ - transition matrix Π which is defined as follows [1] :

$$\begin{aligned} \forall s, s' \in S\ \Pi(s, s') > 0 \Leftrightarrow (s, s') \in T, \\ \forall s \in S\ \Pi(s, s') = 1 \Leftrightarrow Succ(s) = \emptyset, \\ \Pi(s, s') = 0 \text{ otherwise} \end{aligned} \tag{2.7}$$

In a first stage, we can assume that the probabilities of concurrent transitions are uniformly distributed:

$$\forall s \in S, \forall s' \in Succ(s), \Pi(s, s') = \frac{1}{|Succ(s)|} \tag{2.8}$$

Therefore, our Markov chain (s_0, Π) is defined by the set of states and the transition matrix previously defined. For the chain to stop whenever it reaches an attractor, we considered the quotient graph of the STG, *i.e.*, the graph partitioned into blocks according to the SCC. Each complex attractor of the STG is thus interpreted as a single node and, by considering this new STG, each terminal node corresponds to an absorbing state of the corresponding chain. We denote it, in our context, as an absorbing class of the chain. Note that the number of absorbing classes of the Markov chain corresponds to the number of attractors of our model.

For instance, the left Markov chain of Figure 2.9 is not an absorbing Markov chain as it has a terminal cycle composed by nodes C, D, E. But we can group them into a single node and denote it as the absorbing class of the chain. In this case, we have two absorbing classes that are node A and node (C, D, E).

We thus define the Markov chain X on the set $S = T \cup A$, where $T \subset S$ is the set of all the transient states, and A is the set of absorbing classes. The transition matrix π of X is [1]:

- $\pi(a_i, u) = 0\ \forall u \in T, \forall a_i \in A$

- $\pi(u, a_i) = 1 \; \forall a_i \in A$
- $\pi(a_i, a_j) = 0 \; \forall a_i \in A, \forall a_j \in A, i \neq j$
- $\pi(u, v) = \Pi(u, v) \; \forall u, v \in T$;
- $\pi(u, a_i) = \sum_{v \in a_i} \Pi(u, v) \; \forall u \in T, \forall a_i \in A$.

If there are r absorbing classes and t transient states, one can rearrange the matrix π by considering first the transient states and then the absorbing classes, forming the canonical form [16] of the transition matrix:

$$\pi = \begin{pmatrix} Q & R \\ \mathscr{O} & I \end{pmatrix} \tag{2.9}$$

where Q is a *t-by-t* matrix with $Q(u, v) = \pi(u, v)$ for $u, v \in T$, R is a *t-by-r* matrix with $R(u, a) = \pi(u, a)$ for $u \in T$ and $a \in A$, $\mathscr{O}$ is an *r-by-t* null matrix, and I is an *r-by-r* identity matrix.

By introducing the time step in the transition matrix, we denote $\pi^k(u, v)$ the probability that, started in state u, the chain is in state v after k steps and it can be defined as:

$$\pi^k = \begin{pmatrix} Q^k & (\sum_{j=0}^{k-1} Q^j)R \\ \mathscr{O} & I \end{pmatrix} \tag{2.10}$$

From this, the following properties hold [16]:

- Q^k tends to 0 when k tends to infinity ($\lim_{n \to \infty} Q^k = 0$), which means the process will eventually escape from transient states;
- for an absorbing Markov chain X, the matrix $N = (I - Q)^{-1}$ is called the *fundamental matrix*. The entry $N_{i,j}$ of N gives the expected number of times that the process is in the transient state s_j if it is started in the transient state s_i;
- the expected number of steps for absorption of a chain starting in state s_i is given by: $t = Nc$ where c is a column vector with all entries equal to 1;
- the *absorption probabilities* are given by: $B = NR$ where N is the fundamental matrix and R is the sub-matrix of the canonical form of the transition matrix.

By substituting variables, from any $u \in T$, the probability of X being absorbed in $a \in A$ is $P(X_\infty = a | X_0 = u) = (I - Q)^{-1} R(u, a)$.

3

Related Work

Contents

Several research lines have been proposed to analyse the dynamics of logical models of biological systems.

For the identification of attractors, in 2013 D. Bérenguier *et al.* proposed a method based on the Tarjan algorithm [17] to cope with large dynamics [10]. The method consists in the construction of a *Hierarchical Transition Graph* (HTG), a compact representation of the STG, which eases the identification of complex attractors and their *basins of attraction*. In 2007, an approach based on the construction of a representation of the regulatory functions in terms of Multi-valued Decision Diagram (MDD) was proposed by A. Naldi *et al.* [7]. This representation allows the identification of stable states of models with up to hundred components [7].

When it comes to attractor reachability, approaches based on a Boolean Satisfiability Problem (SAT)-algorithm was proposed by Dubrova *et al.* [18]. In 2015, Abou-Jaoudé *et al.* published a method based on Model Checking to assess reachability properties [13].

To estimate the reachability of attractors, Stoll *et al.* developed a C++ software (MaBoSS) that computes the temporal evolution of probability distributions of states of the STG [14]. The algorithms Avatar and Firefront were proposed by Mendes *et al.*. These algorithms, apart from being able to identify the attractors (stable states or complex attractors), were designed to quantify the reachability probability of the attractors [1].

3.1 Attractor identification

The identification of attractors is not an easy challenge. In particular, complex attractors which, apart from being dependent from the updating scheme, are hard to be identified. In this section, we present and discuss existing methods to identify attractors of logical models.

3.1.1 Classical algorithm to identify Strongly Connected Components

In 1972, R. Tarjan published an algorithm [17] for finding the Strongly Connected Components (SCC) of a directed graph, which performs a single pass of a Depth First Search (DFS).

To better understand the algorithm, it is essential to understand the concept of low-link value. The low-link value of a node is the smallest node id reachable from that node when performing a DFS search.

In the Tarjan algorithm, we have:

- an array of low-link of each node;
- an array of id of each node;
- a stack to store visited nodes;

The Tarjan procedure can be described as follows:

1. Mark each vertices as unvisited;
2. Start the DFS at a random node. Upon visiting a node assign it an id and a low-link value. Mark explored nodes as visited and add it to the stack;
3. When reaching an already visited node with all its successors visited, the DFS backtracking is performed:
    - the low-link value of the current node is updated to the minimum between the low-link value of previous node and the low-link value of the current node;
    - if the current node id is equal to its low link value, pop off all the nodes until reaching the current node (the nodes popped form a SCC);
4. repeat the process until all the nodes are being visited.

The pseudo-code of the algorithm is given in Figure 3.1.

Time Complexity: since the procedure is called only once per node, the running time of the algorithm is linear in the number of vertices and edges, *i.e.*, $O(|V| + |E|)$.

Space Complexity: this algorithm requires two supplementary arrays to store the index and low link value of each node, and a stack for storing visited nodes. So the worst-case size of the stack is $|V|$ (when the graph is a single giant SCC). Therefore the space complexity is $O(|V|)$.

The identification of SCC is relevant for the purpose of this work. Apart from easing the detection of terminal SCC which correspond, in our context, to model attractors, it is possible to apply the Tarjan procedure to group SCC and shrink them into a single node, obtaining the SCC graph that is a compacted representation of the STG.

```
procedure tarjanAlgorithm(Node node, List scc, Stack s)
v.index = index
v.lowlink = index
index++
s.push(node)
for each Node n in Adj(node) do
        if n.index == -1
                tarjanAlgorithm(n, scc, s, index)
                node.lowlink = min(node.lowlink, n.lowlink)
        else if stack.contains(n)
                node.lowlink = min(node.lowlink, n.index)

if node.lowlink == node.index
        Node n = null
        List component
        do
                n = stack.pop()
                component.add(n)
        while(n != v)
        scc.add(component)
```

Figure 3.1: R.Tarjan Algorithm

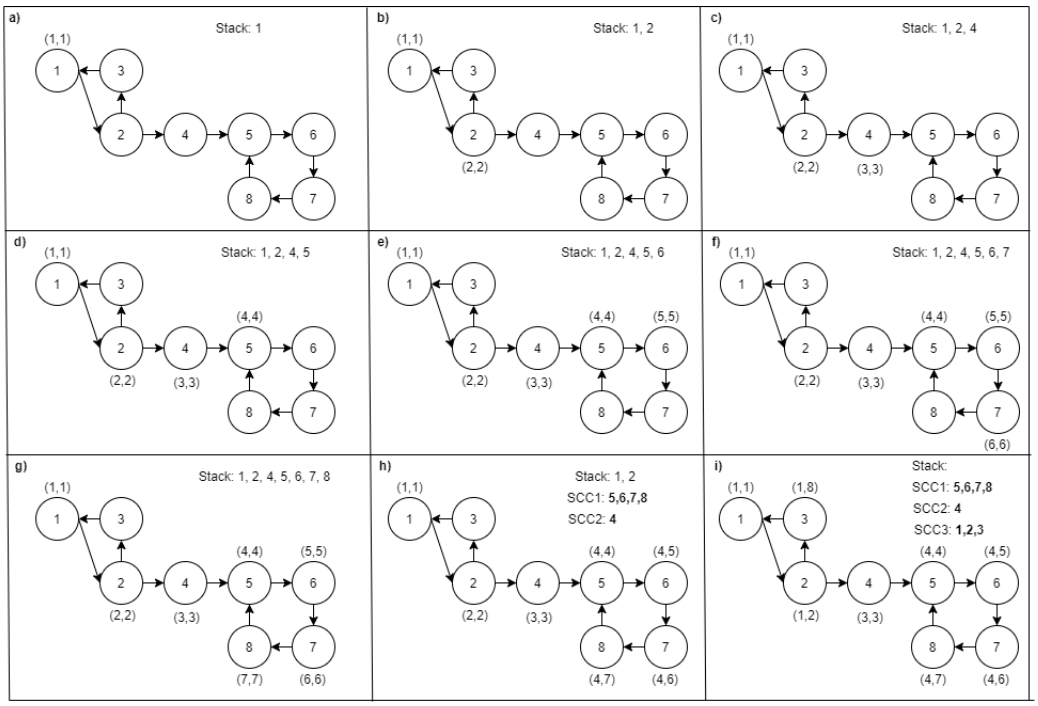


Figure 3.2: Tarjan procedure on a graph where each node is labelled with the couple (id, low-link), and the state of the stack is provided along the procedure

Figure 3.2 illustrates the Tarjan procedure on a graph. We start the DFS at node 1. During the exploration nodes 2, 4, 5, 6, 7 and 8 are visited, and we assign the corresponding low link values and id values in steps *a)* to *g)*. At step *h)*, the DFS backtracking is performed, and we update the low link values of the nodes 6, 7, 8. As a consequence, nodes are popped from the stack until reaching the node whose id is equal to its low link value, forming the SCCs (SCC1: 5, 6, 7, 8 and SCC2: 4). At step *i)*, by proceeding the DFS backtracking, we visit the node 3 and following the exact same procedure described

above, we obtain SCC3: 1,2,3.

3.1.2 Hierarchical Transition Graph

Here a strategy to ease the analysis of large dynamics is briefly presented [10]: the construction of *Hierarchical Transition Graph* (HTG), which can be interpreted as a further reduction of the STG by grouping sets of states that form SCC or that are composed by a chain of trivial SCCs leading to the same set of SCCs and attractors.

To define a HTG, let us recapitulate the types of SCCs that exist in a STG:

- $\mathscr{C}$ denotes the set of complex SCCs;
- $\mathscr{S}$ denotes the set of trivial SCCs;
- The $\mathscr{S}^*$ and $\mathscr{C}^*$ are terminal SCCs and denote, respectively, stable states and complex attractors.

We introduce the application σ that associates to an SCC C, the set of complex or terminal SCC, *i.e.*, $\mathscr{C} U \mathscr{S}^*$, that are reachable from X, including C itself if it is complex or terminal [10].

$$\sigma(X) = \{C \in \mathscr{C} \cup \mathscr{S}^* s.t\ X = C\ or\ \forall s \in X, s \rightsquigarrow C\} \tag{3.1}$$

Furthermore, $\mathscr{P} \subset 2^S$ is defined as the set of irreversible transient components where trivial non-terminal SCCs are grouped together if they share the same σ-image [10],

$$\mathscr{P} = \{I \in 2^S\ s.t.\ \forall s \in I, \{s\} \in \mathscr{S} \setminus \mathscr{S}^*\ and\ s, s' \in I \Rightarrow \sigma(\{s\}) = \sigma(\{s'\})\} \tag{3.2}$$

We can interpret the HTG associated to an STG as a tree whose nodes includes complex SCCs, stable states and sets of linear chains of trivial non-terminal SCCs. Therefore the definition of the HTG can be described as: $\mathscr{H} = (\mathscr{C} \cup \mathscr{P} \cup \mathscr{S}*, \mathscr{T})$, where $\mathscr{T}$ defines the arcs of the $\mathscr{H}$ [10]

$$\mathscr{T}(C, C') = 1 \Leftrightarrow \exists s' \in C, \exists s' \in C'\ s.t\ T(s, s') = 1 \tag{3.3}$$

which means there is an arc from C to C' if and only if there is a transition from a state s in C to a state s' in C'.

Note that from this definition, we can determine two properties:

- A path connecting any HTG component to a non-irreversible component implies the existence of a path in the corresponding STG;

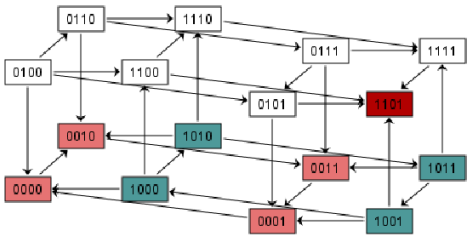


Figure 2.5: asynchronous STG of toy model (repeated from page 10)

- A path between two states in the STG implies the existence of a path between the HTG components they belong to.

These properties are proved in [10].

Given the STG generated from the toy model using the asynchronous scheme presented in Figure 2.5 and its generated HTG is illustrated in Figure 3.3.

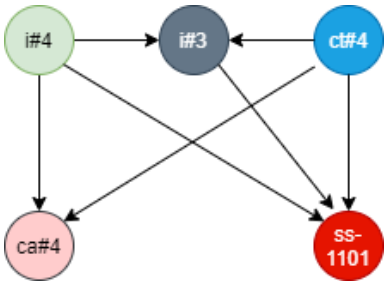


Figure 3.3: HTG of the toy model

We can observe that:

- the red node represents the stable state;
- the pink node represents the complex attractor;
- the blue node represents a transient complex SCC;
- the green and grey nodes represent the groups of trivial SCCs that share the same σ-image σ, as they are able to reach to the same set of SCCs.

The algorithm to generate HTGs, given a set of condition can be found in the supplementary file of the article [10].

The HTG is especially relevant for the dynamical analysis of logical models, easing the visualization of attractors and their basins of attraction.

Indeed, given A^*, an attractor, and C a node of the HTG, the states in C are in B_{A^*}, the *basin of attraction* of A^*, if and only if $A^* \in \sigma(C)$, *i.e.,* A^* belongs to the set of complex or terminal SCCs reachable from states in C. The states in C are in $\overline{B}_{A^*}$, the *strict basin of attraction* of A^*, if and only if $\sigma(C) \cap (\mathscr{C} \cup \mathscr{S}) = \{A^*\}$, *i.e.,* from the states of C the only reachable attractor is A^* [10].

3.1.3 Stable state identification

Here we present methods to identify stable states of logical models. In particular, we describe the use of *Decision Diagrams* (binary or multi-valued) [19] [7].

In a LRG, the logical function K_i associated to a node g_i specifies the evolution of s_i, the activity level of that node. It can take a finite number of values (in $\{0, \dots Max_i\}$), depending on the values of the regulators of g_i (the nodes targeting g_i in the regulatory graph). The representation of the function K_i in terms of a decision diagram is presented below, where the decision variables are the activity levels of the regulators.

A Boolean Decision Diagram (BDD) is a directed acyclic graph composed by decision nodes and terminal nodes. Each decision node has two outgoing edges representing the values 0 and 1, and terminal nodes are labelled with the function Boolean values. Along a path, the child chosen for each non-terminal node is labelled with the value of the corresponding edge. The path from the root node to the 1-terminal node (resp. 0-terminal node) represents a variable assignment for which the Boolean function is true (resp. false). If the function is multi-valued, *Multi-valued Decision Diagrams* (MDD) are used, where decision nodes have as many outgoing edges as the number of possible values, and there are as many terminal nodes as the number of the function values.

One can reduce the sizes of such diagrams by merging isomorphic sub-diagrams which are shared by multiple nodes and bypassing nodes whose children are the roots of isomorphic sub-graphs. The resulting reduced graph is called *Reduced Ordered MDDs* [7].

The representation of the functions K_is by means of MDDs greatly facilitates the analysis of specific dynamical properties [7]. In particular, it allows to identify all the stable states of the model without having to explore the whole STG. As described in Section 2.2, a stable state is such that the current activity of each variable is equal to its target level. In other words, a stable state is a fixed point of the function $(K_i)_{i=1,\dots n}$ defined over the state space. Given a LRG $R = (G, K)$ with $G = \{g_1, \dots, g_n\}$, a state is stable iff:

$$\forall i \in \{1, \dots, n\}, K_i(s) = s_i \tag{3.4}$$

To identify such stable states, we can build the logical stability decision diagram for each gene that represents the conditions for the gene value to be stable. This is done by transforming the MDD rep-

resenting the logical function. For any gene g_i, the decision variable s_i is added in the MDD, and the terminal nodes are set to 0 for a change (a decrease or an increase, if the gene is not in a stable value), or to 1 for no change. The gene g_i is stable when the value of its resulting diagram S_i is 1 (true).

The diagrams constructed thus represent the logical conditions for the stability of the genes. To get the stability condition of the whole, we can perform the product of these diagrams to get a single diagram $S_{i...n}$ that represent the conjunction (AND) of the individual stability conditions. The rules to combine the diagrams are as follows [7]:

- decision variables are ordered with the same order, for all diagrams;
- if S_1 and S_2 are reduced to single nodes, the product is a node, its value being the product of those of S_1 and S_2;
- else if S_1 is a single node with value 1 , the result is S_2;
- else if S_1 and S_2 roots are sub-graphs of nodes with the same order, the result is a root of this order, and its sub-graphs are recursive combination of sub-graphs rooted in its children;
- otherwise, if the root of S_1 has an order less than that of the root of S_2 (or symmetrically), the result is such that:
  - the root is that of S_1;
  - its children are recursive combinations of S_2 with those of S_1.

Finally, the paths in the resulting diagram that lead to 1-terminal nodes give the stable states.

Figure 3.4 presents an example of identification of stable states using the BDD approach. We start by constructing the BDD representation of the regulatory function of each node. Next, the logical stability diagram of each node is derived from the function. Finally, we combine all the decision diagrams into a single one according to the rules described above. The stable states are defined by the paths of the resulting diagram that lead to leaves value 1, which are 010, 101, 110.

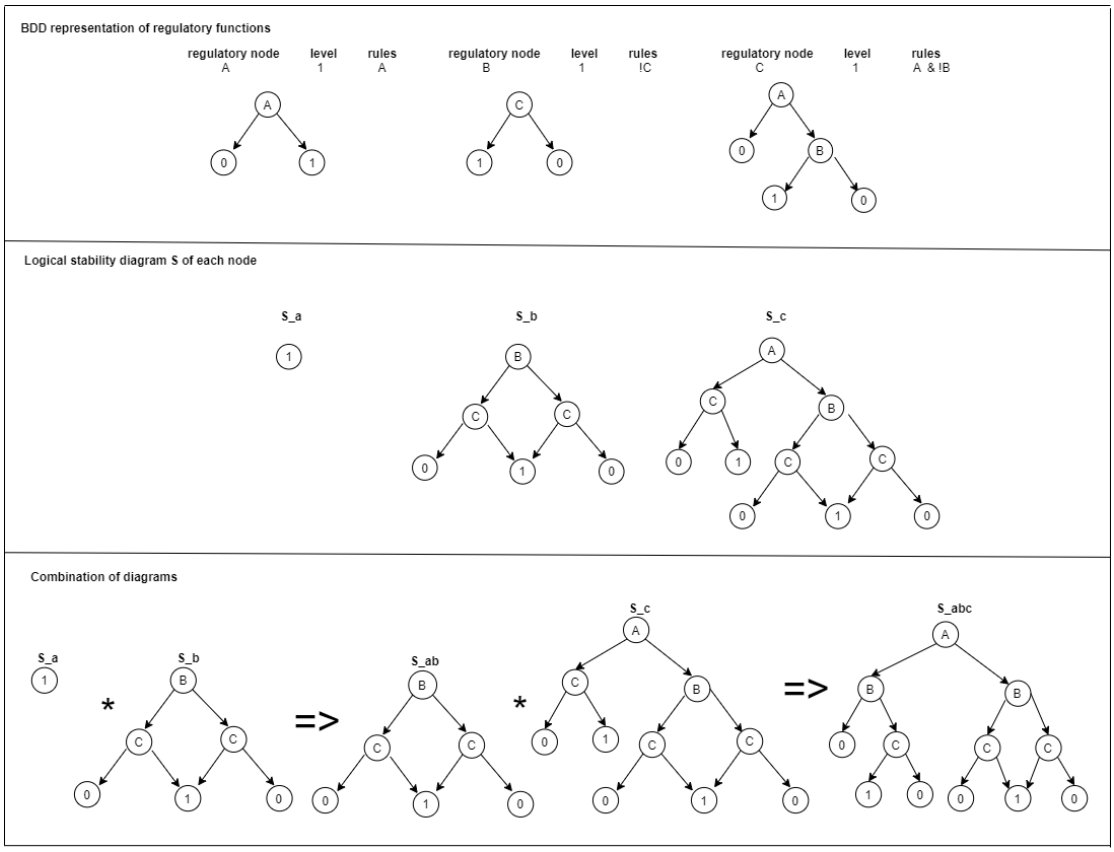


Figure 3.4: Stable state identification using Decision Diagrams

3.2 Attractor reachability

In this part of the document, approaches to identify attractors along with reachability properties, *e.g.*, to determine if an attractors is reachable from a specific initial state, are introduced. In particular, the two main approaches that will be described in this document are: SAT-based approach and Model checking.

