



**Daniel
Oliveira Figueiredo**

**Fundações Lógicas e Ferramentas Computacionais
para a Biologia Sintética**

**Logic foundations and computational tools for
synthetic biology**



**Daniel
Oliveira Figueiredo**

Fundações Lógicas e Ferramentas Computacionais para a Biologia Sintética

Logical foundations and computational tools for synthetic biology

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Matemática Aplicada, realizada sob a orientação científica de Manuel António Gonçalves Martins, Professor do Departamento de Matemática da Universidade de Aveiro e co-orientação de Luís Soares Barbosa, Professor do Departamento de Ciências da Computação da Universidade do Minho

The work in this thesis was supported by FCT via the PhD scholarship PD/BD/114186/2016. It was also supported by ERDF - The European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project POCI-01-0145-FEDER-030947 (KLEE). Complementary support was provided by the France-Portugal partnership PHC PESSOA 2018 between M. Chaves (Campus France #40823SD) and M. A. Martins.

o júri / the jury

presidente / president

Doutora Silvina Maria Vagos Santana

Professora Catedrática da Universidade de Aveiro
(por delegação da Reitora da Universidade de Aveiro)

vogais / examiners committee

Doutor Manuel António Gonçalves Martins

Professor associado da Universidade de Aveiro (orientador)

Doutor Hidde de Jong

Investigador sénior do INRIA Grenoble - Rhne-Alpes

Doutor Pedro Tiago Gonçalves Monteiro

Professor auxiliar da Universidade de Lisboa

Doutor Delfim Fernando Marado Torres

Professor Catedrático da Universidade de Aveiro

Doutora Madalena Cabral Ferreira Chaves

Investigadora sénior do INRIA Sophia Antipolis

Doutor Sérgio Roseiro Teles Marcelino

Investigador da Universidade de Lisboa

Doutor Manuel Augusto Fernandes Delgado

Professor auxiliar da Universidade do Porto

agradecimentos / acknowledgements

Em primeiro lugar, agradeço a Deus por ter estado sempre comigo e me proporcionar tantas experiências fantásticas na vida. Agradeço também aos meus pais, irmãos e, especialmente, à minha mulher por me terem sempre ajudado e apoiado ao longo destes anos.

Agradeço à Associação Musical e Cultural São Bernardo por ter sido uma segunda família que sempre me acolheu e que me proporcionou momentos de descontração e emoção a fazer música, a minha segunda paixão. Também uma palavra ao grupo do CLU de Aveiro, que nos primeiros anos foram uma grande força na minha vida, à comunidade de Pokémon Go de Aveiro e a todos os amigos e colegas do departamento de Matemática.

Um merecido agradecimento a todos os professores que acompanharam o meu percurso ao longo destes anos e, em especial, ao meu orientador Manual António Martins e co-orientador Luís Soares Barbosa que foram sempre presentes e construtivamente críticos durante todo o período do doutoramento. Agradeço ainda à Diana Costa, minha “irmã académica” por toda a ajuda que me deu.

Finalmente, agradeço à FCT o financiamento desta tese e à Universidade de Aveiro por ter sido a minha casa ao longo destes anos.

Palavras-chave

Redes regulatórias biológicas, Modelos lineares por partes, Redes Booleanas reactivas, Grafo assintótico extendido, Bissimulação, Reconfigurabilidade, Reactividade, rPrism, Switch graphs com pesos.

Resumo

O estudo e desenvolvimento de ferramentas para sistemas computacionais é uma área onde facilmente podemos encontrar vários trabalhos, sendo hoje em dia um dos tópicos dominantes na investigação em Ciências da Computação, permitindo, a esta área, o acesso a uma vasta base teórica, além de diversos algoritmos e ferramentas tais como model checkers.

Esta tese centra-se na ideia de que um sistema biológico pode ser visto, em certa forma, como um sistema computacional, onde células e genes substituem o papel dos transístors. De facto, a noção de computação é, muitas vezes, associada ao funcionamento do cérebro de seres vivos. Tendo em conta este ponto de vista, o objetivo deste trabalho é revisitar alguns conceitos básicos no estudo de dinâmicas intracelulares, comuns a todos os seres vivos, de um ponto de vista computacional, de forma a averiguar como podemos aplicar a estes sistemas os conceitos, algoritmos e ferramentas computacionais usadas na área da Informática. Em particular, começamos por revisitar vários tipos de modelos usados para descrever a dinâmica intracelular de seres vivos – modelos lineares por partes e redes Booleanas.