3.2.1 SAT-based algorithm to find attractors

Dubrova and Teslenko [18] proposed an approach to identify attractors in logical models under the synchronous updating scheme. Hence, in this context, the model dynamics is deterministic and finite, where all the variables of the model are updated simultaneously whenever their target level is different from their current level.

The Boolean Satisfiability Problem (SAT) is the problem of determining whether a Boolean formula can be satisfied by an assignment of the problem variables.

Given a model and the STG constructed using the synchronous updating scheme together with an integer k, the algorithm can be described as follows:

1. Generate a vector of random initial states;
2. Choose the ith initial state in the vector and generate a trajectory of length k;
3. For the current trajectory, check whether it contains a loop or not. Since, in a synchronous logical model, each state has a single successor, we can assess the presence of a loop by checking if last state of the trajectory occurs twice.
4. If a loop is found, then mark all the states of the loop as they form an attractor, otherwise double k, continue the generation of the trajectory and proceed to step 3;
5. Repeat the steps 2, 3 and 4 until all the initial states are explored.

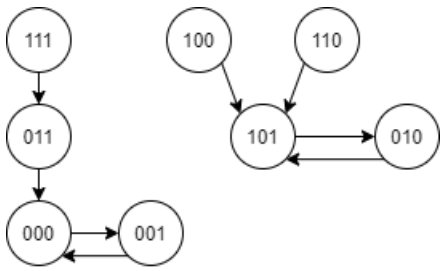


Figure 3.5: A synchronous STG composed by 8 states

Providing an example of the execution of the algorithm in the STG presented in Figure 3.5, The algorithm starts to search for a trajectory of length $k = 2$.

1. Suppose that the trajectory we found is $111 \rightarrow 011 \rightarrow 000$. Since the last state (000) occurs only once, this trajectory does not contain any attractor;

2. As a consequence, we double k's value to 4 and continue the search;

3. Suppose that, in the next iteration, the path found is $100 \rightarrow 101 \rightarrow 010 \rightarrow 101 \rightarrow 010$. In this case, we found two occurrences of state 010, thus (101, 010) is being marked as a two-state attractor;

4. In the following search, the trajectory found is $011 \rightarrow 000 \rightarrow 001 \rightarrow 000 \rightarrow 001$. Here state 001 occurs twice, thus we mark (000, 001) which is a two-state attractor;

5. In the final search, there are no more paths of length 4 that do not contain states of already identified attractor. Therefore, the algorithm terminates.

The method proposed in [18] applies such approach in the context of attractor reachability, in particular, it uses the SAT-solver for the identification of trajectories of a particular length k. For that purpose, a propositional formula F is introduced and a satisfying assignment to this propositional formula corresponds to a valid trajectory in the STG.

The generation of such a formula is obtained by *unfolding* the *transition relation*, where T(s,s') is true if there is a transition in the STG between the states s and s'. The process of unfolding can be interpreted as the recurrent application of the transition relation:

$$T_{p...r} = \bigwedge_{i=p}^{r-1} T(s_i, s_{i+1}) \tag{3.5}$$

where $T_{p...r}$ denotes the transition relation T unfolded from the time step p to the time step r.

Once obtained the formula F, a SAT-solver is called to check the satisfiability of this expression and to find a satisfying assignment. If the expression is unsatisfiable, then there is no trajectories of length k in the STG (this implies that all attractors have been already identified) or the trajectory does not contain a loop (then we double k and proceed with the generation of the new trajectory). On the other hand, if the SAT-solver finds a satisfying assignment, the algorithm checks whether there is a loop in the trajectory corresponding to this assignment.

3.2.2 Model Checking for reachability analysis

Here another approach to verify reachability properties of a logical model is presented. This method involves the use of model checking, which checks whether the model satisfies some properties.

The dynamics of a logical model can be represented by a graph-based *Kripke structure*, which contains all the information regarding the variables of the model and the transitions between the states [20]. To perform a verification, a model checker receives as input such a structure and a set of properties specified as temporal logic formulas. It verifies whether each of these properties is satisfied by the model under study, answering true or false to each of them. Temporal logic is a formalism used to specify the sequence of transitions between states, where the notion of time to make transition is not specified explicitly, only the order between them.

The verification of a model of a biological system aims to validate the existence of a trajectory complying a specific property (*e.g.,* reaching an given attractor) from a given initial condition [9, 13]. In the case of the asynchronous updating, one might be interested in studying each alternative path separately. This suggests the use of a temporal logic that provides path quantifiers where, at each step, a choice can be made between multiple paths, *i.e.*, a branching-time temporal logic. Within the family of branching-time temporal logics, Computation Tree Logic (CTL) is the most commonly used.

Different model checkers are available, differing in their characteristics. For the purpose of efficient application of this technique in the analysis of logical models, a symbolic model checker is used where the state space is generated during its exploration. In this context, *NuSMV-ARCTL* [21] is the most appropriate model checker which is the integration of NuSMV [22] (a symbolic model checker that uses MDDs) with the Action-Restricted CTL (extension of CTL with labels added to the transitions of the Kripke structure) [23].

Furthermore, PRISM model checker (https://www.prismmodelchecker.org/) is a probabilistic model checker which is useful for modeling and analysis of the system exhibiting stochastic behaviour [24]. It supports a CTL extension, performs symbolic model checking, which means state space is generated during the exploration, and it is especially useful in our work as the dynamic of our model (STG) can be translated to a discrete-time Markov chain [24].

To enable the use of model checking to analyse logical models, GINsim provides an export of models under the asynchronous updating scheme into NuSMV specifications [25, 26]. Biological observations and properties that we want to verify are then added to the NuSMV specification as temporal logic formulas.

3.3 Quantification of attractor reachability

We briefly present and discuss existing methods focused on the reachability quantification, which allows further analysis of the biological model and prediction of behaviors that are most likely to happen.

Algorithms enabling reachability quantification are presented, namely: the classical Monte Carlo simulation (for the convenience, we will refer it to be Monte Carlo) , Firefront and Avatar proposed by Mendes *et al.* [1], and MaBoSS presented by Stoll *et al.* [14].

3.3.1 Classical Monte Carlo Simulation

When the brute-force search in a network is not feasible, Monte Carlo is the most common approach to estimate the likelihood of the outcome [27]. It relies on repeating random sampling to obtain numerical results. This method may vary from context to context but tends to follow a particular pattern[1]:

1. Define a domain of possible inputs;
2. Generate inputs randomly from a probability distribution over the domain;
3. Perform a deterministic computation on the inputs;
4. Aggregate the results.

In our context, the algorithm performs a large number of simulations that halt whenever they reach a terminal (stable) state or when a maximal depth is reached. The advantage of this algorithm is that the simulation does not record past states, *i.e.*, it is only based on the present state and the transitions to its successors. Thus, the memory requirements is minimal. When the specified number of runs is achieved, we aggregate the results and estimate the reachability probabilities of the stable states.

However, this approach has 2 limitations:

- Terminal SCCs are not being detected, where it gets trapped and repeats the process endlessly.
- When the structure of the STG is composed by many transient SCCs, it might revisit states already visited an unrestricted number of times, turning it computationally intensive.

3.3.2 Firefront

The next algorithm presented is Firefront, proposed by Mendes *et al.* [1]. Although relatively simple, it is efficient when the size of transient SCCs is relatively small.

[1] Wikipedia: https://en.wikipedia.org/wiki/Monte_Carlo

This algorithm performs a breadth-first search from an initial state s_0 whose probability is assigned, initially, to be 1. Then it distributes and propagates the probability of such state to its successors.

Three sets F, N and A are used.

- F_k is the set of states being expanded at step k, it is called *firefront* as it corresponds to the front line of the breadth-first exploration: $F_k = \{s \in \sum, \exists s_o \rightsquigarrow s$ *of length* k$\}$ where $\sum$ is the set of states. For the convenience, we will refer *firefront* to be F, omitting k. During the expansion, it is calculated for each state s the probability of the Markov chain X to be in s after iterating k times from the initial state s_0, *i.e.,* $P(X_k = s|X_0 = s_0) = \pi^k(s_0, s)$.

- To tackle the efficiency bottleneck, N the set of neglected states is introduced that is used to avoid the exploration of unlikely trajectories. If the probability associated with a state $s \in F$ drops below a certain value α, then s is moved from F to N. The successors of a state in N are not explored. Nevertheless, if such a state is visited again as being the successor of a state of F (therefore updating its probability), and if its probability exceeds the threshold, it is moved back to F.

- If a state in F has no successors, it is moved to A, the set of stable states; if it is already in A, its probability increases according to this new trajectory.

Note that the sum of the probabilities of the states in F, N, and A is 1.

Firefront terminates whenever the total probability in F drops to zero or below some predefined threshold β or a predefined maximum depth is reached. The algorithm can be consulted in the article [1].

3.3.3 MaBoSS

MaBoSS is a software tool that assesses attractor reachability in asynchronous Boolean dynamics [14]. Note that MaBoSS cannot be used when considering multi-valued models (unless we booleanize the models).

It is an approach based on the continuous time Markov processes. To transform a discrete time Markov process described in Section 2.4 in a continuous time Markov process, transition probabilities should be replaced by transition rates $\rho(s \to s')$, where s and s' are the states of the system. These transition rates $\rho(s \to s')$ are non-zero only if s' is a successor of s. Hence, the Boolean function K_i can be replaced by the functions $R_i^{up/down}(s) \in [0, \infty]$ [14]. Given g_i (the node whose value differs from s and s') and s_i (the value of node g_i at state s):

$$\begin{aligned} \rho_{(s \to s')} &= R_{g_i}^{up}(s) \text{ if } s_i = 0 \\ \rho_{(s \to s')} &= R_{g_i}^{down}(s) \text{ if } s_i = 1 \end{aligned} \tag{3.6}$$

where R_i^{up} corresponds to the activation rate of node g_i, and R_i^{down} corresponds to the inactivation rate of node g_i.

MaBoSS uses a simulation algorithm which is based on the Kinetic Monte-Carlo algorithm (also named Gillespie algorithm) [28]. Given an initial state s, a maximum time, two uniform random numbers $u, u' \in [0, 1]$:

1. Compute total rate of transition leaving the state s: $\rho_{tot} = \sum_{s'} \rho_{(s \to s')}$
2. Compute transition time: $\delta t = -log(u)/\rho_{tot}$
3. Order the successor states $s'^{(j)}$ ($j = 1, ...p$, where p is the number of successors) depending on their respective transition rates $\rho^{(j)} = \rho_{(s \to s'^{(j)})}$.
4. Compute the successor state $s'^{(k)}$ such that $\sum_{j=0}^{k-1} \rho_j < (u' \rho_{tot}) \leq \sum_{j=0}^{k} \rho_j, (\text{with } \rho_0 = 0)$.
5. Repeat the steps 1, 2, 3 and 4 until the specified maximum time is reached.

As results, the algorithm produces a set of stochastic trajectories which can be used to compute probabilities [14].

MaBoSS installation instructions and indications to build and run a logical model can be obtained at https://maboss.curie.fr/. The output of the simulation is a set of data files with tables containing the temporal evolution of state probability distributions. Figure 3.6 presents the the plot obtained from the MaBoSS simulation on the p53-Mdm2 model. This model has a unique stable state where p53 = 0, Mdm2cyt = 1, Mdm2nuc = 1 and DNAdam = 0. Therefore, the probability of any state that has p53 component activated, drops along the time.

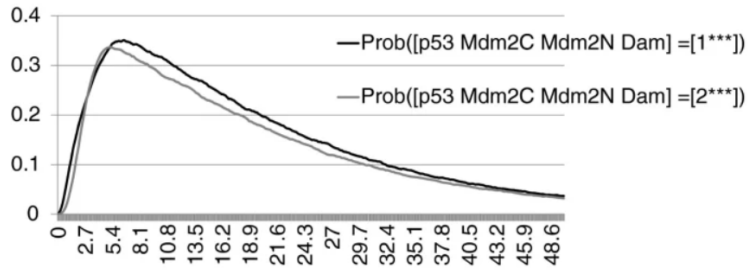


Figure 3.6: Plotting of the distribution of probability of states in p53-Mdm2 model

3.3.4 Avatar

Avatar is the last presented algorithm to quantify attractor reachability and it is the focus of this work. It is a modified Monte Carlo method that aims at efficiently coping with SCCs [1].

The main drawbacks of the classical Monte-Carlo approach are the following: the cycles are not being detected, therefore the simulation does not halt when entering in a complex attractor; also, when entering in a transient SCC, it may repeatedly revisit states already visited.

To solve such limitations, Avatar was designed as an appropriate modification of the classical algorithm: when entering a cycle, this is detected (when a state has already been visited), and the cycle is dismantled while appropriately recalculating probabilities. More precisely, upon visiting a cycle, the transitions between the states of the cycle are replaced by transitions towards each exit state of the cycle, *i.e.*, targets of the transitions leaving the cycle. These new transitions are associated with appropriate probabilities (calculated according to the properties of absorbing Markov chain). This process is called *rewiring* of the STG, and each rewiring creates a new so-called *Incarnation* of the dynamics. The simulation continues from the current state after the rewiring process ends [1].

Below, we describe the theoretical foundations of Avatar rewiring procedure and how the new transition matrix is calculated.

Suppose that we have a Markov chain X, defined by the finite set of states of a STG and its incidence matrix, an initial condition. And a random walk discover a cycle $C = (c_1, c_2, \dots c_k)$. We denote B the set of states directly reachable from C, *i.e.*, exiting states of the cycle.

We further define q the $k \times k$ sub-matrix of π (notation as defined in Section 2.5), for the states $c_1, \dots c_k$ of the cycle, and r the $k \times |B|$ sub-matrix of π, defining the transitions from C to B. The transition matrix of the rewired sub-graph of the STG is modified as follows:

- remove the transitions between the states of C and replace the sub-matrix q with the null matrix;
- add a transition from each state of C to each state of B and replace the sub-matrix r by $\left[(Id - q)^{-1} r\right]$ (the correctness of this process is described in Section 2.5).

Let Y denote this new chain, we can assert that starting from any transient state u and considering any attractor a, X and Y have the same asymptotic behaviors:

$$\forall u \in T, \forall a \in A, P_u(Y_\infty = a) = P_u(X_\infty = a) \tag{3.7}$$

This property is proved in [1].

In Figure 3.7, a illustration of the Avatar rewiring is provided. We can observe that the sub-matrix b is replaced by the null matrix and the sub-matrix r is recalculated according to the properties described above.

Avatar performs an extension step controlled by a parameter τ which is a modified Tarjan's algorithm for SCC identification [17], where more trajectories are being explored up to a depth of τ from states of the original cycle and all the subsequent rewiring is performed over the discovered cycle through the extension. Furthermore, the value of τ is doubled within each attempt to search further cycles in order to speed up the identification of SCC [1].

The current implementation of Avatar has two important parameters: expansion and rewiring limit. The first parameter limits the size of the cycle being detected, whereas the second parameter concerns the size of the cycle to be rewired. According to this, Avatar attempts to extend the cycle to be rewired when a cycle is found. However, if the extended cycle has more states than the specified rewiring Limit, only a sub-SCC of the detected cycle is rewired.

As a consequence, due to the bad time complexity ($O(n^3)$) for inverting a matrix, it is recommended to limit the rewiring limit of the sub-graph to be rewired when performing the *exact exit probabilities* strategy (the approach where the sub-matrix r is replaced by $\left[(Id - q)^{-1} r\right]$).

To tackle such a bottleneck, Avatar offers another strategy for rewiring, that is *uniform exit probabilities*. In this strategy, the transitions between the states of the discovered cycle are removed and, for each state of the cycle, the probabilities of the transitions from this state to the exiting states are distributed uniformly, according to the number of exiting states, *i.e.*, $1/|B|$, where B is the set of exiting states of the cycle. An illustration is presented in Figure 3.8;

The complete algorithm can be consulted in the supplementary material of the article [1].

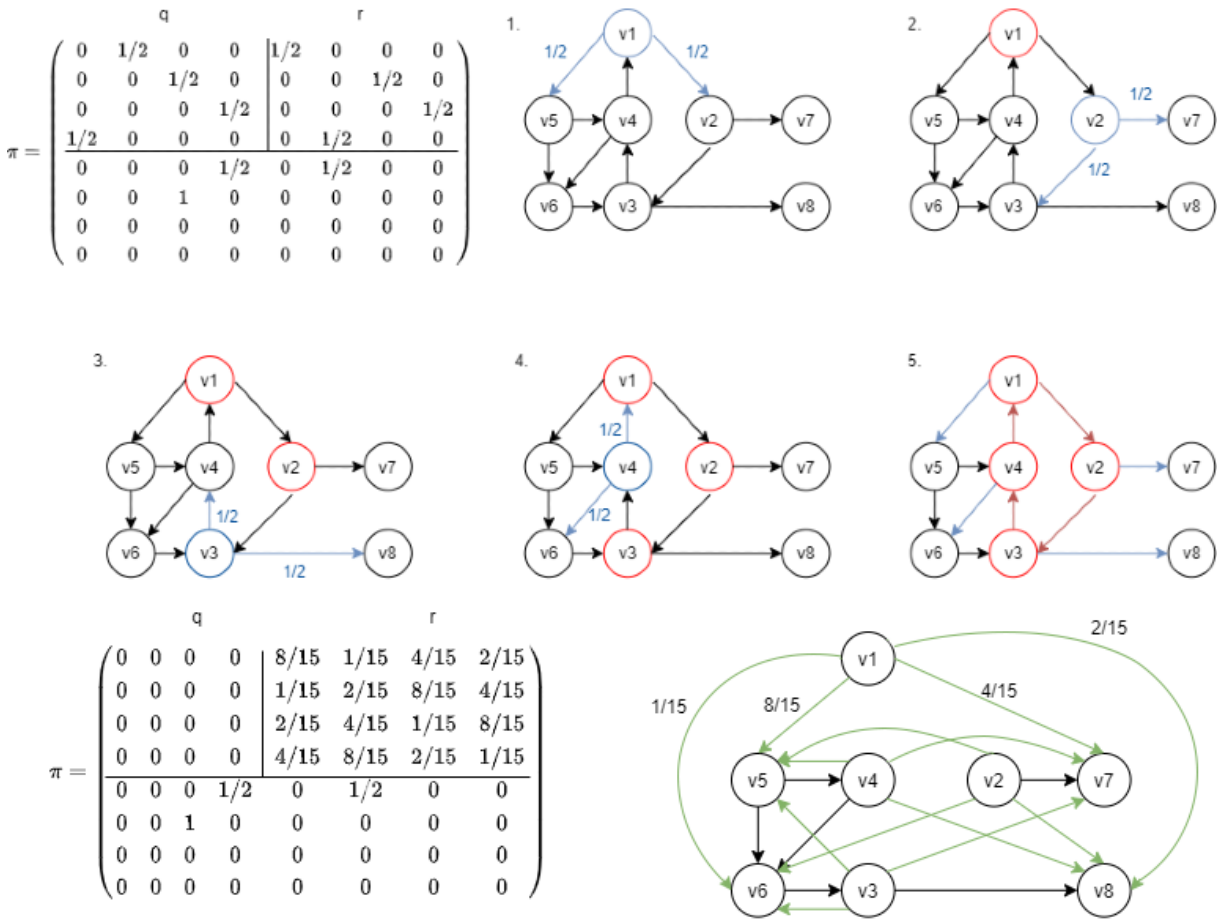


Figure 3.7: Random walk to discover a cycle that is getting rewired

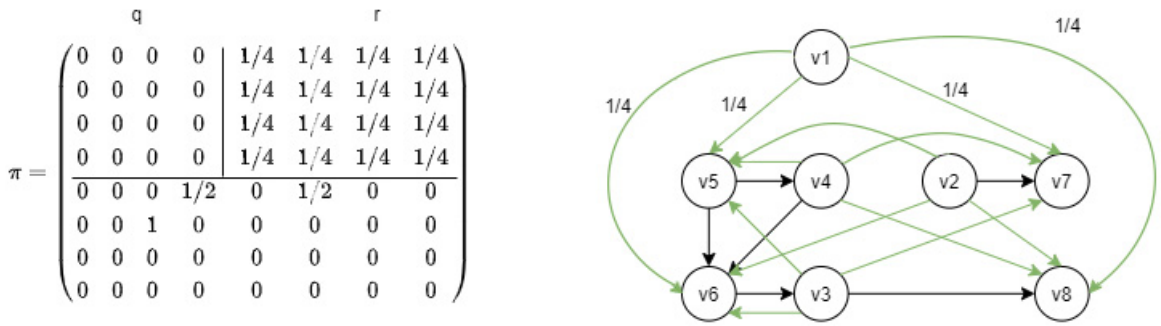


Figure 3.8: Avatar rewiring of the discovered cycle using random exit

4

Extension of the Quantification Reachability Algorithms

Contents

4.1 Software context

4.1.1 BioLQM

In 2018, A. Naldi presented bioLQM, a toolkit for the manipulation and analysis of logical models of biological regulatory networks. The source code of bioLQM is available at https://github.com/colomoto/bioLQM.

Unlike most other software tools that use their own file format to define logical model, bioLQM has a list of supported formats. BioLQM import/export capability is listed in [6]. Furthermore, it is recommended for users to use the SBML qual exchange format when using bioLQM. The format was proposed by Chaouiya *et al.*, 2013 to improve interoperability between modeling tools [29].

To generate the dynamics of a logical model, that is defined by the states and the transitions between them (described in Section 2.2), bioLQM provides an extensive choice of updating modes (in addition to the synchronous and asynchronous updating modes). The description of available updating modes can be found in [6]. Moreover, bioLQM currently supports the identification of stable states through the use of decision diagrams (extract and combine stability conditions from the logical rules as described in Subsection 3.1.3). The efficient identification of complex attractors and their reachability probabilities remains a challenge. To tackle this issue, we integrated the existing Monte Carlo, Firefront, Avatar in bioLQM.

The full instruction and examples that explain how to use bioLQM can be found in [6], where further information regarding model modifiers are explained (*e.g.,* how to define perturbations, model reduction and Boolean mapping of multi-valued models).

4.1.2 GINsim

The classical Monte Carlo simulation, as well as the Firefront and Avatar algorithms are currently implemented in GINsim [25], an integrated software built in Java, available at http://ginsim.org/. This software tool offers a user friendly graphical interface for the modeling and simulation of regulatory network. Its general architecture can be seen as follows:

- The graphical interface uses an Application Program Interface (API) in the ”serviceGUI” layer which is responsible of displaying the frame of each panel.
- the serviceGUI is responsible for calling the appropriate tool from the bioLQM API; however in the case of the Avatar, Firefront and Monte Carlo algorihms, these are implemented in GINsim.
- For the dynamical analysis of the model and its simulation, it uses the API offered by bioLQM;

- Avatar, Firefront and Monte Carlo use updaters (*i.e.,* Java classes responsible for the updating modes) in bioLQM to get the successors of a state.

An illustration of the current architecture is presented in Figure 4.1.

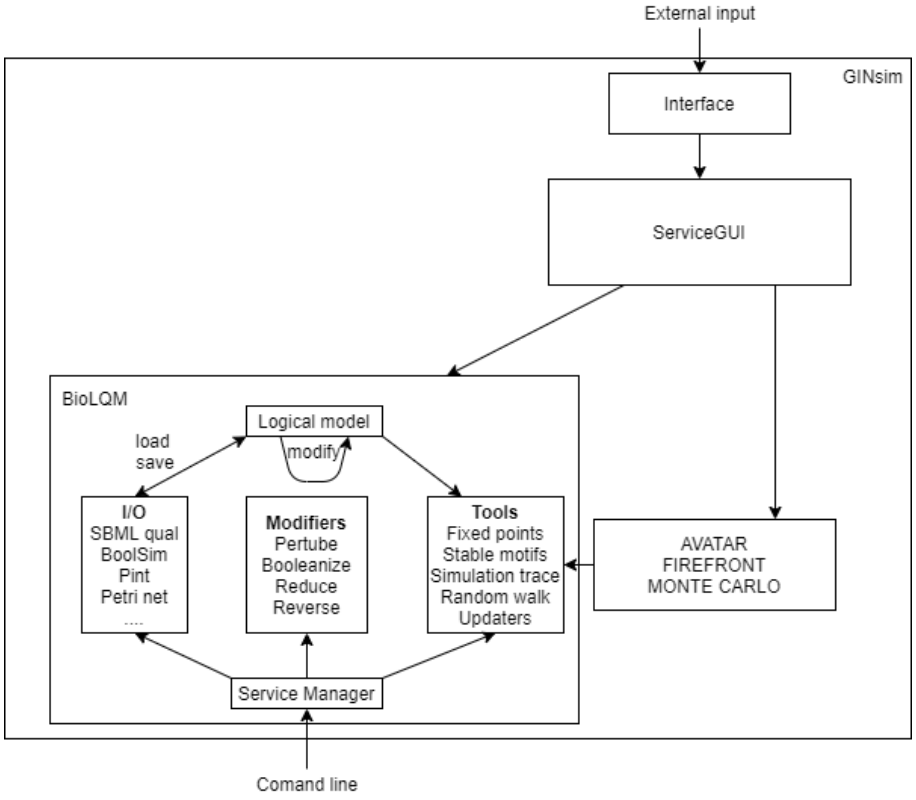


Figure 4.1: Current architecture of GINsim

4.2 Refactoring and migration of Avatar, Firefront and Monte Carlo to bioLQM

As mentioned above, Avatar, Firefront and Monte Carlo implementation was initially integrated in GINsim, with a code mixed with that of the graphical user interface.

We started by cleaning up the code and migrating the algorithms to bioLQM. In the next bioLQM release, the new versions of Avatar, Firefront and Monte Carlo will be available as new bioLQM tools. As a consequence, these algorithms can now be launched from the command line. In the future, GINsim will have a interface for Avatar, Firefront and Monte Carlo and the serviceGUI of GINsim will call the corresponding service of bioLQM to use quantification reachability algorithms. An overview of the new architecture is presented in Figure 4.2.

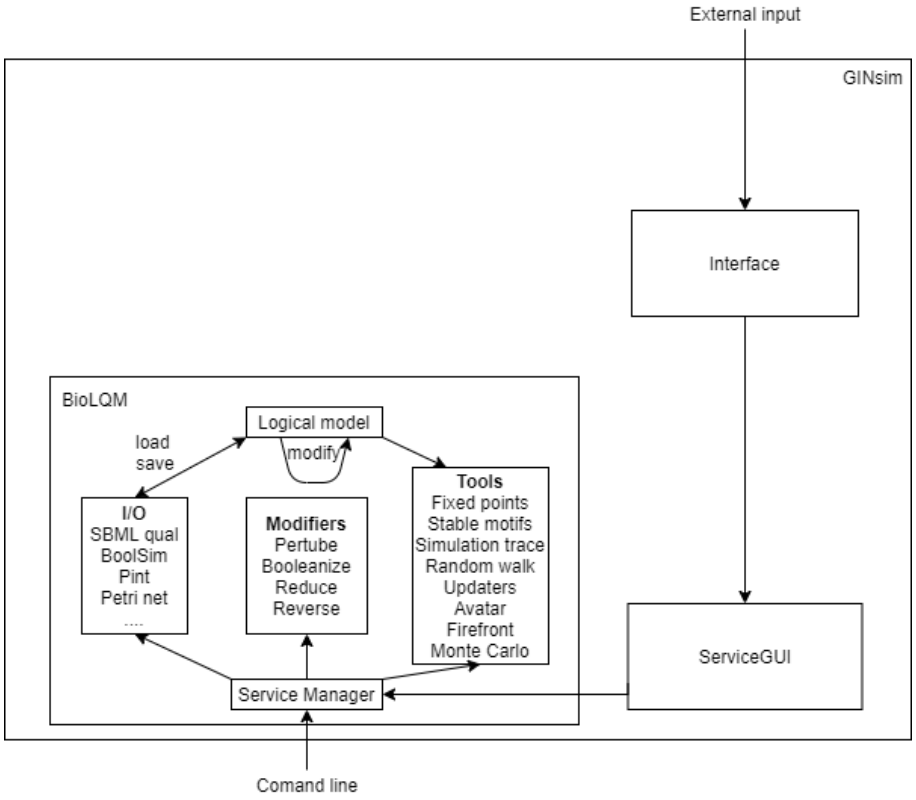


Figure 4.2: New architecture of GINsim

4.3 Supporting new updating modes in Avatar and Monte Carlo

So far, Avatar and Monte Carlo as implemented in GINsim supported the sole asynchronous update mode, considering that the probabilities of concurrent transitions were uniformly distributed. For this work, we aimed to surpass these limitations and:

- Support the definition of priority classes as described in Section 2.2. More precisely, given a state and its successors, the transition probabilities are set to zero for updates affecting genes that are in classes with lower ranks, and are uniformly distributed for updates affecting genes in the class with the higher rank (for this work, the definition of priority classes involves the sole asynchronous update).
- Account for user defined transition rates according to some knowledge of the underlying biological processes, *i.e.*, a qualitative rate associated to the update of each gene; the probabilities of concurrent transitions are then determined according to these relative rates.
- Integrate the two concepts described above in a single feature, where the algorithm handles dynamics generated using the asynchronous updating scheme with priority classes, and where transition probabilities are defined according to some biological information.

Note that Firefront was not extended. Extension of this algorithm might be considered as a future work.

4.3.1 Non-uniform transition probabilities

The integration of qualitative rates in Monte Carlo is relatively simple as this algorithm does not cope with SCCs. Therefore, when having non-uniform transition rates, the transition probability of a state to one of its successors is given by the ratio between the rate of that transition and the sum of the rates of all the transitions leaving the state.

In contrast, in Avatar, because of the rewiring procedure, when transition rates are different, the calculation of the probabilities of transitioning from the states of a cycle to the exiting states must be adapted.

To get the intuition of defining probabilities to exiting states and validate the correctness of the adapted rewiring process, we started by building examples of STGs. Let us take as an example a transient cycle (shown in Figure 4.3) and try to rewired it considering non-uniform rates.

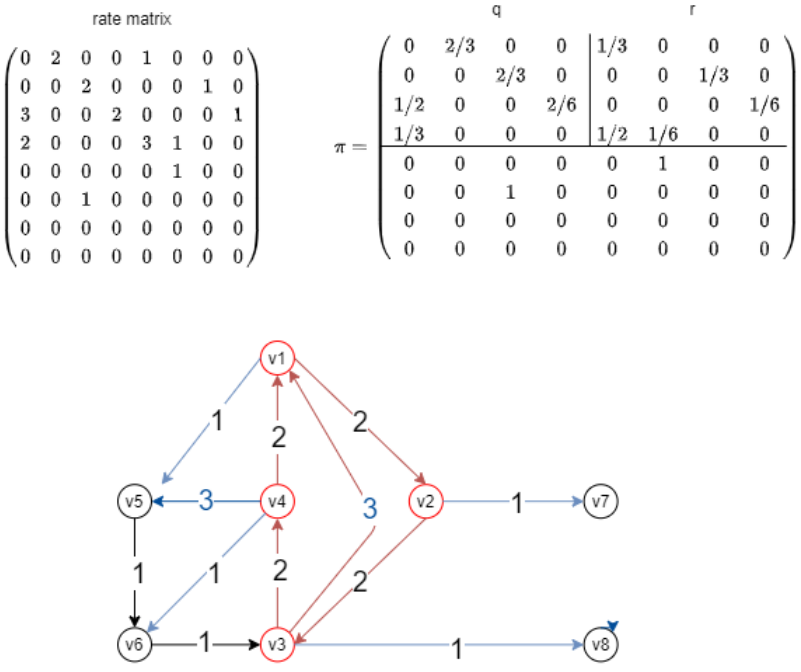
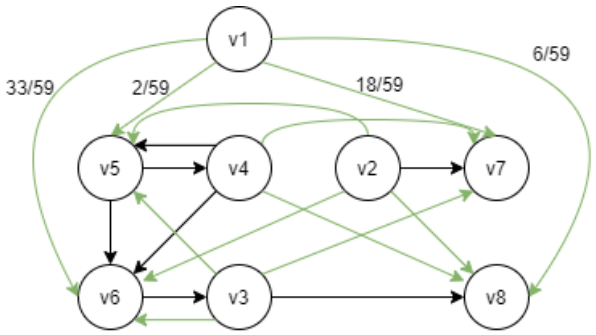


Figure 4.3: Transient cycle with non-uniform rates

Avatar has two rewiring strategies: exact exit probabilities and uniform exit probabilities. In the first approach, the integration of non-uniform transition rates does not imply any modification compared to the existing procedure (see 3.3.4). Indeed, the calculation of the new transition matrix is based on

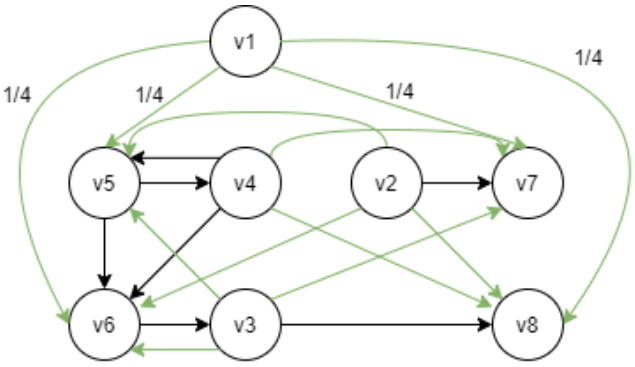
properties of absorbing Markov chains (the resulting rewired graph of the example presented above and the new transition matrix are illustrated in Figure 4.4).



$$\pi = \begin{pmatrix} 0 & 0 & 0 & 0 & 33/59 & 2/59 & 18/59 & 6/59 \\ 0 & 0 & 0 & 0 & 20/59 & 3/59 & 27/59 & 9/59 \\ 0 & 0 & 0 & 0 & 30/59 & 9/118 & 11/59 & 27/118 \\ 0 & 0 & 0 & 0 & 81/118 & 21/118 & 6/59 & 2/59 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 4.4: Rewired cycle with non-uniform transition probabilities (exact exit probabilities)

In the second approach, when probabilities of concurrent transitions are uniformly distributed, the transition probabilities of each state of the cycle to the exiting states are also uniformly distributed, according to the number of the exiting states. However, with the consideration of distinct transition rates, these probabilities must be redefined. For this, a new method to calculate the transition probabilities was designed. Given a state c_1 of a cycle C and a set of exiting states of the cycle $B = (b_1, \dots b_k)$, the probability of transitioning from c_1 to any exiting state b_i of the cycle is given by $R_{c_1\_b_i}/T\_R$ where $R_{c_1\_b_i}$ is the rate associated with the transition from state c_1 to state b_i and T_R is the sum of the rates associated with the transitions from the state c_1 to the states of B ($\sum_{i=1}^{k} R_{c_1\_b_i}$). We assumed that, if there is no direct transition from c_1 to b_i , the rate $R_{c_1\_b_i}$ is set to 1. The rewired graph for the example presented above and the new transition matrix are illustrated in Figure 4.5.



$$\pi = \begin{pmatrix} 0 & 0 & 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 \\ 0 & 0 & 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 \\ 0 & 0 & 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 \\ 0 & 0 & 0 & 0 & 1/2 & 1/6 & 1/6 & 1/6 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 4.5: Rewired cycle with non-uniform transition probabilities (uniform exit probabilities)

4.3.2 Priority Classes

The integration of priority classes (described in Section 2.2) in Avatar and Monte Carlo is relatively easy to implement. BioLQM includes in its set of tools a priority updater (Java class responsible for the

priority class updating mode). Both algorithms can thus directly use this updater to perform a simulation considering priority classes. In this scheme, updating calls concerning genes of lower rank are only possible if there are no updating calls for genes in a higher rank.

The current implementation of priority classes in bioLQM supports the definition of synchronous and asynchronous classes (*i.e.,* sets of genes in given ranks that are updated synchronously or asynchronously). Note that each class of a given rank can be associated with a different updating scheme. However, the definition of priority classes is currently being refined by other members of the group, and here we only focused on simpler definitions involving the sole asynchronous (uniform and non-uniform updates).

So far, extending Avatar and Monte Carlo to support priority classes was restricted to classes associated to the asynchronous updating scheme (potentially with rates).

4.4 Implementation of new updating modes

For both Monte Carlo and Avatar, we included the option to select the updater with priority classes. With the integration of rates, Monte Carlo and Avatar pick a successor according to the calculated probability (based on rates). Algorithm 4.1 describes how Monte Carlo and Avatar get the successors of a given state, and determines the transition probability for each of them. The parameter ”updater” is used to determine the successors of a state (asynchronous or priority) and the parameter ”rates_of_updating_calls” stores the qualitative rates associated to the updates of the model components.

Algorithm 4.1 Generate_successors_and_probs

```
   Input: current_state, updater, rates_of_updating_calls
 1: totalRate = 0
 2: successors ← updater.getSuccessors(current_state)
 3: successors_and_rates ← applyRates(current_state, successors, rates_of_updating_calls)
 4: successors_and_probs ← empty
 5: for all succ in successors_and_rates do
 6:    totalRate + = succ.rate
 7: end for
 8: for all succ in successors_and_rates do
 9:    probability ← succ.rate/totalRate
10:    successors_and_probs.add(succ, probability)
11: end for
12: return successors_and_probs
```

The two strategies for the rewiring procedure are outlined in the Algorithms 4.2 and 4.3. These algorithms describe how the rewiring of the sub-graph comprising the states of the discovered cycle and their immediate successors (i.e. exits of the cycle) is performed. More precisely, the output of the rewiring procedure is the new sub-matrix of the transition matrix describing the transitions between the

states of the cycle and the exiting states of the cycle.

We denote q as the probability sub-matrix for the transitions between the states of the cycle, and r as the probability sub-matrix for the transitions from the states of the cycle to the exiting states of the cycle.

In the first strategy, the matrix r is re-calculated as $(Id-q)^{-1}r$. In the novel version of this procedure, the transition probabilities in the matrices q and r are no longer uniformly distributed (they are calculated in the Algorithm 4.1).

Algorithm 4.2 First strategy of rewiring procedure (exact exit probability)

```
Input: Cycle, Exits, updater, rates_of_updating_calls
q = [Cycle.size()][Cycle.size()]
q ← null matrix
r = [Cycle.size()][Exits.size()]
r ← null matrix
for all v in Cycle do
    successors_and_probs ← Generate_successors(v, updater, rates_of_updating_calls)
    v_index ← Cycle.indexOf(v)
    for all w in successors_and_probs do
        if w in Exits then
            r[v_index][Exits.indexOf(w)] = w.probability
        else if w in Cycle then
            q[v_index][Cycle.indexOf(w)] = w.probability
        end if
    end for
end for
r ← (Id − q)^{-1} r
return r
```

In the second strategy, the matrix r is initialized with the rates associated with the rates of the transitions from the cycle states to the exit states (if there is no transition, then the associated rate is 1). Then it is appropriately manipulated to get the probabilities associated to the transitions from each cycle state to all the exit states.

Algorithm 4.3 Second strategy of rewiring procedure (uniform exit probability)

Input: Cycle, Exits, updater, rates_of_updating_calls

```
r = [Cycle.size()][Exits.size()]
r ← null matrix
total_rates = [Cycle.size()]
total_rates = [0...0]
for all v in Cycle do
    successors ← updater.getSuccessors(current_state)
    successors_and_rates ← applyRates(current_state, successors, rates_of_updating_calls)
    v_index ← Cycle.indexOf(v)
    for all w in Exits do
        if w in successors_and_rates then
            r[v_index][Exits.indexOf(w)] = successors_and_rates.get_rate(w)
            total_rates[Cycle.indexOf(w)] + = successors_and_rates.get_rate(w)
        else
            r[v_index][Exits.indexOf(w)] = 1
            total_rates[Cycle.indexOf(w)] + = 1
        end if
    end for
end for
for i = 0; i < Cycle.size(); i++ do
    for j = 0; j < Exits.size(); j++ do
        r[i][j] ← r[i][j] / total_rates[i]
    end for
end for
return r
```

4.5 Integration into ColoMoTo Interactive Notebook

Logical models have been successfully applied to regulatory system. The continuous growth of the community has motivated the development of an environment where scientists and developers could reach a consensus within the sub-domain of logical models, thus, promoting the cooperative development of shared standard and tools [30]. For this purpose, the Consortium for Logical Models and Tools (CoLoMoTo, http://colomoto.org) was born at Instituto Gulbenkian de Ciência (Portugal) in 2010. It is an international open community that aims at the standardization of model representation and the promotion of methods, models and tools [30].

Dynamical analysis of models of regulatory networks often relies on workflows in which different software tools are chained together [31]. Recently, the scientific community has been increasingly concerned about difficulties in reproducing published results, especially when some of these tools might be difficult to install or require a steep learning curve (thus, hindering the accessibility and the reproducibility of results) [31]. In our context, a user must be minimally familiar with the command-line commands and read the manual presented in Appendix A in order to reproduce our results.

To address the aforementioned reproducibility and reusability issues, CoLoMoTo presented a reproducibility oriented framework which provides an easy-to-use environment to edit, execute, share, and

reproduce analyses of qualitative models of biological networks by integrating various logical modeling software tools, called CoLoMoTo Interactive Notebook (available at https://colomoto.github.io/colomoto-docker/) [31]. This framework integrates a set of pre-installed tools from the CoLoMoTo community and it is possible to integrate further tools (in our context, Avatar and Monte Carlo) by defining the corresponding Python interface [31]. The framework is composed by three main components: CoLoMoTo Docker image, Python interface and Jupyter web interface [31]. Its general architecture can be seen as follow:

- The framework is distributed as a Docker image which contains the set of pre-installed software/-tools (*e.g.,* GINsim [25], bioLQM [6], MaBoSS [14]). The CoLoMoTo Docker is a container technology based on virtualization, which is lightweight and provides an isolated environment (similar to a Virtual Box, but the isolation is at software level).
- For each of the tools embedded in the image, a Python interface is developed which greatly ease its execution and fetch the results.
- Jupyter web interface enables users to interact efficiently with the system (*e.g.,* upload models, include textual annotations and get a greater visualization of the results). Moreover, the resulting file can then be saved and re-executed or shared with other users.