De seguida, propomos uma nova perspetiva sobre os modelos lineares por partes, considerando estes modelos como reconfiguráveis. Isto permite-nos o uso de ferramentas computacionais como o KeYmaera e dReach. Por outro lado, discretizando este modelos mas mantendo a noção de reconfigurabilidade, obtemos a noção de rede Booleana reactiva, baseada no formalismo de *switch graphs*, e propomos uma linguagem lógica para expressar e verificar propriedades destes sistemas bem como uma noção de bissimulação. No que diz respeito a modelos Booleanos, apresentamos uma nova visão sobre a noção de “terminal” (ou atrator) de forma a relacioná-lo com a noção de bisimulação, muito usada em computação. De seguida, focamos a atenção no método de obtenção do gráfico assintótico e, após um estudo profundo, propomos um método intermédio que, apesar de menos eficiente a nível computacional, se mostra mais adequado ao contexto. Finalmente, consideramos um novo tipo de modelo estocástico ao incorporar pesos na arestas dos *switch graphs* e desenvolvemos uma extensão do PRISM *model checker* – rPrism – para estudar esta classe específica de modelos.

Keywords

Biological regulatory networks, Piecewise linear models, Reactive Boolean networks, Extended asymptotic graph, Bisimulation, Reconfigurability, Reactivity, rPrism, Weighted switch graphs.

Abstract

The study and development of tools for computational systems is an area where we can easily find diverse works and, nowadays, it is one of the dominant topics when we think about the research on computer science. As consequence, the field of computation has access to a solid theoretical basis, as well as to a wide collection of algorithms and tools (such as model checkers).

The focus of this thesis is to look at a biological system under a computational perspective, where cells and gens replace the role of transistors as the fundamental elements of a computational system. Indeed, the notion of computation is often compared to the functioning of a brain in an animal. Taking into account this point of view, the goal of this work is to revisit basic concepts present on the study of intracellular dynamics, which are fundamentally the same for all living organisms, under a computer science perspective. Thus, we intend to understand how we can apply concepts, algorithms and computational tools, which are used the field of Computer Science, to the mentioned biological systems. In particular, we start by describing some kinds of models used to model the intracellular dynamics of living organisms – Piecewise linear models and Boolean networks.

Hence, we propose a new perspective over Piecewise linear model, considering these models as reconfigurable. This allows one to use computational tools like KeYmaera and dReach to reason about these models. Afterward, discretizing this kind of model but maintaining the notion of reconfigurability, we obtain the concept of reactive Boolean network, based on the switch graph formalism, and propose a logical language to express and formally check properties of these systems along with a notion of bisimulation. In what relates to Boolean networks, we provide a new point of view over the notion of “terminal”, by relating it to the notion of bisimulation, which is widely known in the area of Computater Science. Then, we focus in the asymptotic graph method and, after a fundamental study, we propose a generalized and intermediate method that is less efficient in a computational perspective but more suitable to the intended context. Finally, we consider a new kind of stochastic model which is obtaining embedding weights in edges of switch graphs. We also develop an extension of PRISM model checker – rPrism – to ease the study of this specific class of stochastic models.

Contents

Contents	i
List of Figures	iii
List of Tables	v
1 Introduction	1
2 Background on quantitative and qualitative models	5
2.1 Quantitative models.	6
2.1.1 System of ordinary differential equations.	7
2.1.2 Piecewise Linear models.	10
2.2 Qualitative models.	15
2.2.1 Boolean networks.	16
Asynchronous vs synchronous approach	17
Multiple qualitative levels	17
2.2.2 From PWL models to Boolean networks.	20
Boolean networks with probabilities	22
2.2.3 Asymptotic graph.	22
2.3 Stable states in biological context.	27
3 Logic and computational tools to study PWL models	29
3.1 Differential dynamic logic.	29
3.1.1 Syntax.	29
3.1.2 Semantics.	30
3.1.3 Proof calculus.	35
3.2 Application to PWL models: Biological systems with hybrid dynamics.	36
3.2.1 Relating hybrid and reconfigurable systems.	36
3.2.2 PWL models as hybrid models.	39
3.2.3 Limitations and alternatives.	44
dReach - A complement to KeYmaera.	44
3.3 Final remarks.	46
4 Reactive Boolean networks: An intermediary step between PWL and BN models	49
4.1 Switch graphs.	49
4.1.1 Reactive frames, logic and bisimulation.	51