Currently, we are still working on the development of the Python interface, as we are waiting for the integration of Avatar and Monte Carlo in the official version of bioLQM. Subsequently, CoLoMoTo Jupyter Notebook will be able to use these algorithms by importing the upcoming version of bioLQM.

5

Evaluation

Contents

5.1 Junit test cases

Unit tests are automated tests written to ensure that a unit (*i.e.*, a section of an application) meets its design, and behaves as intended. With the continuous development and extension of software applications, the interest of unit tests is to ensure the unit's correctness along introduced extensions and modifications of the whole.

In this work, the unit is the module related to the reachability algorithms (Avatar and Monte Carlo). Three simple toy models were defined to implement such tests. The first two models were designed to test the integration of the priority updater with uniform transition probabilities. The third model integrates priority classes and non-uniform transition probabilities.

Note that, as Monte Carlo cannot assess the reachability of complex attractors (when entering a terminal SCC, the process is trapped and proceeds the exploration until reaching the maximum depth), the probabilities for reaching stable states are provided as intervals, and we denote N/D (not defined) for the probabilities reaching the complex attractors.

5.1.1 Test model 1

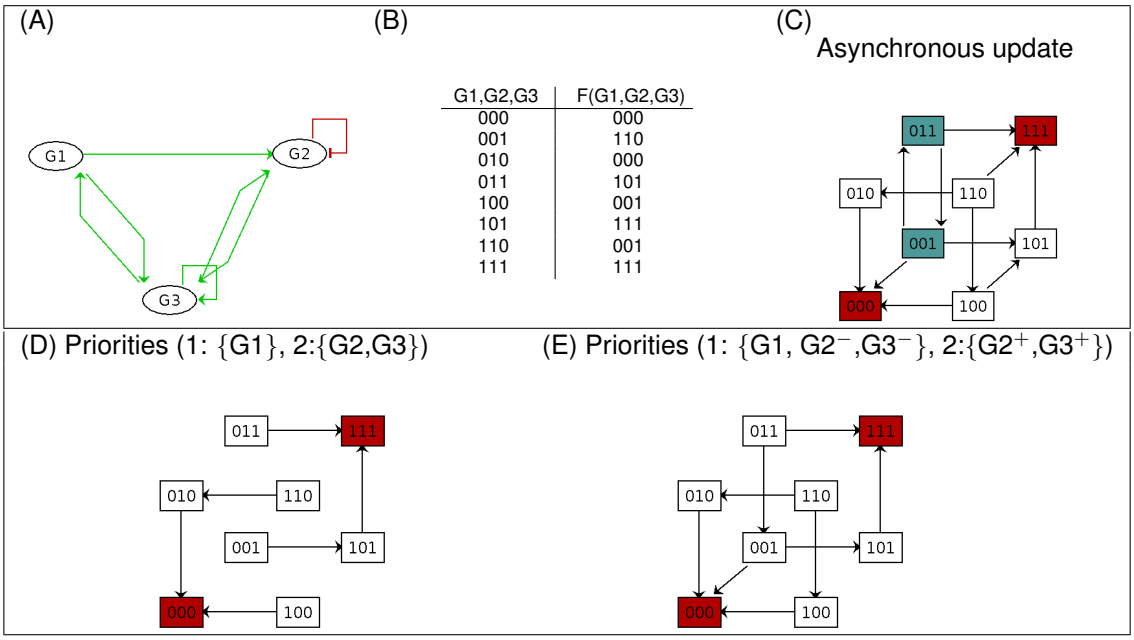


Figure 5.1: Model 1: (A) Regulatory graph; (B) Truth table; (C) STG for the asynchronous update, with 2 stable states in red and a transient SCC in green; (D) STG for the priority classes (1: {G1}, 2:{G2,G3}) with 2 stable states in red; (E) STG for the priority classes (1: {G1, G2$^-$,G3$^-$}, 2:{G2$^+$,G3$^+$}) with 2 stable states in red.

The first model encompasses 3 components. Its LRG and its dynamics are presented in Figure 5.1. We can observe that identified stable states are invariant for all updating modes, but reachability

properties vary: stable state 111 is reachable from state 100 in the asynchronous dynamics, which is not the case when considering priority classes (Figure 5.1, panel (D) and panel (E)).

Algorithm	Updating mode	Initial state	Reached attractors	Probability	% of successful simulations	Time
Avatar	Asynchronous update	[100]	SS1 [000] SS2 [111]	0.503 0.497	100	0.4 s
		[011]	SS1 [000] SS2 [111]	0.201 0.799	100	0.4 s
	Priority 1: {G1}, 2:{G2,G3}	[100]	SS1 [000] SS2 [111]	1 0	100	0.4 s
		[011]	SS1 [000] SS2 [111]	0 1	100	0.4 s
	Priority 1: {G1, G2$^-$,G3$^-$}, 2:{G2$^+$,G3$^+$}	[100]	SS1 [000] SS2 [111]	1 0	100	0.4 s
		[011]	SS1 [000] SS2 [111]	0.25 0.75	100	0.4 s
Monte Carlo	Asynchronous update	[100]	SS1 [000] SS2 [111]	[0.509, 0.509] [0.491, 0.491]	100	0.3 s
		[011]	SS1 [000] SS2 [111]	[0.206, 0.206] [0.794, 0.794]	100	0.3 s
	Priority 1: {G1}, 2:{G2,G3}	[100]	SS1 [000] SS2 [111]	[1.0, 1.0] [0, 0]	100	0.3 s
		[011]	SS1 [000] SS2 [111]	[0, 0] [1.0, 1.0]	100	0.3 s
	Priority 1: {G1, G2$^-$,G3$^-$}, 2:{G2$^+$,G3$^+$}	[100]	SS1 [000] SS2 [111]	[1.0, 1.0] [0, 0]	100	0.3 s
		[011]	SS1 [000] SS2 [111]	[0.25, 0.25] [0.75, 0.75]	100	0.3 s

Table 5.1: Reachability probabilities for the test model 1, asynchronous dynamics and two priority classes (see Fig. 5.1). Avatar's parameters: runs=1E3, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E4. Monte Carlo's parameters: runs=1E3, maximum depth=1E4.

For convenience, stable states are denoted SS and complex attractors CA.

5.1.2 Test model 2

The LRG and STG of the model 2 are presented in Figure 5.2. This model has a transient cycle in the asynchronous dynamics, that turns into a cyclic attractor for the priority classes 1: {G0$^+$, G1$^-$, G2, G3}, 2:{G0$^-$,G1$^+$} (see panel D of Figure 5.2).

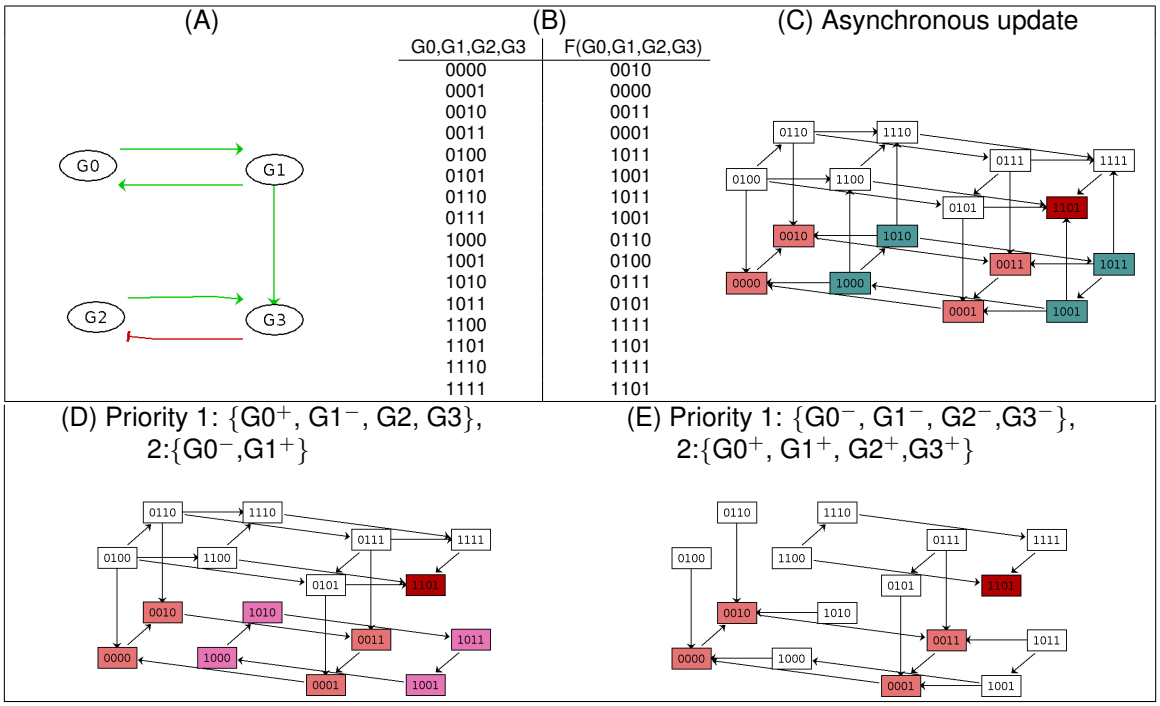


Figure 5.2: Model 2: (A) Regulatory graph; (B) Truth table; (C) STG for the asynchronous update, with 1 stable state in dark red, a complex attractor in light red and a transient SCC in green; (D) STG for the priority classes (1: {G0$^+$, G1$^-$, G2, G3}, 2:{G0$^-$,G1$^+$}), with 1 stable state in dark red, 2 complex attractors in light red and pink; (E) STG for the priority classes (1: {G0$^-$, G1$^-$, G2$^-$,G3$^-$}, 2:{G0$^+$, G1$^+$, G2$^+$,G3$^+$}) with 1 stable state in dark red and 1 complex attractor in light red.

The results of the reachability quantification for the test model 2 are presented in Table 5.2. We can assert the correctness of the results by directly analyzing the STGs and the probability of reaching each attractor. Note that, for priority classes 1: {G0$^-$, G1$^-$, G2$^-$,G3$^-$}, 2:{G0$^+$, G1$^+$, G2$^+$,G3$^+$}, Monte Carlo cannot identify any attractor starting from the initial states 1000 and 0110, as the simulation ends up in a cyclic attractor and repeats the process until reaching the maximum depth (same for the priority classes 1: {G0$^+$, G1$^-$, G2, G3}, 2:{G0$^-$,G1$^+$} and initial state 1000). As a consequence, we can observe that the time to compute these three simulations (see 9^{th}, 11^{th} and 12^{th} rows of Table 5.2) are longer compared to the time used to compute all other simulations. Indeed, in these cases, Monte Carlo indicates that 0 successful runs (below max depth) out of 1000 reached an attractor.

Algorithm	Updating mode	Initial state	Reached attractors	Probability	% of successful simulations	Time
Avatar	Asynchronous update	[1000]	SS1 [1101] CA1 [00**]	0.501 0.499	100	0.4 s
		[0110]	SS1 [1101] CA1 [00**]	0.487 0.513	100	0.4 s
	Priority 1: {G0$^+$, G1$^-$, G2, G3}, 2:{G0$^-$,G1$^+$}	[1000]	SS1 [1101] CA1 [00**] CA2 [10**]	0 0 1	100	0.4 s
		[0110]	SS1 [1101] CA1 [00**] CA2 [10**]	0.518 0.482 0	100	0.4 s
	Priority 1: {G0$^-$, G1$^-$, G2$^-$,G3$^-$}, 2:{G0$^+$, G1$^+$, G2$^+$,G3$^+$}	[1000]	SS1 [1101] CA1 [00**]	0 1	100	0.4 s
		[0110]	SS1 [1101] CA1 [00**]	0 1	100	0.4 s
Monte Carlo	Asynchronous update	[1000]	SS1 [1101] CA1 [00**]	[0.509, 1.0] N/D	50.9	3 s
		[0110]	SS1 [1101] CA1 [00**]	[0.503, 1.0] N/D	50.3	3 s
	Priority 1: {G0$^+$, G1$^-$, G2, G3}, 2:{G0$^-$,G1$^+$}	[1000]	SS1 [1101] CA1 [00**] CA2 [10**]	[0, 1.0] N/D N/D	0	6 s
		[0110]	SS1 [1101] CA1 [00**] CA2 [10**]	[0.503, 1.0] N/D N/D	50.3	3 s
	Priority 1: {G0$^-$, G1$^-$, G2$^-$,G3$^-$}, 2:{G0$^+$, G1$^+$, G2$^+$,G3$^+$}	[1000]	SS1 [1101] CA1 [00**]	[0, 1.0] N/D	0	6 s
		[0110]	SS1 [1101] CA1 [00**]	[0, 1.0] N/D	0	6 s

Table 5.2: Reachability probabilities for the test model 2, asynchronous dynamics and two priority classes (see Fig. 5.2). Avatar's parameters: runs=1E3, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E4. Monte Carlo's parameters: runs=1E3, maximum depth=1E4.

5.1.3 Test model 3

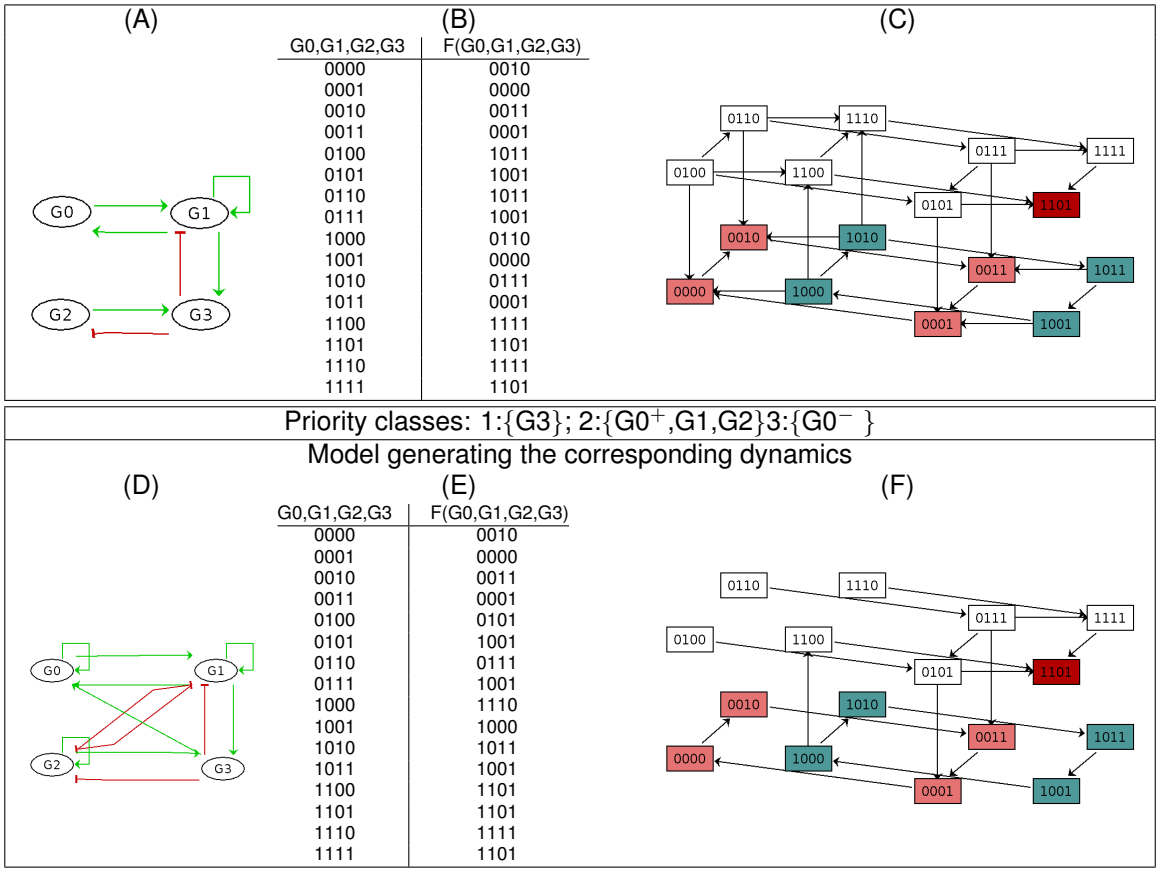


Figure 5.3: Model 3: (A) Regulatory graph; (B) Truth table; (C) Asynchronous STG, with 1 stable state and a cyclic attractor in red and a transient SCC in green. The model generating the same STG as that of the original model, with the priority classes as indicated: (D) Regulatory graph; (E) Truth table; (F) asynchronous STG, isomorphic to that of the STG of the original model with the priority classes

For this junit test case, the focus is to combine priority classes with non-uniform rates. We defined the qualitative rates associated to the updating of each gene as follows: G0:1, G1$^-$:0.5, G1$^+$:2, G2:1.5, G3:1. The test model 3 is displayed in Figure 5.3 A-C.

We intend to compare the results obtained with the revised version of Avatar against MaBoSS. However, as MaBoSS cannot assess reachability probabilities with priority settings, we built the equivalent model that generates the same dynamics as displayed in Figure 5.3 panels D-F, which is the STG of the model when considering the priority classes 1:{G3}; 2:{G0$^+$,G1,G2}3:{G0$^-$}.

The results of the simulations are presented in the tables 5.3 and 5.4. We can observe that the differences between the probabilities of reaching the stable states obtained using MaBoSS, the new version of Avatar and Monte Carlo do not exceed 0.03.

Algorithm	Updating mode	Initial state	Reached attractors	Probability	% of successful simulations	Time
Avatar	Asynchronous update uniform rates	[****]	SS1 [1101]	0.446		
			CA1 [00**]	0.554	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 0.447			
		[1001]	SS1 [1101]	0.231		
			CA1 [00**]	0.769	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 0.231			
		[0111]	SS1 [1101]	0.501		
			CA1 [00**]	0.499	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 0.495			
	Asynchronous update non-uniform rates	[****]	SS1 [1101]	0.521		
			CA1 [00**]	0.479	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 0.519			
		[1001]	SS1 [1101]	0.317		
			CA1 [00**]	0.683	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 0.312			
		[0111]	SS1 [1101]	0.669		
			CA1 [00**]	0.331	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 0.679			
	Priority classes: 1:{G3}; 2:{G0$^+$,G1,G2}; 3:{G0$^-$ } uniform rates.	[****]	SS1 [1101]	0.631		
			CA1 [00**]	0.359	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 0.631			
		[1001]	SS1 [1101]	1		
			CA1 [00**]	0	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 1			
		[0111]	SS1 [1101]	0.495		
			CA1 [00**]	0.505	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 0.495			
	Priority classes: 1:{G3}; 2:{G0$^+$,G1,G2}; 3:{G0$^-$ } non-uniform rates.	[****]	SS1 [1101]	0.676		
			CA1 [00**]	0.324	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 0.673			
		[1001]	SS1 [1101]	1		
			CA1 [00**]	0	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 1			
		[0111]	SS1 [1101]	0.487		
			CA1 [00**]	0.513	100	0.4 s
			Probability of the SS1 w/ MaBoSS: 0.493			

Table 5.3: Reachability probabilities evaluated by Avatar for the model 3, asynchronous dynamics and priority classes (see Fig. 5.3). Avatar parameters: runs=1E3, expansion limit=1E4, rewiring limit=1E4, tau=3, minimum SCC to rewire=4, maximum depth=1E4, rates= G0:1,G1[-]:0.5,G1[+]:2,G2:1.5,G3:1

Algorithm	Updating mode	Initial state	Reached attractors	Probability	% of successful simulations	Time
Monte Carlo	Asynchronous update uniform rates	[****]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 0.447	[0.456, 1.0] N/D	45.6	3 s
		[1001]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 0.231	[0.233, 1.0] N/D	23.3	4 s
		[0111]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 0.495	[0.501, 1.0] N/D	50.1	3 s
	Asynchronous update non-uniform rates	[****]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 0.519	[0.521, 1.0] N/D	52.1	3 s
		[1001]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 0.312	[0.302, 1.0] N/D	30.2	4 s
		[0111]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 0.679	[0.681, 1.0] N/D	68.1	3 s
	Priority classes: 1:{G3}; 2:{G0$^+$,G1,G2}; 3:{G0$^-$ } uniform rates	[****]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 0.631	[0.635, 1.0] N/D	63.5	3 s
		[1001]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 1	[1.0, 1.0] N/D	100	0.2 s
		[0111]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 0.495	[0.495, 1.0] N/D	49.5	4 s
	Priority classes: 1:{G3}; 2:{G0$^+$,G1,G2}; 3:{G0$^-$ } non-uniform rates	[****]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 0.673	[0.676, 1.0] N/D	67.6	3 s
		[1001]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 1	[1.0, 1.0] N/D	100	0.2 s
		[0111]	SS1 [1101] CA1 [00**] Probability of the SS1 w/ MaBoSS: 0.493	[0.488, 1.0] N/D	48.8	4 s

Table 5.4: Reachability probabilities evaluated by Monte Carlo for the model 3, asynchronous dynamics and priority classes (see Fig. 5.3). Monte Carlo's parameters: runs=1E3, maximum depth=1E4, rates= G0:1,G1[-]:0.5,G1[+]:2,G2:1.5,G3:1

We can observe that, for both priority settings, starting from 1001, Monte Carlo only needed 0.2 seconds to finish because all runs ended up in the stable state, at a limited depth. In contrast, when considering the whole state space as initial conditions, about 1/3 of the runs were trapped in the cyclic attractor (see table 5.3), requiring a larger number of simulation steps (the maximum depth specified).

5.2 Synthetic models

In order to ensure the consistency of the algorithm modifications, we aimed to reproduce the results listed in Table 2 of [1]. We chose the same five models that were used to test the original version of Avatar (see Table 1 of [1]). These are presented in Figures 5.4, 5.5, 5.6, 5.7 and 5.8 (due to the large size of the models, it is not possible to display their STGs, we rather provide information regarding the dynamics). To avoid deviations caused by the different performances of computers, we re-ran the algorithms, previous and revised versions, on the same computer. We compared the results in terms of reachability probabilities and of running times. The results of the simulations (Avatar and Monte Carlo) are listed below in Tables 5.5, 5.6, 5.7, 5.8 and 5.9, for the five synthetic models respectively.

We can observe that there are some discrepancies between the results obtained in the original and revised version of Monte Carlo algorithms. After analysing the probabilities, it is clear that the old version of Monte Carlo had some bugs that have been solved during this refactoring.

Note that the synthetic model 2 has a large transient SCC composed by 8192 states in an asynchronous dynamics. Starting from the state 0000001000000100, the process must traverse such a SCC to reach the stable state $SS3$ (1111100101101001). As a consequence, some runs reached the predefined maximum depth (10^4) and failed to reach attractor $SS3$. Indeed, as we can observe in the Table 5.9, only 6.2% of the Monte Carlo runs were successful. To address this issue, we had to raise the maximum depth to 10^7 in order to increase the percentage of successful simulations. Nevertheless, still 0.3% of the Monte Carlo runs were unsuccessful and the presence of such a transient SCC causes Monte Carlo to revisit states of the SCC an unrestricted number to times, leading to time overload (see Table 5.10). In contrast, Avatar is able to complete all the runs and offers a better performance by rewiring the encountered cycles.

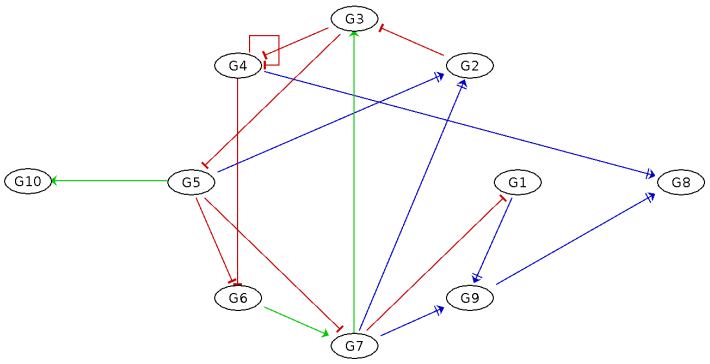
Random model 1	Model dynamics (asynchronous update)
	Number of states: 1024 Largest transient: 880 Number of stable state: 1 SS1: 0110011010 Number of complex attractor: 1 CA1: 110*100*11

Figure 5.4: Random model 1 from [1].

Initial state	Algorithm	Version	Reached attractor	Probability	% of successful simulations	Time
[0000000000]	Avatar	old	SS1 CA1	0.672 0.328	100	1 s
		revised	SS1 CA1	0.683 0.317	100	1 s
	Monte Carlo	previous	SS1	[0.905, 1.0]	90.5	2 s
		revised	SS1	[0.674, 1.0]	67.4	1 s

Table 5.5: Reachability probabilities for the random model 1. Avatar parameters: runs=1E3, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E4, asynchronous update, uniform transition probability. Monte Carlo's parameters: runs=1E3, maximum depth=1E4, asynchronous update, uniform transition probability.

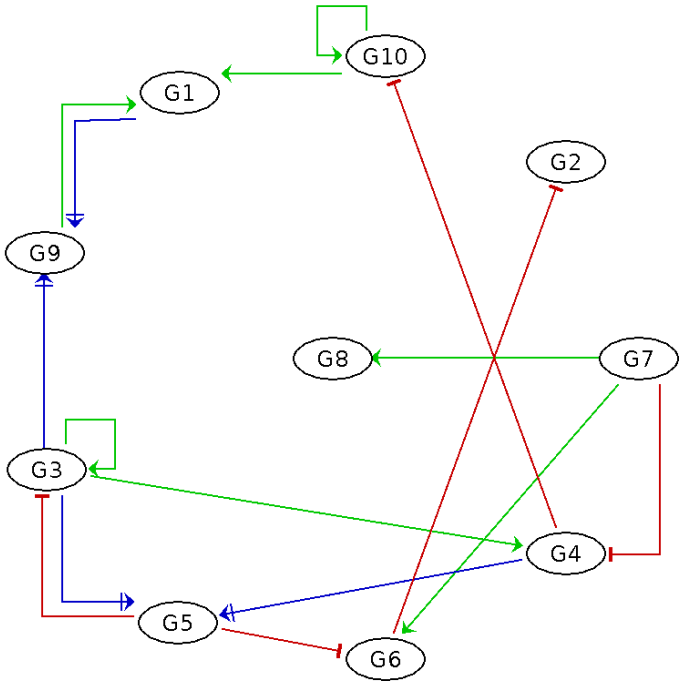
Random model 2	Model dynamics (asynchronous update)
	Number of states: 88 Largest transient: 4 Number of stable state: 1 SS1: 0011011110 Number of complex attractor: 1 CA1: *0110111*1

Figure 5.5: Random model 2 from [1].

Initial state	Algorithm	Version	Reached attractor	Probability	% of successful simulations	Time
[0100011100]	Avatar	previous	SS1 CA1	0.246 0.754	100	0.1 s
		revised	SS1 CA1	0.265 0.735	100	0.4 s
	Monte Carlo	previous	SS1	[0.794, 1.0]	79.4	1 s
		revised	SS1	[0.243, 1.0]	24.3	4 s

Table 5.6: Reachability probabilities for the random model 2. Avatar parameters: runs=1E3, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E4. Monte Carlo parameters: runs=1E3, maximum depth=1E4.

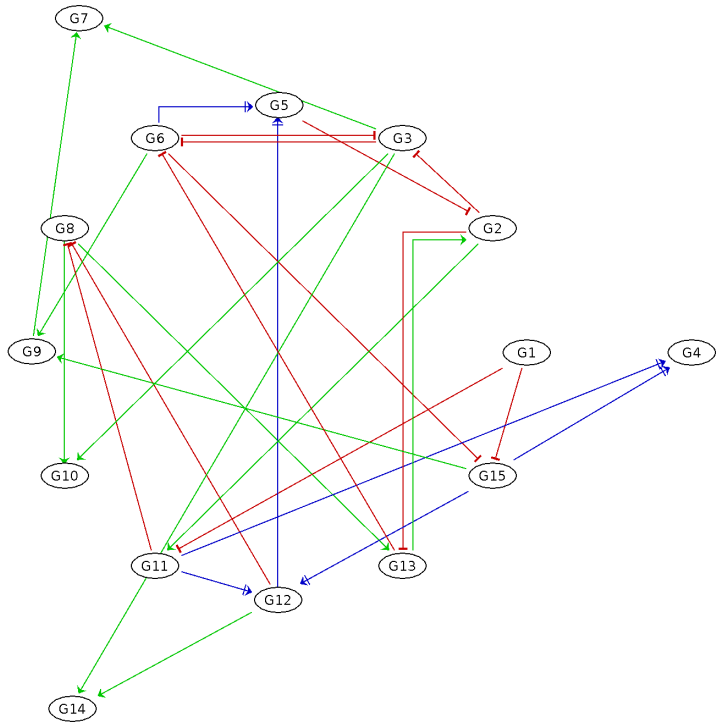
Random model 3	Model dynamics (asynchronous update)
	Number of states: 1408 Largest transient: 168 Number of stable state: 1 SS1: 101010000001110 Number of complex attractor: 1 CA1: 1*0001001001*00

Figure 5.6: Random model 3 from [1].

Initial state	Algorithm	Version	Reached attractor	Probability	% of successful simulations	Time
[100000000000000]	Avatar	previous	SS1 CA1	0.211 0.789	100	0.1 s
		revised	SS1 CA1	0.208 0.792	100	0.7 s
	Monte Carlo	previous	SS1	[0.081, 1.0]	8.1	7 s
		revised	SS1	[0.216, 1.0]	21.6	4 s

Table 5.7: Reachability probabilities for the random model 3. Avatar parameters: runs=1E3, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E4. Monte Carlo's parameters: runs=1E3, maximum depth=1E4.

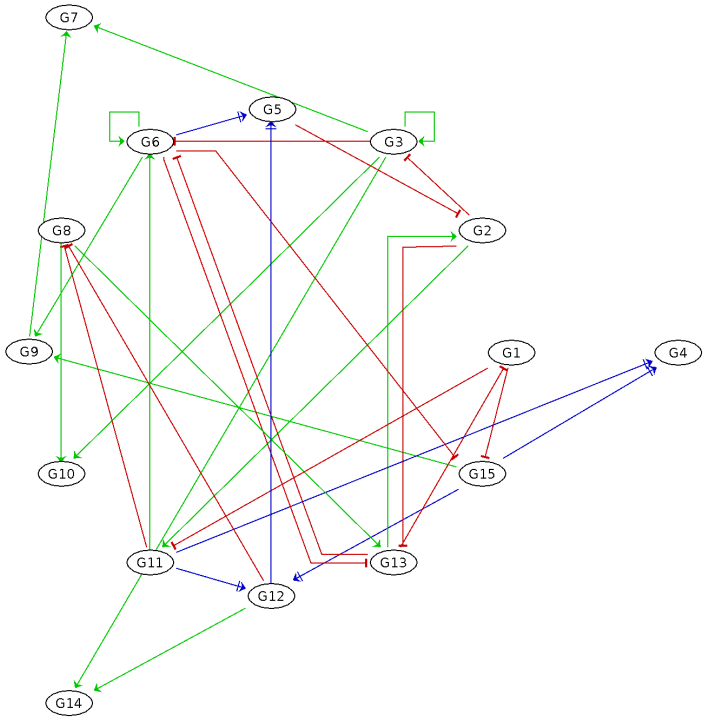
Synthetic model 1	Model dynamics (asynchronous update)
	Number of states: 28320 Largest transient: 1024 Number of stable state: 1 SS1: 101001101001010 Number of complex attractor: 1 CA1: **1**0*********

Figure 5.7: Synthetic model 1 from [1].

Initial state	Algorithm	Version	Reached attractor	Probability	% of successful simulations	Time
[000001100110111]	Avatar	previous	SS1 CA1	0.595 0.405	100	4 s
		revised	SS1 CA1	0.562 0.438	100	4 s
	Monte Carlo	previous	SS1	[0.222, 1.0]	22.2	6 s
		revised	SS1	[0.597, 1.0]	59.7	5 s

Table 5.8: Reachability probabilities for the synthetic model 1. Avatar parameters: runs=1E3, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E4. Monte Carlo's parameters: runs=1E3, maximum depth=1E4.

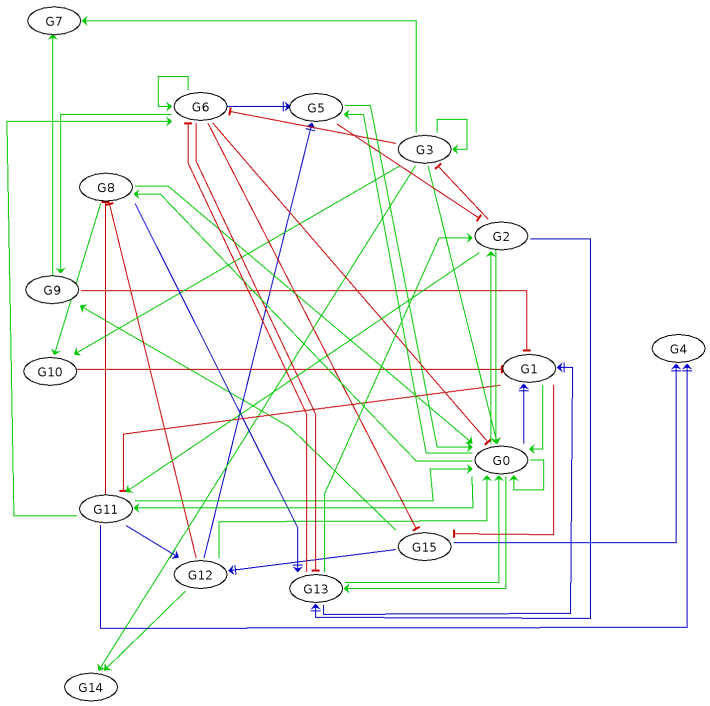
Synthetic model 2	Model dynamics (asynchronous update)
	Number of states: 16224 Largest transient: 8192 Number of stable state: 3 Stable state 1 (SS1): 1010011010010100 Stable state 2 (SS2): 1101100100101001 Stable state 3 (SS3): 1111100101101001

Figure 5.8: Synthetic model 2 from [1].

Initial state	Algorithm	Version	Reached attractor	Probability	% of successful simulations	Time
[0000001000000100]	Avatar	previous	SS1 SS3	0.07 0.579	64.9	142 s
		revised	SS1 SS3	0.069 0.667	73.6	173 s
	Monte Carlo	previous	SS1	[0.008, 1.0]	0.8	8 s
		revised	SS1 SS3	[0.068, 0.996] [0.004, 0.932]	6.2	9 s

Table 5.9: Reachability probabilities for the synthetic model 2. Avatar parameters: runs=1E3, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E4. Monte Carlo's parameters: runs=1E3, maximum depth=1E4.

Initial state	Algorithm	Version	Reached attractor	Probability	% of successful simulations	Time
[0000001000000100]	Avatar	revised	SS1 SS3	0.065 0.935	100	267 s
	Monte Carlo	revised	SS1 SS3	[0.072, 0.075] [0.925, 0.928]	99.7	1610 s

Table 5.10: Reachability probabilities evaluated the by revised version of Monte Carlo for the synthetic model 2. Avatar parameters: runs=1E3, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E6. Monte Carlo's parameters: runs=1E3, maximum depth=1E6.

5.3 Biological models

To further evaluate our implementation on published biological models, we considered the segment polarity module proposed by L. Sánchez *et al.* in 2008 [2], available at http://ginsim.org/model/SP, and a model of T helper cells differentiation proposed by L. Mendoza in 2006 [3], available at http://ginsim.org/node/41.

5.3.1 The segment polarity model

The segment polarity model describes a specific step of the segmentation typically involved in the *Drosophila* embryo (see Figure 5.9).

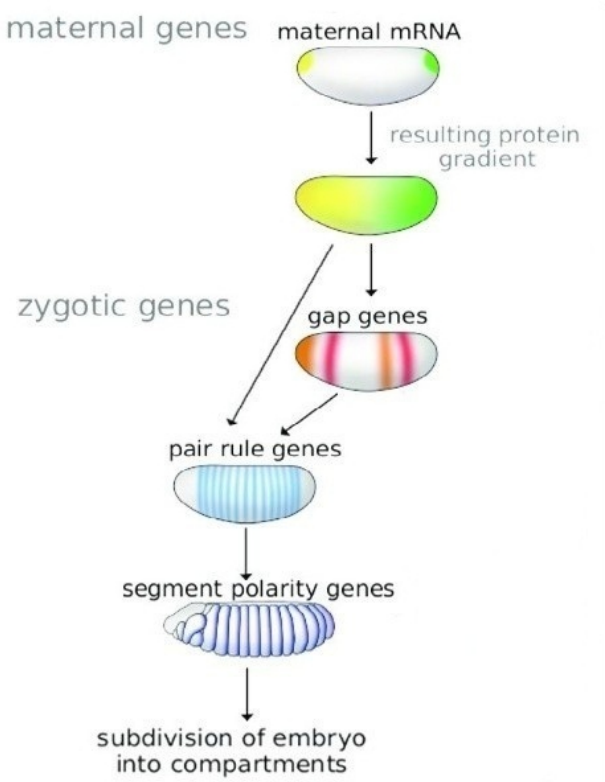


Figure 5.9: The segmentation process involved in the development of the fly *Drosophila* (image adapted from SD. Hueber's PhD thesis)

Note that the model does not consider the whole process of the development starting from the zygote. Indeed, this model focuses on the process that consolidates the borders between the embryo segments, starting from the state induced by the pair-rule genes.

To account for this segmentation process, the model needs to consider 6 neighbouring cells, including cell-cell signalling mechanisms. Its intra-cellular network encompasses 12 nodes, namely Fz, Dsh, En, Ptc, Slp, Nkd, Pka, Ciact, Cirep, Ci, Wg, Hh (see Figure 5.10). The network has 2 input components

(Wg_input and Hh_input) that correspond to the receptors of signals emitted by neighbouring cells. These are defined by Wg and Hh signals as shown in Figure 5.11.

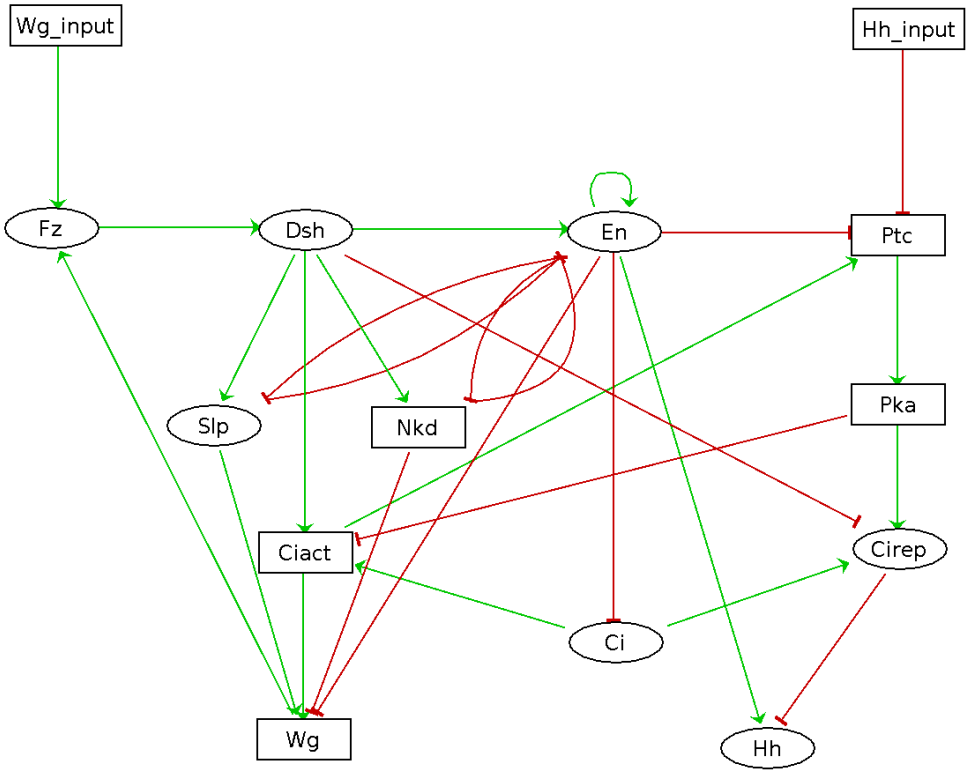


Figure 5.10: Intra-cellular network of the segment polarity model.

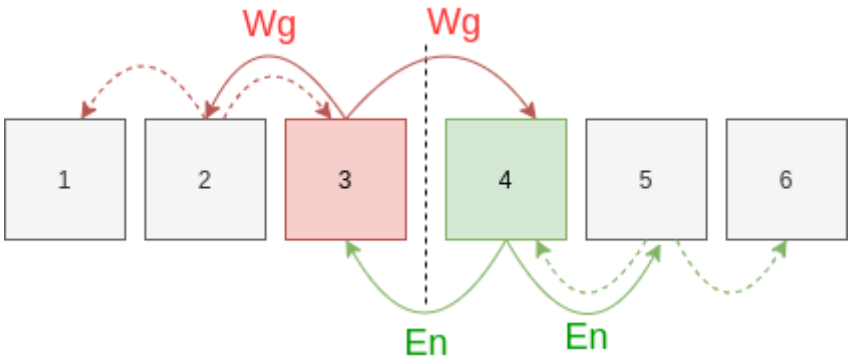


Figure 5.11: Inter-cellular interactions of segment polarity model

By fixing the inputs, we performed the reachability analysis for the intra-cellular network. The attractors obtained are presented in Table 5.11. By omitting the 2 stable states obtained by activating both inputs Wg_input and Hh_input (taking into account that a cell should not receive both signals), we obtained 5 stable states denoted, respectively, trivial phenotype (**T**), **C** phenotype, **Wg** phenotype, **En**

phenotype and Nkd phenotype (**N**). Table 5.11 shows these stable states.