4.1.2	Weighted switch graphs.	61
	Fuzzy switch graphs.	62
4.2	Reactive Boolean networks.	65
4.2.1	Switch graph as a discrete reconfigurable system.	66
4.2.2	Recovering attractors.	67
4.3	Reactive Boolean networks with probabilities.	71
4.4	Final remarks.	73
5	Studying asymptotic dynamics in Boolean networks	75
5.1	Bisimulation and attractors.	75
	A related approach.	77
5.2	Extended asymptotic graphs.	78
5.2.1	Analyzing the AG method.	78
5.3	Final considerations.	83
6	A reactive approach to stochastic methods	85
6.1	Stochasticity and reactivity in biological context.	85
6.2	The tool: rPrism.	86
6.2.1	PRISM model checker.	87
6.2.2	Weighted graphs and PRISM models.	88
6.2.3	Biological regulatory networks as weighted switch graphs.	89
6.2.4	Language and examples.	92
6.3	Final remarks.	94
7	Conclusion and future research	97
	Bibliography	101

List of Figures

1.1	Levels of abstraction of a biological regulatory network model according to [10].	2
1.2	Enriched schema with biological regulatory network models.	4
2.1	Graph diagram of a regulatory network for the circadian rhythm of a cyanobacteria.	6
2.2	Comparison between several Hill functions (n=2, left; n=4, middle, n=16, right).	7
2.3	Example of a flow crossing more than one domain in a PWL model.	12
2.4	Some examples of boundaries with special dynamics.	13
2.5	Some examples of resulting dynamics at boundaries.	14
2.6	Boolean network of (synchronous) Bistable switch.	16
2.7	Synchronous (left) vs. asynchronous (right) approach.	17
2.8	Graph of a Boolean network with a variable admitting three qualitative values.	19
2.9	Graph of the Boolean network described by Equation 2.5.	19
2.10	Graph of an unknown BN.	20
2.11	A toy PWL model (left) and the corresponding BN model (right).	21
2.12	BN model with probabilities on edges.	22
2.13	Graphs of the submodels for the admissible input values.	25
2.14	The full asymptotic graph of the system in Example 2.2.7.	25
2.15	Graphs of the submodels for the AG model of circadian rhythm.	26
2.16	The full asymptotic graph.	26
3.1	Illustration of the hybrid programs induced transitions.	33
3.2	Bouncing ball example.	34
3.3	Initial screen of the bouncing ball problem in KeYmaera.	39
3.4	Orbit of the system.	41
3.5	Solution for the reachability problem.	47
4.1	Example of a switch graph representing a counter.	51
4.2	Evolution of a switch graph representing a counter.	51
4.3	The reactive frame of two non bisimilar reactive models.	56
4.4	Two bisimilar switch graph models.	57
4.5	Example of a Fuzzy switch graph.	64
4.6	Obtaining higher-level edges for a RBN model.	67
4.7	Illustrating the stable steady states.	68
4.8	Simplified Boolean network.	68
4.9	A reactive Boolean network.	69
4.10	BN model for the PWL model in Example 2.1.3.	72

4.11	Part of a BN model with probabilities.	72
4.12	Part of a RBN model with probabilities.	73
5.1	BN model of the circadian rhythm in a cyanobacteria.	77
5.2	Reduced model for the circadian rhythm in a cyanobacteria.	77
5.3	BN models for the subsystems A and B	79
5.4	Asymptotic graph with a spurious attractor.	79
5.5	Performance of the extended asymptotic graph method for asynchronous networks and comparison with the AG original method.	82
5.6	Auxiliary graphs for the construction of an EAG.	82
5.7	An extended asymptotic graphs with no spurious attractors.	83
6.1	Model representing the cooperativity of hemoglobin.	86
6.2	Plain representation for a hemoglobin cooperativity reactive model.	88
6.3	A graph diagram of circadian rhythm with some weights representing rates (left); and a complete weighted switch graph obtaining by removing inhibitor edges and component A (right).	90
6.4	Results from a stochastic simulation.	91
6.5	PRISM output for the growth of <i>E. coli</i> with glucose and lactose.	92

List of Tables

2.1	Truth table for x^+ (left) and y^+ (right).	20
3.1	Piecewise linear model with a discrete control variable μ	41

Chapter 1

Introduction

Synthetic biology is a broad field which comprehends diverse subareas and that can be seen, in an informal context, as the “engineering of biology”. The concept of “synthetic biology” was probably first mentioned in [48] by Poinat. This subject comprises diverse areas such as, for instance, genetic design, the synthesis of biocomponents and molecular logic [64] and has gained much attention during the last years due to the development of new and improved techniques to manipulate DNA.