Wg _input	Hh _input	Wg	Fz	Dsh	Slp	Nkd	En	Hh	Ci	Ciact	Cirep	Pka	Ptc	State
0	0	0	0	0	0	1	0	0	1	0	1	2	1	**T**
0	1	0	0	0	0	1	0	0	1	1	0	0	0	**C**
0	1	2	1	1	1	2	0	0	1	2	0	0	0	**Wg**
1	0	0	1	1	0	0	1	1	0	0	0	0	0	**En**
1	0	0	1	1	1	2	0	0	1	1	0	2	2	**N**
1	1	2	1	1	1	2	0	0	1	2	0	0	0	**Wg**
1	1	0	1	1	0	0	1	1	0	0	0	0	0	**En**

Table 5.11: Stable states of the segment polarity model (single cell).

Furthermore, by taking into consideration that protein modification events are faster than gene expression processes, Sánchez *et al.* proposed two priority classes $\{1{:}\{$Ciact, Cirep, Fz, Dsh, Pka$\}$; $2{:}\{$Ci, Wg, Nkd, En, Slp, Hh, Ptc$\}$ [2]. These priority configurations were used when assessing the reachability probabilities with Avatar and Monte Carlo algorithms. The results of reachability probabilities for the model with a single cell are presented in Table 5.12.

We can observe that, with the introduction of priority classes or non-uniform rates, the probability of reaching the **Wg** phenotype state by having the input Hh_input activated is higher compared to the reachability probability in an asynchronous and uniform rates model dynamics.

Algorithm	initial states	Updating mode	Inputs	Reached	Probability	% of successful simulations	Time
Avatar	[**********] whole state space	Asynchronous update Uniform rates	Wg_input= 0 Hh_input= 0	**T**	1.0	100	0.7 s
			Wg_input= 0 Hh_input= 1	**C** **Wg**	0.858 0.142	100	0.6 s
			Wg_input= 1 Hh_input= 0	**En** **N**	0.503 0.497	100	0.5 s
		Priority classes 1:{Ciact,Cirep,Fz, Dsh, Pka}; 2:{Ci,Wg,Nkd,En,Slp, Hh,Ptc}. Uniform rates	Wg_input= 0 Hh_input= 0	**T**	1.0	100	0.7 s
			Wg_input= 0 Hh_input= 1	**C** **Wg**	0.753 0.247	100	0.6 s
			Wg_input= 1 Hh_input= 0	**En** **N**	0.493 0.507	100	0.5 s
		Asynchronous update. Rates: Ciact, Cirep, Fz, Dsh, Pka: 50; Ci, Wg, Nkd, En, Slp, Hh, Ptc: 1	Wg_input= 0 Hh_input= 0	**T**	1.0	100	0.7 s
			Wg_input= 0 Hh_input= 1	**C** **Wg**	0.767 0.233	100	0.7 s
			Wg_input= 1 Hh_input= 0	**En** **N**	0.498 0.502	100	0.6 s
Monte Carlo	[**********] whole state space	Asynchronous update. Uniform rates	Wg_input= 0 Hh_input= 0	**T**	[1.0, 1.0]	100	0.3 s
			Wg_input= 0 Hh_input= 1	**C** **Wg**	[0.843, 0.843] [0.157, 0.157]	100	0.3 s
			Wg_input= 1 Hh_input= 0	**En** **N**	[0.503, 0.503] [0.497, 0.497]	100	0.3 s
		Priority classes 1:{Ciact,Cirep,Fz, Dsh, Pka}; 2:{Ci,Wg,Nkd,En,Slp, Hh,Ptc}. Uniform rates	Wg_input= 0 Hh_input= 0	**T**	[1.0, 1.0]	100	0.3 s
			Wg_input= 0 Hh_input= 1	**C** **Wg**	[0.764, 0.764] [0.236, 0.236]	100	0.3 s
			Wg_input= 1 Hh_input= 0	**En** **N**	[0.501, 0.501] [0.499, 0.499]	100	0.3 s
		Asynchronous update. Rates: Ciact, Cirep, Fz, Dsh, Pka: 50; Ci, Wg, Nkd, En, Slp, Hh, Ptc: 1	Wg_input= 0 Hh_input= 0	**T**	[1.0, 1.0]	100	0.3 s
			Wg_input= 0 Hh_input= 1	**C** **Wg**	[0.769, 0.769] [0.231, 0.231]	100	0.3 s
			Wg_input= 1 Hh_input= 0	**En** **N**	[0.513, 0.513] [0.487, 0.487]	100	0.3 s

Table 5.12: Reachability probabilities for the segment polarity model (single cell). Avatar parameters: runs=1E3, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E6, initial state = whole state space. Monte Carlo's parameters: runs=1E3, maximum depth=1E6, initial space= whole state space.

When considering the inter-cellular model connected through cell signaling, we have a huge state space with $(3^5 * 2^7)^{\sharp num\_of\_cells}$ states and a maximum of $5^{\sharp num\_of\_cells}$ stable states. However, this number would be effective if the inter-cellular signaling did not restrict the compatibility between neighbouring stable states by interconnecting the intra-cellular network, as it might have no effect. Hence, we have a restricted number of stable states.

In this case study, we are particularly interested in the model dynamics of the complete model of 6 connected cells. Since the pattern of expression induced by the pair-rule genes is the initial condition for the subsequent segment polarity module [2], we performed the reachability analysis starting from this initial state. As a result, the system leads to two significant stable states: a stable state composed by the sequence of cellular phenotypes **T Nkd Wg En C T**, denoting the wild type pattern (the pattern observed experimentally), and a stable state composed by all cells in the trivial state (**T**), which is a consequence of an early loss of the pair-rule signal, *i.e.*, loss of **Wg** and **En** expression in the cells 3 and 4, which are responsible for defining the borders between the two segments [2] (see Figure 5.12). It is important to

mention that, the trivial state is not biologically observed, and the probability of reaching this stable state is the highest (approximately 0.85) in an asynchronous dynamics. With the introduction of the priority classes and non-uniform transition probabilities, we expect to obtain the wild type pattern as the most probable attractor.

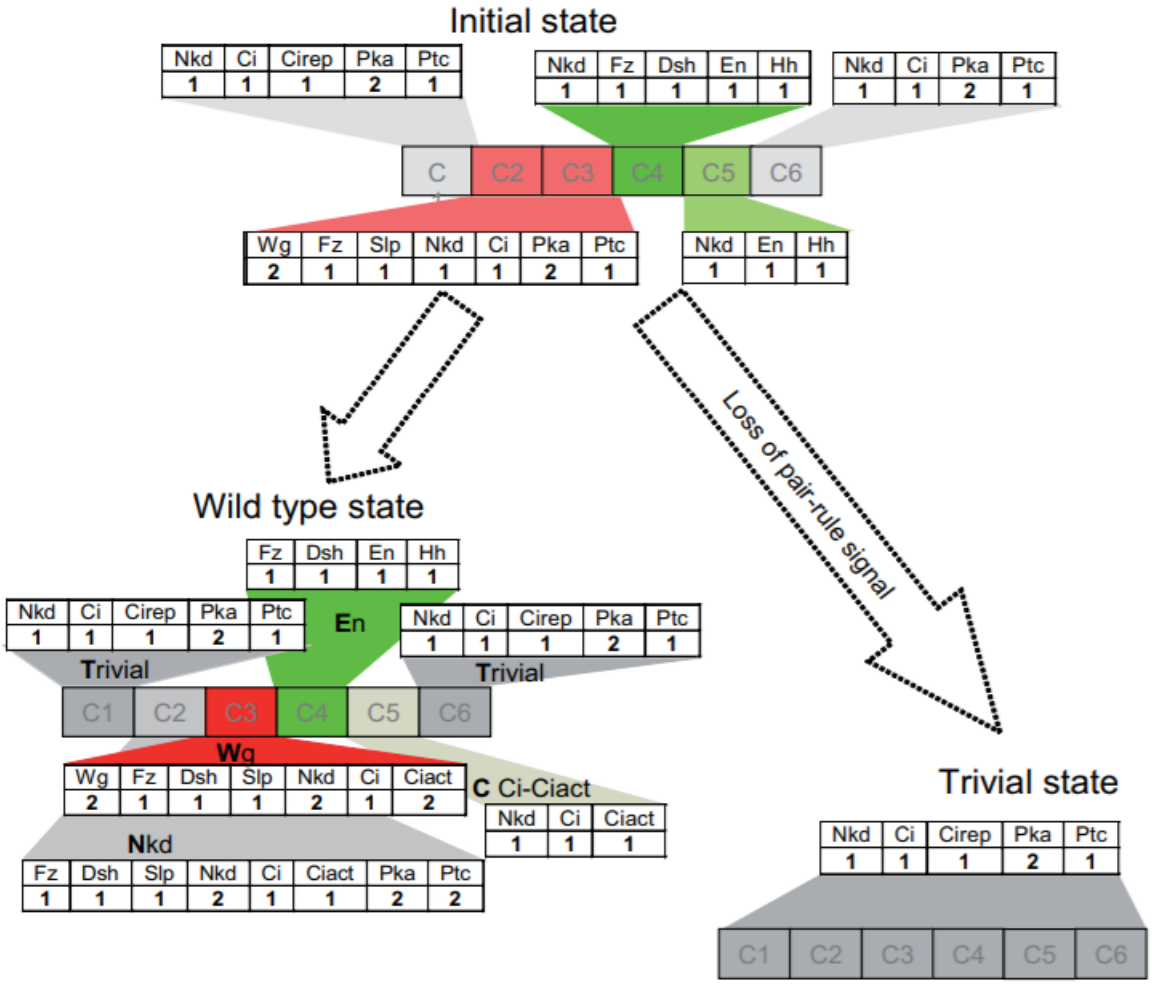


Figure 5.12: Model dynamic of segment polarity model (6 cells) by considering priority classes 1: {Ciact, Cirep, Fz, Dsh, Pka}; 2: {Ci, Wg, Nkd, En, Slp, Hh, Ptc}. Image retrieved from [2]

Results are presented in Table 5.13. Note that all attractors not explicitly specified in the table are stable states that have a very low reachability probability. As all runs performed by Monte Carlo were successful, and no complex attractors were identified by Avatar. Overall, we were thus able to recover the results presented in [2], complementing them with estimates of the attractor probabilities. We can observe that, while the probability of reaching the trivial state is the highest in an asynchronous dynamics, with the introduction of priority settings, the probability of reaching the wild type state has raised significantly, becoming the most probable attractor. Moreover, as an alternative to the priority settings, we performed the reachability analysis by considering that the rate of the transitions related to nodes of the high priority class is 50 and the rate of the transitions related to nodes of the low priority class is 1. As a result, the reachability probabilities obtained are similar to the probabilities obtained with the priorities. Interestingly, when integrating both priority settings and non-uniform transition probabilities (we considered that synthesis of a regulatory factor, *i.e.*, transition increasing the value of the component, is 50 times faster than its degradation, *i.e.*, transition increasing its value), the probability of reaching the trivial state has dropped significantly (from 0.84 to 0.14).

Regarding the performances of the algorithms, Avatar is constrained by the need to assess the complex structure of the model. The largest transient encountered has over 3 million of states and Avatar struggles at identifying those transients. The reachability probabilities obtained with Avatar might be more precise but, this model has no cyclic attractors, Monte Carlo was able to retrieve, more efficiently most of the attractors (in particular those stable states reachable without the need to visit large transient SCCs).

Algorithm	Initial state	Updating mode	Reached attractors	Probability	% of successful simulations	Time
Avatar	Pair-rule module	Asynchronous update Uniform rates	**T N Wg En C T** **T T T T T T** **C En Wg En C T** other attractors	0.076 0.8453 0.0476 0.0311	100	239 s
		Priority classes 1: {Ciact, Cirep, Fz, Dsh, Pka}; 2:{Ci, Wg, Nkd, En, Slp, Hh, Ptc}. Uniform rates	**T N Wg En C T** **T T T T T T** **En Wg Wg En C T** other attractors	0.5822 0.2926 0.0707 0.0545	100	43 s
		Asynchronous update. Rates: Ciact, Cirep, Fz, Dsh, Pka: 50; Ci, Wg, Nkd, En, Slp, Hh, Ptc:1	**T N Wg En C T** **T T T T T T** **En Wg Wg En C T** other attractors	0.5775 0.2905 0.0675 0.0665	100	306 s
		Priority classes 1:{Ciact, Cirep, Fz, Dsh, Pka}; 2:{C1, Wg, Nkd, En, Slp, Hh, Ptc} Rate for the increasing update: 50; Rate for the decreasing update: 1	**T N Wg En C T** **T T T T T T** **En Wg Wg En C T** other attractors	0.5191 0.1551 0.2351 0.0907	100	40 s
Monte Carlo	Pair-rule module	Asynchronous update Uniform rates	**T N Wg En C T** **T T T T T T** **C En Wg En C T** other attractors	[0.0795, 0.0795] [0.8415, 0.8415] [0.0479, 0.0479] [0.0311, 0.0311]	100	6 s
		Priority classes 1:{Ciact, Cirep, Fz, Dsh, Pka}; 2:{Ci, Wg, Nkd, En, Slp, Hh, Ptc}. Uniform rates	**T N Wg En C T** **T T T T T T** **En Wg Wg En C T** other attractors	[0.5786, 0.5786] [0.2965, 0.2965] [0.0672, 0.0672] [0.0577, 0.0577]	100	7 s
		Asynchronous update. Rates: Ciact, Cirep, Fz, Dsh, Pka: 50; Ci, Wg, Nkd, En, Slp, Hh, Ptc:1	**T N Wg En C T** **T T T T T T** **En Wg Wg En C T** other attractors	[0.5567, 0.5567] [0.317, 0.317] [0.0607, 0.0607] [0.0656, 0.0656]	100	6 s
		Priority classes 2:{Ciact, Cirep, Fz, Dsh, Pka}; 2:{Ci, Wg, Nkd, En, Slp, Hh, Ptc} Rate for the increasing update: 50 Rate for the decreasing update: 1	**T N Wg En C T** **T T T T T T** **En Wg Wg En C T** other attractors	[0.5121, 0.5121] [0.1443, 0.1443] [0.2545, 0.2545] [0.0891, 0.0891]	100	6 s

Table 5.13: Reachability probabilities for the segment polarity model (6 cells). Avatar parameters: runs=1E4, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E6. Monte Carlo's parameters: runs=1E4, maximum depth=1E6.

Finally, when considering different type of genetic background, we applied some perturbations on the model to observe their impact on the model dynamics. A gene mutation can be classified into loss-of-function, ectopic expression or partial loss-of-function where the activity levels of the gene is constrained, respectively, to 0, its maximal level, and the range *i.e.*, 0 to 1 for a multi-valued component. We can observe from the Tables 5.14 and 5.15 that, when applying perturbations to the model, some attractors not identified in a wild type situation arose. Most of our results are coherent with the outcomes presented by Sanchez *et al.* (see Appendix of [2]). However, according to our results, for the loss-of-function of Nkd, it is not possible to reach the expected pattern, where "the Wg-stripe expands anteriorly, whereas the En-stripe expands posteriorly and induces Wg in the adjacent posterior cell" [2].

Genetic background	Most probable attractors	Probability	Description
Wild-type	**T N Wg En C T** **T T T T T T** other attractors	0.5822 0.2926 0.1252	**T** Trivial cell, **N** Nkd cell, **Wg** Wg cell, **En** En cell, **C** Ci,Ci-act cell
Loss of function of Wg, En, Hh and Ci	**T T T T T T**	1.0	**T** trivial cell
Loss of function of Ptc	**Wg Wg Wg En C C** **En Wg Wg En C C** other attractors	0.415 0.393 0.192	**Wg** Wg cell, **En** En cell, **C** Ci,Ci-act cell
Loss of function of Ptc and Hh	**Wg Wg Wg En* C C** **e Wg Wg e C C** other attractors	0.401 0.381 0.218	**Wg** Wg cell, **En*** En cell without Hh expression, **C** Ci,Ci-act cell
Loss of function of Nkd	**T* A Wg* En* C* T*** **A A Wg* En C* T*** other attractors	0.4921 0.1325 0.4054	**T*** Trivial cell without Nkd expression, **A** previous Nkd cell that loses Nkd expression and gains medium level of Wg, **Wg*** Wg cell without Nkd expression, **C*** C cell without Nkd expression
Loss of function of Ptc, Wg	**C C C C C C**	1.0	**C** Ci,Ci-act cell
Loss of function of Ptc; Partial loss-of-function of Wg	**C Wg* Wg* C C C** **Wg* Wg* Wg* C C C** **C Wg* Wg* C C C** **C C Wg* C C C** other attractors	0.6185 0.1521 0.0961 0.0875 0.0458	**C** Ci,Ci-Ciact cell, **Wg*** Wg cell having medium level of Wg
Ectopic expression of En	**En* En* En* En* En* En***	1.0	**En*** Cell expressing only En and Hh
Ectopic expression of Wg	**En* Wg Wg En* En* En*** **N* N* Wg En* En* En*** **En* Wg Wg En* En* Wg** **N* N* Wg En* En* Wg** other attractors	0.211 0.192 0.195 0.197 0.205	**En*** En cell having an ctopic expression of Wg **N*** Nkd having an ectopic expression of Wg, **Wg** Wg cell
Ectopic expression of Nkd	**T* N Wg En* C* T** **T* T* T* T* T* T*** other attractors	0.5812 0.2943 0.1245	**T*** Trivial cell with Nkd at its maximal level, **N** Nkd cell, **Wg** Wg cell, **En*** En cell with Nkd at its maximal level, **C*** Ci,Ci-Ciact cell with Nkd at its maximal level
Ectopic expression of Wg; loss-of-function of Slp	**En* En* En* En* En* En***	1.0	**En*** En cell expressing Wg transgene

Table 5.14: Reachability probabilities evatuated by Avatar for the segment polarity model (6 cells) by considering mutants. Priority classes 1: {Ciact, Cirep, Fz, Dsh, Pka}; 2: {Ci, Wg, Nkd, En, Slp, Hh, Ptc}, initial state = pair rule module. Avatar's parameters: runs=1E4, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E6.

Genetic background	Most probable attractors	Probability	Description
Wild-type	**T N Wg En C T** **T T T T T T** other attractors	[0.5786,0.5786] [0.2965, 0.2965] [0.1249, 0.1249]	**T** Trivial cell, **N** Nkd cell, **Wg** Wg cell, **En** En cell, **C** Ci,Ciact cell
Loss of function of Wg, En, Hh and Ci	**T T T T T T**	[1.0, 1.0]	**T** trivial cell
Loss of function of Ptc	**Wg Wg Wg En C C** **En Wg Wg En C C** other attractors	[0.410, 0.410] [0.381, 0.381] [0.209, 0.209]	**Wg** Wg cell, **En** En cell, **C** Ci,Ciact cell
Loss of function of Ptc and Hh	**Wg Wg Wg En* C C** **En* Wg Wg En* C C** other attractors	[0.409, 0.409] [0.379, 0.379] [0.212, 0.212]	**Wg** Wg cell, **En*** En cell without Hh expression, **C** Ci,Ci-act cell
Loss of function of Nkd	**T* A Wg* En* C* T*** **A A Wg* En C* T*** other attractors	[0.4821, 0.4821] [0.134, 0.134] [0.3839, 0.3839]	**T*** Trivial cell without Nkd expression, **A** previous Nkd cell that loses Nkd expression and gains medium level of Wg, **Wg*** Wg cell without Nkd expression, **C*** C cell without Nkd expression
Loss of function of Ptc, Wg	**C C C C C C**	[1.0, 1.0]	**C** Ci,Ciact cell
Loss of function of Ptc; Partial loss-of-function of Wg	**C Wg* Wg* C C C** **Wg* Wg* Wg* C C C** **C Wg* Wg* C C C** **C C Wg* C C C** other attractors	[0.6165, 0.6165] [0.1515, 0.1515] [0.0954, 0.0954] [0.0862, 0.0862] [0.0504, 0.0504]	**C** Ci,Ciact cell, **Wg*** Wg cell having medium level of Wg
Ectopic expression of En	**En* En* En* En* En* En***	[1.0, 1.0]	**En*** Cell expressing only En and Hh
Ectopic expression of Wg	**En* Wg Wg En* En* En*** **N* N* Wg En* En* En*** **En* Wg Wg En* En* Wg** **N* N* Wg En* En* Wg** other attractors	[0.206, 0.206] [0.197, 0.197] [0.191, 0.191] [0.197, 0.197] [0.209, 0.209]	**En*** En cell having an ctopic expression of Wg **N*** Nkd having an ectopic expression of Wg, **Wg** Wg cell
Ectopic expression of Nkd	**T* N Wg En* C* T*** **T* T* T* T* T* T*** other attractors	[0.5719, 0.5719] [0.3042, 0.3042] [0.1239, 0.1239]	**T*** Trivial cell with Nkd at its maximal level, **N** Nkd cell, **Wg** Wg cell, **En*** En cell with Nkd at its maximal level, **C*** Ci,Ciact cell with Nkd at its maximal level
Ectopic expression of Wg; loss-of-function of Slp	**En* En* En* En* En* En***	[1.0, 1.0]	**En*** En cell expressing Wg transgene

Table 5.15: Reachability probabilities evatuated by Monte Carlo for the segment polarity model (6 cells) by considering mutants. Priority classes 1: {Ciact, Cirep, Fz, Dsh, Pka}; 2: {Ci, Wg, Nkd, En, Slp, Hh, Ptc}, initial state = pair rule module. MonteCarlo's parameters: runs=1E4, maximum depth=1E6.

To solve this discrepancy, we followed the hypothesis discussed by Sánchez et al. in [2]. We considered a variant of the segment polarity model, where Wg signals can diffuse towards two neighboring cells in the absence of Nkd factor (see Figure 5.13).

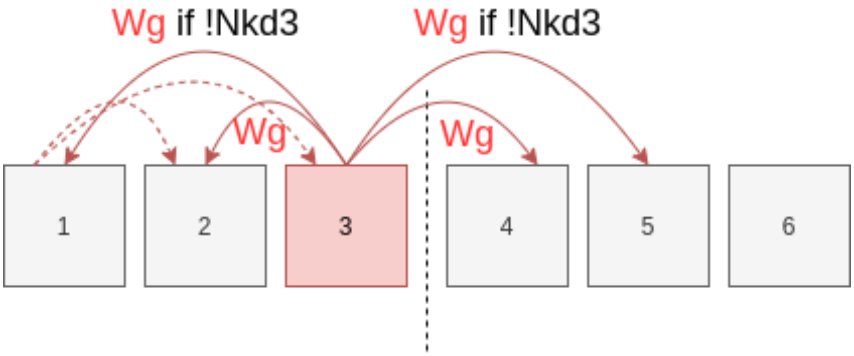


Figure 5.13: Inter-cellular interactions where Wg signals can diffuse towards two neighboring cells, with Nkd preventing this longer range diffusion.

With this model modification, the expected pattern arises as reachable attractor. By summing the probabilities of the attractors that match the expected pattern (that is a wider anterior Wg stripe, a wider En posterior stripe, and the appearance of Wg in the posterior region), we obtain the results presented in Table 5.16. Nevertheless, the probability of reaching such a pattern is relatively low even, when considering priority classes and non-uniform rates. This would suggest that the update setting needs to be refined.

Algorithm	Genetic backgrounds	Updating mode	Matching attractors	Probability	% of successful % of simulations	Time
Avatar	loss of function of Nkd	Asynchronous update. Uniform rates.	* * **Wg En Wg** * * * **Wg En En Wg** other attractors	0.1521 0.0243 0.8236	100	1779 s
		Asynchronous update. Rates: Ciact, Cirep, Fz, Dsh, Pka: 50.	* * **Wg En Wg** * other attractors	0.1851 0.8149	100	1746 s
		Priority update. Uniform rates.	* * **Wg En Wg** * other attractors	0.2293 0.7707	100	546 s
		Priority update. Rate for the increasing update:50; Rate for the decreasing update:1.	* * **Wg En Wg** * other attractors	0.2479 0.7321	100	520 s
Monte Carlo	loss of function of Nkd	Asynchronous update. Uniform rates.	* * **Wg En Wg** * * * **Wg En En Wg** other attractors	[0.1516, 0.1516] [0.0245, 0.0245] [0.8239, 0.8239]	100	9 s
		Asynchronous update. Rates: Ciact, Cirep, Fz, Dsh, Pka: 50.	* * **Wg En Wg** * * * **Wg En En Wg** other attractors	[0.1848, 0.1849] [0.0001, 0.0001] [0.8151, 8151]	100	6 s
		Priority update. Uniform rates.	* * **Wg En Wg** * other attractors	[0.2287, 0.2287] [0.7713, 0.7713]	100	6 s
		Priority update. Rate for the increasing update:50; Rate for the decreasing update:1.	* * **Wg En Wg** * other attractors	[0.2687, 0.2687] [0.7313, 0.7313]	100	6 s

Table 5.16: Reachability probabilities for the modified segment polarity model (6 cells). Initial state = pair rule module, priority classes 1:{Ciact, Cirep, Fz, Dsh, Pka}; 2:{Ci, Wg, Nkd, En, Slp, Hh, Ptc}. Avatar's parameters: runs=1E4, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E6. Monte Carlo's parameters: runs=1E4, maximum depth=1E6.

5.3.2 The T helper cells differentiation model

Our second case study is a published model of T helper cells differentiation model presented by Mendoza [3]. The immune system of vertebrates encompasses various type of cells. Among them, Cd4+ T

cells (a type of T lymphocytes) can be sub-classified as Th1 or Th2 cells, which have a common precursor Th0 [3]. These cells differ in their secretion pattern of cytokines (*i.e.,* protein that are secreted by the cell), and in their role in the cellular responses: the molecules secreted by Th1 cells lead to inflammatory immune responses (IFN-g), while those secreted by Th2 cells intervene in humoral immune responses (IL-4) [3]. Importantly, these cytokines secreted by Th cells promote the differentiation of these cells, and inhibit the proliferation of each other [3] (see Figure 5.14).

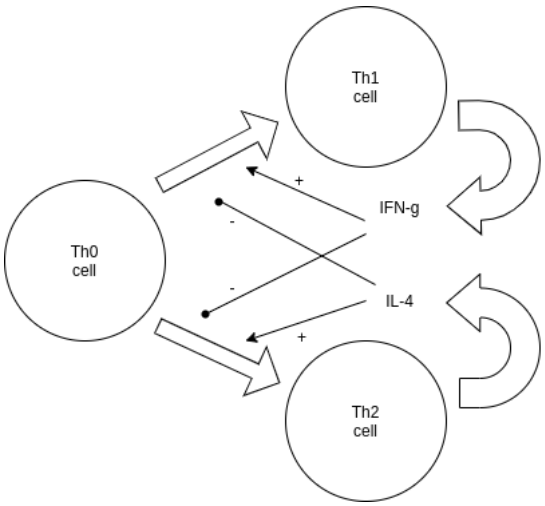


Figure 5.14: Differentiation of Th cells. Th0 cells differentiate into Th1 or Th2 cells, taken from [3]

The network defined by Mendoza encompasses 17 nodes, namely IFN-g, IFN-bR, IFN-b, IFN-gR, IL-4, IL-12, IL-18, IL-4R, IL-12R, IL-18R, STAT-1, STAT-6, STAT-4, IRAK, SOCS-1, GATA-3, and T-bet (see Figure 5.15) [3]. To briefly describe the regulatory network, the nodes of the network represent various kinds of molecules: secreted cytokines, receptors, signal transducers and transcription factors. IFN-g is a determinant of the Th1 phenotype, which acts on its target cells by binding to a receptor (IFN-gR) present in the Th1 cells themselves [3]. The transduction of the IFN-g/IFN-gR signal acts via STAT-1, which can be activated by IFN-g via IFN-gR [3]. T-bet is a transcription factor that activates the production of IFN-g, and its expression is up-regulated by IFN-g through a STAT-1-dependent mechanism [3]. To summarize, the circuit IFNg→IFN-gR→STAT1→Tbet creates a positive feedback loop that promotes the differentiation of Th1 cells. For the Th2 cells, IL-4 is an important cytokine associated with the Th2 phenotype. The transduction pathway that mediates IL-4 starts by the binding of IL-4 to its receptor (IL-4R) [3]. Then, the IL-4R signal is transduced by STAT-6, which activates GATA-3 [3]. GATA-3 itself is capable of inducing IL-4, thus establishing a positive feedback loop and promoting the differentiation of Th2 cells [3]. While IFN-g and IL-4 are two cytokines that promote, respectively, the differentiation of Th1 and Th2 cells, IL-12 is a molecule produced by monocytes and dendritic cells that, synergistically, act with IL-18 (a molecules produced by many cell types) to cause a significant secretion of IFN-g. [3]

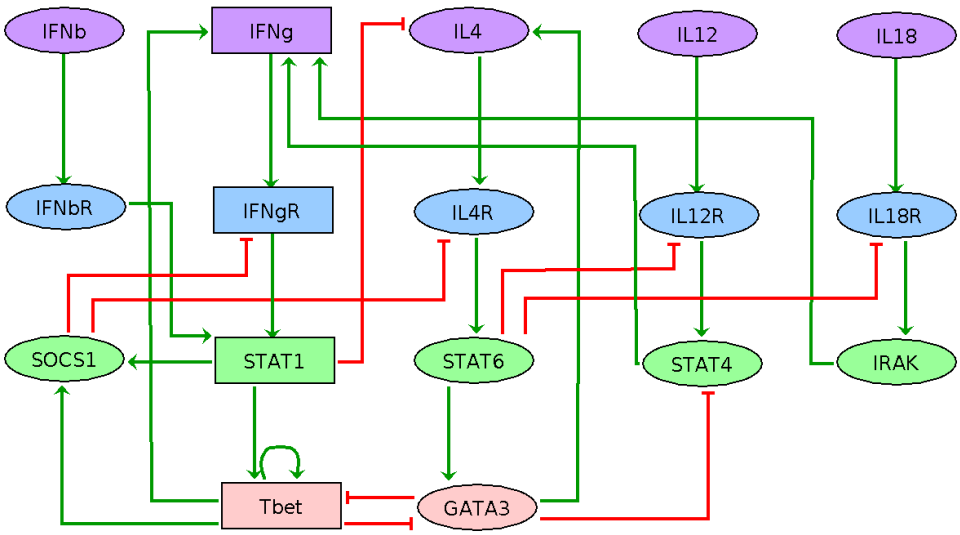


Figure 5.15: Logical regulatory graph of T helper differentiation network

In terms of dynamics, the state space of this model is composed by a total of $2^{13} * 3^4$ states and the four attractors reached have the following characteristics:

1. Inactivation of all the network nodes;
2. High expression of IFN-g, SOCS-1 and T-bet (reaching their maximum level), medium expression of IFN-gR, STAT-1, and all other nodes inactivated;
3. High expression of of SOCS-1, medium expression of IFN-g, IFN-gR, STAT-1, T-bet, and all other nodes inactivated;
4. High levels of IL-4, IL-4R, STAT-6, GATA-3, and all other nodes inactivated.

These four attractors have a clear biological interpretation. The first attractor corresponds to Th0 cells (**Th0**); the second (**Th1_m**) and third attractors (**Th1_h**) are states related with Th1 cells as they both have IFN-g activated. The difference between them lies in the activity level: **Th1_h** has a higher level of IFN-g, which leads to a higher level of its secretion. The fourth attractor represents the state of Th2 cells (**Th2**).

Note that the differentiation process itself can be represented by the transition of the system from one attractor to another. Hence, we can give the network a stimulus that would make the system transitioning from one basin of attraction to another [3], thus promoting the differentiation of Th0 cell to Th1 or Th2. For this purpose, we aimed to check the probability of reaching these attractors by considering the following conditions:

- Starting from the Th0 attractor, we give to the system a stimulus of IFN-g, IL-4 or a mixture of IL-12 and IL-18, *i.e.*, set the initial states as the state of Th0 plus the activation of these inputs;

- Furthermore, we can set a low rate for these cytokines (we considered that the rate for the update of these components is 0.2) and observe how the probabilities vary.

Algorithm	Updating mode	Initial states	Reached attractor	Probability	% of successful simulations	Time
Avatar	Asynchronous update Uniform rates	[**********] Sampling	**Th0** **Th1_m** **Th1_h** **Th2**	0.0913 0.4632 0.4056 0.0401	100	1 s
		Th0 + transient input IFN-g:1	**Th0** **Th1_m**	0.6873 0.3127	100	1 s
		Th0 + transient input IFN-g:2	**Th0** **Th1_m** **Th1_h**	0.4472 0.4812 0.0709	100	1 s
		Th0 + transient input IL-12:1	**Th0** **Th1_m**	0.7686 0.2314	100	1 s
		Th0 + transient input IL-18:1	**Th0**	1.0	100	1 s
		Th0 + transient inputs IL-12:1, IL-18:1	**Th0** **Th1_m** **Th1_h**	0.7452 0.2419 0.0129	100	1 s
		Th0 + transient input IL-4:1	**Th0** **Th2**	0.7443 0.2557	100	1 s
Monte Carlo	Asynchronous update Uniform rates	[**********] Sampling	**Th0** **Th1_m** **Th1_h** **Th2**	[0.0915, 0.0915] [0.4608, 0.4608] [0.409, 0.409] [0.0387, 0.0387]	100	1 s
		Th0 + transient input IFN-g:1	**Th0** **Th1_m**	[0.6907, 0.6907] [0.3093, 0.3093]	100	1 s
		Th0 + transient input IFN-g:2	**Th0** **Th1_m** **Th1_h**	[0.4373, 0.4373] [0.4874, 0.4874] [0.0753, 0.0753]	100	1 s
		Th0 + transient input IL-12:1	**Th0** **Th1_m**	[0.7679, 0.7679] [0.2321, 0.2321]	100	1 s
		Th0 + transient input IL-18:1	**Th0**	[1.0, 1.0]	100	1 s
		Th0 + transient inputs IL-12:1, IL-18:1	**Th0** **Th1_m** **Th1_h**	[0.7674, 0.7674] [0.2216, 0.2216] [0.011, 0.011]	100	1 s
		Th0 + transient input IL-4:1	**Th0** **Th2**	[0.7461, 0.7461] [0.2539, 0.2539]	100	1 s

Table 5.17: Reachability probabilities for the T helper differentiation model when considering transient inputs. Avatar parameters: runs=1E4, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E6. Monte Carlo's parameters: runs=1E4, maximum depth=1E6.

The results listed in Table 5.17 confirm the role of the cytokines described above, and we can observe that:

- A higher level of IFN-g can increase the probability of reaching the Th1 state (both Th1_m and Th1_h);
- IL-18 does not promote the differentiation of Th0, but, together with IL-12, it stimulates the differentiation into Th1 with high level of IFN-g.

Algorithm	Updating mode	Initial states	Reached attractor	Probability	% of successful simulations	Time
Avatar	Asynchronous update Rates of inputs: 0.2 Rates of non-inputs components: 1	[**********] Sampling	**Th0** **Th1_m** **Th1_h** **Th2**	0.0928 0.4712 0.404 0.032	100	1 s
		Th0 + transient input IFN-g:1	**Th0** **Th1_m**	0.282 0.718	100	1 s
		Th0 + transient input IFN-g:2	**Th0** **Th1_m** **Th1_h**	0.0711 0.6692 0.2597	100	1 s
		Th0 + transient input IL-12:1	**Th0** **Th1_m**	0.5613 0.4387	100	1 s
		Th0 + transient input IL-18:1	**Th0**	1.0	100	1 s
		Th0 + transient inputs IL-12:1, IL-18:1	**Th0** **Th1_m** **Th1_h**	0.5575 0.2419 0.0255	100	1 s
		Th0 + transient input IL-4:1	**Th0** **Th2**	0.3807 0.6193	100	1 s
Monte Carlo	Asynchronous update Rates of inputs: 0.2 Rates of non-inputs components: 1	[**********] Sampling	**Th0** **Th1_m** **Th1_h** **Th2**	[0.0805, 0.0505] [0.4772, 0.5072] [0.4108, 0.4108] [0.0315, 0.0315]	100	1 s
		Th0 + transient input IFN-g:1	**Th0** **Th1_m**	[0.2807, 0.2807] [0.7193, 0.7193]	100	1 s
		Th0 + transient input IFN-g:2	**Th0** **Th1_m** **Th1_h**	[0.068, 0.068] [0.6642, 0.6642] [0.2678, 0.2678]	100	1 s
		Th0 + transient input IL-12:1	**Th0** **Th1_m**	[0.5545, 0.5545] [0.4455, 0.4455]	100	1 s
		Th0 + transient input IL-18:1	**Th0**	[1.0, 1.0]	100	1 s
		Th0 + transient inputs IL-12:1, IL-18:1	**Th0** **Th1_m** **Th1_h**	[0.5553, 0.5553] [0.4181, 0.4181] [0.0266, 0.0266]	100	1 s
		Th0 + transient input IL-4:1	**Th0** **Th2**	[0.3713, 0.3713] [0.6287, 0.6287]	100	1 s

Table 5.18: Reachability probabilities for the T helper differentiation model when considering transient inputs and non-uniform transition rates. Avatar parameters: runs=1E4, expansion limit=1E4, rewiring limit=1E3, tau=3, minimum SCC to rewire=4, maximum depth=1E6. Monte Carlo's parameters: runs=1E4, maximum depth=1E6.

When considering a low rate for the updating of these molecules, we can observe that the probability of reaching the Th0 state has dropped significantly, while the probabilities of all other attractors have increased notably (see Table 5.18).

Finally, the results listed in Tables 5.19 and 5.20 present the reachability results when applying perturbations to the model. We can observe that, when fixing the value of these cytokines, the model dynamics differ notably from a wild type. Furthermore, by analysing these results, we were able to get some insights into the network:

- It is expected to reach the same attractors with a loss of function of IFN-g or IFN-gR (since they belong to the same feedback loop). However, the attractors of these mutants differ in the level of activation of IFN-g (which is observed biologically [3]);
- With ectopic expression of IFN-g, the system leads to Th1 state with high level of IFN-g. Whereas, when fixing the value of IFN-g to 1, the attractors reached correspond to Th1 state with medium expression of IFN-g, which is coherent to the dynamic role that this molecules plays in the differentiation process;
- In the case of ectopic expression of IL-18, the system does not behave differently from the wild-type as the four reached attractors are similar to those observed in a wild-type (with the activation of IL-18, IL-18R, and IRAK, as shown by Table 5.19);
- IL-18 is a potent inducer in combination with IL-12 [3]. Indeed, when having both IL-12, IL-18 over-expressed, the system leads to the loss the Th0 attractor and both reached Th1 states have IFN-g highly expressed;
- With ectopic expression of GATA-3, the system has a high probability of reaching the Th2 state which is coherent to the fact that GATA-3 is a transcription factor that induces IL-4.

Genetic background	Reached attractors	Probability
Wild-type	**Th0**: Th0 state	0.0913
	Th1_m: Th1 state with medium level of IFN-g	0.4632
	Th1_h: Th1 state with high level of IFN-g	0.4056
	Th2: Th2 state	0.0401
Loss of function of IFN-g	**Th0**: Th0 state	0.1126
	Th0*: Th0 state with medium expression of T-bet and high expression of SOCS-1	0.4401
	Th0*: Th0 state with high expression of SOCS-1 and T-bet	0.3991
	Th2: Th2 state	0.0482
Loss of function of IFN-gR	**Th0:** Th0 state	0.1412
	Th1_m*: Th1_m state without the expression of STAT-1 and IFN-gR.	0.4085
	Th1_h*: Th1_h state without the expression of STAT-1 and IFN-gR	0.3811
	Th2: Th2 state	0.0692
Restriction of activity of IFN-g to 1	**Th1_m** Th1_m state	0.5949
	Th1_m* Th1_m state with high level of T-bet	0.4051
Ectopic expression of IFN-g	**Th1_h***: Th1_m state with medium expression of T-bet	0.5509
	Th1_h: Th1_h state	0.4491
Ectopic expression of IL-12	**Th2***: Th2 state with IL-12 activated	0.0355
	Th1_m*: Th1_m state with IL-12, IL-12R and STAT-4 activated	0.5568
	Th1_h*: Th1_h state with IL-12, IL-12R and STAT-4 activated	0.4077
Ectopic expression of IL-18	**Th0**: Th0 state	0.0946
	Th1_m*: Th1_m state with IL-18, IL-18R and IRAK activated	0.4647
	Th1_h*: Th1_h state with IL-18, IL-18R and IRAK activated	0.3987
	Th2: Th2 state with IL-18 activated	0.042
Ectopic expression of IL-12 and IL-18	**Th0**: Th0 state with IL-12 and IL-18 activated	0.035
	Th1_h*: Th1_h state with IL-18, IL-18R, IRAK, IL-12, IL-12R activated and medium expression of T-bet.	0.5274
	Th1_h*: Th1_h state with IL-18, IL-18R, IRAK, IL-12, IL-12R activated and high expression of T-bet	0.4376
Ectopic expression of GATA-3	**Th1_m***: Th1_m state with GATA-3 activated	0.2636
	Th1_h*: Th1_h state with GATA-3 activated	0.1923
	Th2: Th2 state	0.5441

Table 5.19: Reachability probabilities evaluated by Avatar for the T helper cells differentiation model by considering mutants. initial state = whole state space, Avatar's parameters: runs=1E4, expansion limit = 1E4, rewiring limit = 1E3, maximum depth=1E6.