Developments within these areas are relevant since they can then be diversely applied. Some examples of these applications are, for instance, the development of new medicines and medical therapies (such as documented in [34]), nanocomponents, or even biocomponents for biocomputers [67]. However, the microscopic dimension of the elements taking part on these processes is not reflected in the complexity of their interactions. In this way, detailed and specialized knowledge about genetic functionality, regulation and interconnection is required because one must understand the elementary units used to build bio-synthetic systems in order to obtain precise and sound models or designs for them.

The understanding of genetic networks is one of major concerns within the general subject of synthetic biology. It is important to understand the role that each gene plays individually in an organism, as well as its interconnection with other genes/components, the way they occur and which biochemical processes are involved. In order to attain this goal, several approaches are used. We can separate these approaches into two classes: (i) the ones that study the functions of genes by analyzing the DNA by itself; and (ii) the ones that study the functions of gene by observing its behavior and outputs in different environments or when interconnected with other genes. On one hand, class (i) includes, for instance, methods depending on the sequencing of DNA such as statistical analysis, the recognition of genetic patterns and phylogenetics. On the other, the methods on class (ii) comprise the analysis of genetic networks such as the modeling of biological regulatory networks where both theoretical analysis and simulations are performed; and empirical studies with laboratory experiments.

The focus of this thesis is placed on the second class of methods and, in particular, on the study of biological regulatory networks. Despite being embedded in a larger context, the subject of biological regulatory networks is a complex and wide topic by itself, in such a way that is possible to take several directions of study. This diversity begins with the large number of models and procedures proposed in the literature and continues with the diverse properties that can be studied. For example, we can consider a model for a regulatory network and our interest be the estimation of parameters; the study of the interconnections between

two genes or a gene and a protein; or the study of steady states of a system and/or their (asymptotic) stability or cyclic behavior(s). Moreover, for each of these topics, we can find in literature diverse approaches and methods to apply. To begin with, we can consider either a deterministic or stochastic approach. Because of this, and in order to follow a concise and consistent direction in this work, we took the document [10] of Chaves as a guiding line for this thesis.

In the referred document a deterministic methodology for the study of a biological regulatory network is presented. Instead of using a unique formal model to study biological regulatory networks, the author proposes to combine several formalisms in order to obtain the maximum benefit of each of them regarding their relative advantages and disadvantages. In particular, an algorithmic procedure to obtain different kinds of models (going from continuous to discrete) is proposed. This methodology consists of starting with a continuous model and obtaining, step-by-step, gradually more abstract ones. In practice, given a classical model composed by a *system of ordinary differential equations* (ODE model), it is explained how we can obtain a corresponding *Piecewise linear model* (PWL model) (an hybrid model comprising both continuous and discrete formalisms); from a Piecewise linear model, we can go more abstract to obtain an *Asynchronous Boolean network* (asynchronous BN). The chain ends with a formalism intended to reduce the size of the Boolean model is the *Asymptotic graph method* (AG). These levels of abstraction, as illustrated by Figure 1.1, are powerful tools for the study of the dynamics of biological regulatory networks, namely in what concerns the study of steady states of the system.

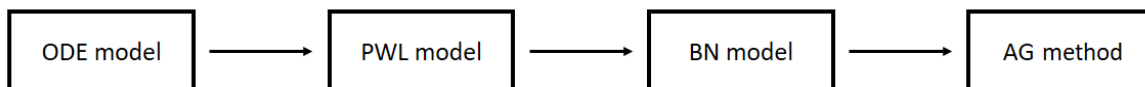


Figure 1.1: Levels of abstraction of a biological regulatory network model according to [10].

The study of a biological system modeled by a system of ODEs can then be performed in the following way: we start by constructing each one of the gradually more abstract models. Then, starting with the simpler one, its main dynamics are studied. Afterward we can gradually move to more complex models so that more detailed properties of the system are recovered.