Genetic background	Reached attractors	Probability
Wild-type	**Th0**: Th0 state	[0.0915, 0.0915]
	Th1_m: Th1 state with medium level of IFN-g	[0.4608, 0.4608]
	Th1_h: Th1 state with high level of IFN-g	[0.409, 0.409]
	Th2: Th2 state	[0.0387, 0.387]
Loss of function of IFN-g	**Th0**: Th0 state	[0.1133, 0.1133]
	Th0*: Th0 state with medium expression of T-bet and high expression of SOCS-1	[0.4392, 0.4392]
	Th0*: Th0 state with high expression of SOCS-1 and T-bet	[0.3983, 0.3983]
	Th2: Th2 state	[0.0492, 0.0492]
Loss of function of IFN-gR	**Th0:** Th0 state	[0.1367, 0.1367]
	Th1_m*: Th1_m state without the expression of STAT-1 and IFN-gR.	[0.412, 0.412]
	Th1_h*: Th1_h state without the expression of STAT-1 and IFN-gR	[0.3795, 0.3795]
	Th2: Th2 state	[0.0718, 0.0718]
Restriction of activity of IFN-g to 1	**Th1_m** Th1_m state	[0.6082, 0.6082]
	Th1_m* Th1_m state with high level of T-bet	[0.3918, 0.3918]
Ectopic expression of IFN-g	**Th1_h***: Th1_m state with medium expression of T-bet	[0.5559, 0.5559] [0.4441]
	Th1_h: Th1_h state	
Ectopic expression of IL-12	**Th2***: Th2 state with IL-12 activated	[0.0355, 0.0355]
	Th1_m*: Th1_m state with IL-12, IL-12R and STAT-4 activated	[0.5561, 0.5561]
	Th1_h*: Th1_h state with IL-12, IL-12R and STAT-4 activated	[0.4084, 0.4084]
Ectopic expression of IL-18	**Th0**: Th0 state	[0.0949, 0.0949]
	Th1_m*: Th1_m state with IL-18, IL-18R and IRAK activated	[0.4556, 0.4556]
	Th1_h*: Th1_h state with IL-18, IL-18R and IRAK activated	[0.4113, 0.4113]
	Th2: Th2 state with IL-18 activated	[0.0382, 0.0382]
Ectopic expression of IL-12 and IL-18	**Th0**: Th0 state with IL-12 and IL-18 activated	[0.0353, 0.0353]
	Th1_h*: Th1_h state with IL-18, IL-18R, IRAK, IL-12, IL-12R activated and medium expression of T-bet	[0.526, 0.526]
	Th1_h*: Th1_h state with IL-18, IL-18R, IRAK, IL-12, IL-12R activated and high expression of T-bet	[0.4387, 0.4387]
Ectopic expression of GATA-3	**Th1_m***: Th1_m state with GATA-3 activated	[0.2665, 0.2665]
	Th1_h*: Th1_h state with GATA-3 activated	[0.1971, 0.1971]
	Th2: Th2 state	[0.5364, 0.5364]

Table 5.20: Reachability probabilities evaluated by Monte Carlo for the T helper cells differentiation model by considering mutants. initial state = whole state space, Avatar's parameters: runs=1E4, expansion limit = 1E4, rewiring limit = 1E3, maximum depth=1E6.

6

Conclusion

Logical modelling has been proposed to capture salient dynamical properties models of regulatory networks, and has proved well suited for this intent. We are particularly interested in identifying model attractors and to quantify their reachability.

The dynamics of a model can be represented by a State Transition Graph (STG). To identify the attractors of a logical model, one can compute its STG and directly analyse it. However, due to the combinatorial explosion of the number of states, this procedure becomes impractical for large models. To address this issue, Mendes *et al.* [1] proposed some tools (Avatar, Firefront and Monte Carlo) to surpass efficiency bottlenecks and solve the problem of quantification of attractor reachability. Nevertheless, Avatar and Monte Carlo were limited to solely support the asynchronous update scheme, as probabilities of concurrent transitions were defined as uniformly distributed. To overcome these limitations, we extended both Avatar and Monte Carlo algorithms to support priority classes and non-uniform probabilities.

At first, we refactored the original three algorithms, and made the code migration from GINsim to bioLQM in order to increase their interoperability and extensibility. These algorithms can now be used by calling the corresponding service of bioLQM, and, in the future, this could be easily extended to support other updating modes. In addition, we integrated priority classes by considering the priority updater existing in bioLQM [6]. Moreover, the transition probabilities, initially uniform, are now calculated according to the qualitative rate associated to each component update. Furthermore, when considering non-uniform transition probabilities in Avatar, we redefined the method of calculating the probabilities of transitions exiting a cycle to the exiting states of the cycle. In particular, in the second strategy of rewiring procedure, the probabilities of exiting a rewired cycle are calculated according to the number of transition from each state of the cycle to exit states, and to the rates associated to these transitions.

The results obtained from the JUnit test cases and the synthetic models have shown the correctness of the revised algorithms and their ability to identify efficiently the attractors and assess their reachability. Monte Carlo has an excellent performance in the absence of large transient SCCs and its use is recommended when the attractors are known to be all stable states. Avatar supports the identification of complex attractors and rewires transient SCCs, as a consequence, its performance is highly dependent on the STG structure and the chosen parameters. When considering biological models, the integration of priority classes and non-uniform transition probabilities permits further analyses of the model properties. For example, by taking into account biological knowledge related to the kinetic of the regulatory processes, we were able to maximize the reachability probabilities of the wild type state of the segment polarity model [2], thus, reducing the probability of reaching attractors not biologically observed.

To promote the accessibility of the tools developed (as well as the reproducibility of our results), we started to integrate our tools in CoLoMoTo Interactive Notebook [31] by developing the corresponding Python programming interface. While this work is still incomplete, we aim to finish the implementation

as soon as the official version of bioLQM integrates our tools.

As a future work, other updating modes could be proposed and integrated in the quantification reachability algorithms. In particular, the current priority settings only account for sole asynchronous updating for each class, we could refine these settings to classes with different types of updating modes, such as the synchronous updating scheme and the complete updating scheme [6].

Bibliography

[1] N. D. Mendes, R. Henriques, E. Remy, J. Carneiro, P. T. Monteiro, and C. Chaouiya, "Estimating attractor reachability in asynchronous logical models," *Frontiers in Physiology*, vol. 9, Sep. 2018. [Online]. Available: https://doi.org/10.3389/fphys.2018.01161

[2] L. Sanchez, C. Chaouiya, and D. Thieffry, "Segmenting the fly embryo: logical analysis of the role of the segment polarity cross-regulatory module," *The International Journal of Developmental Biology*, vol. 52, no. 8, pp. 1059–1075, 2008. [Online]. Available: https://doi.org/10.1387/ijdb.072439ls

[3] L. Mendoza, "A network model for the control of the differentiation process in th cells," *Biosystems*, vol. 84, no. 2, pp. 101–114, May 2006. [Online]. Available: https://doi.org/10.1016/j.biosystems.2005.10.004

[4] H. de Jong, "Modeling and simulation of genetic regulatory systems: A literature review," *Journal of Computational Biology*, vol. 9, no. 1, pp. 67–103, Jan. 2002. [Online]. Available: https://doi.org/10.1089/10665270252833208

[5] N. L. Novère, "Quantitative and logic modelling of molecular and gene networks," *Nature Reviews Genetics*, vol. 16, no. 3, pp. 146–158, Feb. 2015. [Online]. Available: https://doi.org/10.1038/nrg3885

[6] A. Naldi, "BioLQM: A java toolkit for the manipulation and conversion of logical qualitative models of biological networks," *Frontiers in Physiology*, vol. 9, Nov. 2018. [Online]. Available: https://doi.org/10.3389/fphys.2018.01605

[7] A. Naldi, D. Thieffry, and C. Chaouiya, "Decision diagrams for the representation and analysis of logical models of genetic networks," in *Computational Methods in Systems Biology*. Springer Berlin Heidelberg, 2007, pp. 233–247. [Online]. Available: https://doi.org/10.1007/978-3-540-75140-3_16

[8] W. Abou-Jaoudé, D. A. Ouattara, and M. Kaufman, "From structure to dynamics: Frequency tuning in the p53–mdm2 network," *Journal of Theoretical Biology*, vol. 258, no. 4, pp. 561–577, Jun. 2009. [Online]. Available: https://doi.org/10.1016/j.jtbi.2009.02.005

[9] W. Abou-Jaoudé, P. Traynard, P. T. Monteiro, J. Saez-Rodriguez, T. Helikar, D. Thieffry, and C. Chaouiya, "Logical modeling and dynamical analysis of cellular networks," *Frontiers in Genetics*, vol. 7, May 2016. [Online]. Available: https://doi.org/10.3389/fgene.2016.00094

[10] D. Bérenguier, C. Chaouiya, P. T. Monteiro, A. Naldi, E. Remy, D. Thieffry, and L. Tichit, "Dynamical modeling and analysis of large cellular regulatory networks," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 23, no. 2, p. 025114, Jun. 2013. [Online]. Available: https://doi.org/10.1063/1.4809783

[11] A. Faure, A. Naldi, C. Chaouiya, and D. Thieffry, "Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle," *Bioinformatics*, vol. 22, no. 14, pp. e124–e131, Jul. 2006. [Online]. Available: https://doi.org/10.1093/bioinformatics/btl210

[12] A. Naldi, J. Carneiro, C. Chaouiya, and D. Thieffry, "Diversity and plasticity of th cell types predicted from regulatory network modelling," *PLoS Computational Biology*, vol. 6, no. 9, p. e1000912, Sep. 2010. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1000912

[13] W. Abou-Jaoudé, P. T. Monteiro, A. Naldi, M. Grandclaudon, V. Soumelis, C. Chaouiya, and D. Thieffry, "Model checking to assess t-helper cell plasticity," *Frontiers in Bioengineering and Biotechnology*, vol. 2, Jan. 2015. [Online]. Available: https://doi.org/10.3389/fbioe.2014.00086

[14] G. Stoll, E. Viara, E. Barillot, and L. Calzone, "Continuous time boolean modeling for biological signaling: application of gillespie algorithm," *BMC Systems Biology*, vol. 6, no. 1, p. 116, 2012. [Online]. Available: https://doi.org/10.1186/1752-0509-6-116

[15] P. Brémaud, "Discrete-time markov models," in *Texts in Applied Mathematics*. Springer New York, 1999, pp. 53–93. [Online]. Available: https://doi.org/10.1007/978-1-4757-3124-8_2

[16] C. M. Grinstead and J. L. Snell, *Introduction to Probability*. AMS, 2003. [Online]. Available: http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.html

[17] R. Tarjan, "Depth-first search and linear graph algorithms," in *12th Annual Symposium on Switching and Automata Theory (swat 1971)*. IEEE, Oct. 1971. [Online]. Available: https://doi.org/10.1109/swat.1971.10

[18] E. Dubrova and M. Teslenko, "A SAT-based algorithm for finding attractors in synchronous boolean networks," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 8, no. 5, pp. 1393–1399, Sep. 2011. [Online]. Available: https://doi.org/10.1109/tcbb.2010.20

[19] A. Garg, I. Xenarios, L. Mendoza, and G. DeMicheli, "An efficient method for dynamic analysis of gene regulatory networks and in silico gene perturbation experiments," in *Lecture*

Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 62–76. [Online]. Available: https://doi.org/10.1007/978-3-540-71681-5_5

[20] E. M. Clarke, T. A. Henzinger, and H. Veith, "Introduction to model checking," in *Handbook of Model Checking*. Springer International Publishing, 2018, pp. 1–26. [Online]. Available: https://doi.org/10.1007/978-3-319-10575-8_1

[21] S. Busard and C. Pecheur, "Rich counter-examples for temporal-epistemic logic model checking," *Electronic Proceedings in Theoretical Computer Science*, vol. 78, p. 39–53, Feb 2012. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.78.4

[22] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NUSMV: a new symbolic model checker," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 2, no. 4, pp. 410–425, Mar. 2000. [Online]. Available: https://doi.org/10.1007/s100090050046

[23] A. Lomuscio, C. Pecheur, and F. Raimondi, "Automatic verification of knowledge and time with nusmv," in *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*. IJCAI/AAAI Press, 2007, pp. 1384–1389.

[24] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Computer Aided Verification*. Springer Berlin Heidelberg, 2011, pp. 585–591. [Online]. Available: https://doi.org/10.1007/978-3-642-22110-1_47

[25] A. Naldi, C. Hernandez, W. Abou-Jaoudé, P. T. Monteiro, C. Chaouiya, and D. Thieffry, "Logical modeling and analysis of cellular regulatory networks with GINsim 3.0," *Frontiers in Physiology*, vol. 9, Jun. 2018. [Online]. Available: https://doi.org/10.3389/fphys.2018.00646

[26] P. T. Monteiro and C. Chaouiya, "Efficient verification for logical models of regulatory networks," in *6th International Conference on Practical Applications of Computational Biology & Bioinformatics*. Springer, 2012, pp. 259–267. [Online]. Available: https://doi.org/10.1007/978-3-642-28839-5_30

[27] G. A. Bird, "Monte-Carlo simulation in an engineering context," *Progress in Astronautics and Aeronautics*, vol. 74, pp. 239–255, Jan. 1981. [Online]. Available: https://ui.adsabs.harvard.edu/abs/1981PrAA...74..239B

[28] D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," *Journal of Computational Physics*, vol. 22, no. 4, pp. 403–434, Dec. 1976. [Online]. Available: https://doi.org/10.1016/0021-9991(76)90041-3

[29] C. Chaouiya, D. Bérenguier, S. M. Keating, A. Naldi, M. P. van Iersel, N. Rodriguez, A. Dräger, F. Büchel, T. Cokelaer, B. Kowal, B. Wicks, E. Gonçalves, J. Dorier, M. Page, P. T. Monteiro, A. von

Kamp, I. Xenarios, H. de Jong, M. Hucka, S. Klamt, D. Thieffry, N. L. Novère, J. Saez-Rodriguez, and T. Helikar, “SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools,” *BMC Systems Biology*, vol. 7, no. 1, p. 135, 2013. [Online]. Available: https://doi.org/10.1186/1752-0509-7-135

[30] A. Naldi, P. T. Monteiro, C. Mussel, H. A. Kestler, D. Thieffry, I. Xenarios, J. Saez-Rodriguez, T. Helikar, and C. C. and, “Cooperative development of logical modelling standards and tools with CoLoMoTo,” *Bioinformatics*, vol. 31, no. 7, pp. 1154–1159, Jan. 2015. [Online]. Available: https://doi.org/10.1093/bioinformatics/btv013

[31] A. Naldi, C. Hernandez, N. Levy, G. Stoll, P. T. Monteiro, C. Chaouiya, T. Helikar, A. Zinovyev, L. Calzone, S. Cohen-Boulakia, D. Thieffry, and L. Paulevé, “The CoLoMoTo interactive notebook: Accessible and reproducible computational analyses for qualitative biological networks,” *Frontiers in Physiology*, vol. 9, Jun. 2018. [Online]. Available: https://doi.org/10.3389/fphys.2018.00680

A

Manual of bioLQM

Here we present a small user manual of the modified bioLQM proposed in this thesis, through the use of command-line arguments [6].

A.1 Overview

By running the executable through the command `java -jar bioLQM.jar`, bioLQM will provide a scripting interface to illustrate all the instructions to manipulate a logical model. As illustrated in Figure A.1, it is mandatory to provide a model file when intent to run the bioLQM tool. The extensive list of model file supported by bioLQM is listed in Figure A.2.

```
| Usage:
--------------------------------------------------------------------------------
# Convert a single file:
java -jar bioLQM.jar [-if informat] infile [-m modifier [options]] [-of outformat] outfile

# Run a tool on an imported model:
java -jar bioLQM.jar [-if informat] infile [-m modifier [options]] -r tool [options]

# Run a script:
java -jar bioLQM.jar -s script.js [arguments...]
```

Figure A.1: Usage of bioLQM

```
| Available formats:
|   '<'/'>': import / export  ; 'b'/'B'/'M' Boolean/Booleanized/Multivalued
--------------------------------------------------------------------------------
B <>  bnet              bnet functions format
B <>  booleannet        Alternative functions format
B <>  boolsim           boolsim format
B <>  cnet              cnet functions format
B <>  boolfunctions     Raw functions format
M  >  ginml             GINML format
M  >  gna               GNA (non-xml) format
M <>  itnet             Implicant Table format
B  >  bnd               MaBoSS format
M <>  mnet              Multi-valued functions format
b  >  apnn              APNN format
b  >  ina               INA Petri net format
b  >  pnml              PNML format
M  >  an                Pint format
M <>  sbml              SBML-qual v1.0 format
M <>  tt                Truth table format
```

Figure A.2: Formats supported by bioLQM

A user may need to apply a modifier to the model before running a specific tool. To apply a modifier, the corresponding argument is `-m` followed by the modifier in question. Example: `java -jar bioLQM.jar model_file -m modifier`. The list of the available modifiers in bioLQM is listed in Figure A.3.

```
| Available modifiers:
--------------------------------------------------------------------------------
booleanize              model booleanizer                   (no parameters)
buffer                  add buffer components to a model    :buffer | :delay | source:target1,target2
perturbation            model perturbation                  Example: Node1%0 Node2%1
reduce                  model reduction                     :fixed | :duplicate | :output
reverse                 model reverser                      (no parameters)
sanitize                sanity cleanups                     :name2id
submodel                extract sub-model                   list of component IDs
restrict                Restrict a model into a subspace    pattern, i.e. 00----1--0-1-
```

Figure A.3: Modifiers available in bioLQM

To run a specific tool, the corresponding argument is `-r` followed by the desired designation. Note that, when manipulating a modified model, it is necessary to modify it before executing a tool, *i.e.*, the argument `-m modifier` should never be specified after the argument `-r tool`. The list of the available

tools in bioLQM is presented in Figure A.4.

```
| Available tools:
--------------------------------------------------------------------------------
M fixpoints     Search fixed (stable) states
M avatar        Estimate attractor reachability using Avatar
M firefront     Estimate attractor reachability using Firefront
M montecarlo    Estimate attractor reachability using Monte Carlo
M trace Compute deterministic trace
M random        Perform a random walk in the dynamics
M trapspaces    Search trap spaces using ASP or BDDs
```

Figure A.4: Tools available in bioLQM

More examples of usage of bioLQM can be seen in Figure A.5.

```
| Examples:
--------------------------------------------------------------------------------
java -jar bioLQM.jar infile.sbml outfile.ginml
java -jar bioLQM.jar infile.sbml -m booleanize outfile.sbml
java -jar bioLQM.jar -if sbml infile.xml -of ina file.txt
java -jar bioLQM.jar -if sbml file1.in...filen.in -of ginml /path/to/outfolder/
java -jar bioLQM.jar infile.sbml -r stable
```

Figure A.5: Example of usage of bioLQM

To define initial conditions, a user can specify a pattern by passing the corresponding argument `-i pattern`. It is possible to set multiple patterns as initial conditions by separating them with a `"-"` (example: `-i pattern1-pattern2`). Whenever we would like to perform simulations using Avatar or Monte Carlo, bioLQM will choose a random state from the union of the specified patterns and set it as an initial state at each given iteration. Note that this argument can be also used in the following tools: `random walk` and `compute deterministic trace`.

To facilitate its use and improve the readability of the results obtained, it is possible to provide to the system a file, which contains the information regarding a phenotype (a named (set of) pattern(s)), through the argument `"-ns file_name"`. As benefits, users can directly specify the name of a phenotype, when defining the initial conditions (example: `"-i phenotype"` or `"-i" phenotype1-phenotype2"` when setting multiple patterns as initial conditions). Moreover, when retrieving simulation results, the system will automatically parse the reached pattern to identify the corresponding phenotype. Table A.1 shows an example.

T	0000*****121
C	0000****1***
Wg	21112*******
En	*****11*****
N	0***200*****

Table A.1: Phenotypes and the corresponding patterns

A.2 Input parameters of algorithms

Since Avatar and Monte Carlo have their own extensive list of parameters, we provided scripting pages to show which parameters are supported by each algorithm. For Avatar, it is required to always specify the number of runs, rewiring and expansion limits (see Figure A.6), while, for Monte Carlo, only the number of runs needs to be specified (see Figure A.7).

```
yu@yu-Legion-5-15ACH6H:~/git/bioLQM1/target$ java -jar bioLQM-0.8.jar ex_cycle.sbml -r avatar
Supported parameters:
--runs (MANDATORY)
--expansionLimit (MANDATORY)
--rewiringLimit (MANDATORY)
--maxDepth (DEFAULT=1000000)
--minTransientSize (DEFAULT=200)
--minStatesRewiring (DEFAULT=4)
--tau (DEFAULT=3)
--strategy [MatrixInversion/RandomExit] (default=MatrixInversion)
--notKeepTransients (default=keepingTransients)
--verbose (default=false)
--rates [rates separated by ,] (G1:Rate,G2[+]:Rate)
--priorityClass [classes separated by /] (G1/G2:G3/G4)
```

Figure A.6: Available parameters of Avatar

```
yu@yu-Legion-5-15ACH6H:~/git/bioLQM1/target$ java -jar bioLQM-0.8.jar ex_cycle.sbml -r montecarlo
Supported parameters:
--runs (MANDATORY)
--maxDepth (DEFAULT=1000)
--verbose (default=false)
--rates [rates separated by ,] (G1:Rate,G2[+]:Rate)
--priorityClass [classes separated by /] (G1/G2:G3/G4)
```

Figure A.7: Available parameters of Monte Carlo

Furthermore, since the format of input string of some of these parameters might not be intuitive, we provided the following information to ease its use:

- For the parameter `--priorityClasses`, nodes in the same priority are separated by a "/" , and classes are separated by a ":", where the class with the highest priority is the first in the string. Example: `java -jar bioLQM.jar model.sbml -r avatar --priorityClasses G1[+]:G2/G3/G4/G1[-]` (the synthesis of G1 is in the higher class, while G2, G3, G4 and the degration of G1 are in the lower class).

- For the parameter `--rates`, to assign a value to the rate of a node, we have a pair ”node:rate”, where ”node” is the name of the node and ”rate” is a floating number. The pairs are separated by a comma ”,”. The rates of nodes not explicitly specified are, initially assigned to value 1.
  Example: `java -jar bioLQM.jar model.sbml -r avatar --rates G1[+]:5,G3[-]:3` (the rate of synthesis of G1 is 5, the rate of degradation of G3 is 3 and the rest of the rates are 1).