The main contribution of this thesis is to establish connections between concepts, algorithms and tools which are already used in Computer Science and the study of biological regulatory networks. In this way, biological concepts, methods and models are revisited under a computational perspective, which turns possible to approach them under a new outlook and to apply some methods and tools that come from the fields of Logic and Computer Science. As mentioned above, in this thesis we take the work of Chaves as a guiding line and enrich the diagram presented in Figure 1.1 with new levels and kinds of models. Moreover, we provide additional insights and tools for the study of some already existing formalisms. Therefore, our work and contributions can be summarized on the following points:

- In Chapter 3, PWL models are studied as hybrid and, even more generally, as reconfigurable models. This allows one to use logical and computational formalisms which were specifically designed to study hybrid systems. We propose the use of differential dynamic logic [55] because the syntax of differential dynamic logic is particularly useful

for describing hybrid systems (which are sometimes known as cyber-physical). Also, this logic has a proof calculus which can be used to prove some properties of these systems. Moreover, a tool called KeYmaera was developed by A. Platzer to obtain the proof of a formula in a semi-automatic way. We discuss the utility of this tool as it seems to be specially useful to prove safety properties of PWL models and not particularly useful to prove reachability properties. Finally, we compare it with other approaches and propose an alternative tool – dReach –, for some specific reachability problems. The work presented in this topic is based in a paper published in [17].

- In Chapter 4, we propose reactive transition state models – *Reactive Boolean networks* (RBN models) – as an intermediate model between PWL and BN model. When comparing PWL and BN models, we lose a considerable amount of information. RBN models can reduce this loss and their advantages and disadvantages when comparing to both BN models and PWL models are studied. A RBN is based on the formal concept of switch graphs introduced by Marcelino and Gabbay [26]. During this chapter we further study this kind of structure and obtain some important notions and results like a suitable notion of bisimulation and an Hennessy-Milner theorem. A modal logic is also proposed for describing properties of this kind of structure and its semantics are adapted to the reactive features of switch graphs. Afterward, this concept is generalized leading to the introduction of weighted switch graphs, along with some additional study of the fuzzy case. Weighted and reactive structures allows us to propose weighted reactive Boolean networks as a generalization of Reactive Boolean networks, in order to include weights on edges. We discuss how they can be applied to the study of biological regulatory networks. In particular, we think about the case where these weights are probabilities. The work presented here can be found in [19, 20, 22, 59].
- In Chapter 5, we study logical concepts and properties in Boolean networks and, furthermore, establish a connection between attractors (a main property of some set of vertices in biological BN models) and bisimulation. This connection provides insights about the notion of attractors in a theoretical and formal context and paves the way for the development of new methodologies and algorithms to the study of these models. We also study the disadvantages of the AG method when comparing to BN models and propose an intermediate one – *Extended asymptotic graph* (EAG). Its performance is analyzed by comparing it with both BN models and the usual AG method. Asymptotic graphs are obtained by simplifying a Boolean network and comprise, for sufficiently larger models, much less states than typical BN models. Even so, they preserve much information about the original Boolean Network. We study which important information is lost and the cause for that loss. The work presented in this chapter is published in [18, 13].

In this way, we obtain a new classification schema extending the one shown in Figure 1.1. This is depicted in Figure 1.2 where the contributions of this thesis are highlighted by dashed boxes.

- Finally, and apart of this methodology, the pertinence of determinism is discussed and models which are both reactive and stochastic are proposed in Chapter 6. We also show how the software PRISM can be used to simulate and study such systems. This is presented as a complementary work, since deterministic models are not suitable for

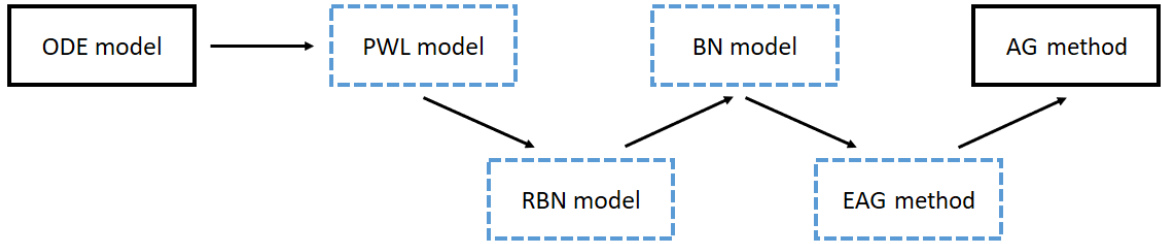


Figure 1.2: Enriched schema with biological regulatory network models.

general systems and can, themselves, be seen as simplifications of stochastic models. For this, we consider weighted reactive models, already mentioned, and explain how these models can be constructed and studied. In particular, a tool which was developed in this thesis to specifically study this kind of models is presented. This software, called rPrism, is able to recognize and deal with reactive models inside the PRISM framework. This is published in [22].

In order to provide a guiding line along all these kinds of models and approaches, we consider, whenever possible, a common biological system to exemplify the application of each model or tool. This example is the circadian rhythm system of a cyanobacteria according to a model documented in [14].

Outline. In Chapter 2, we describe the state-of-art: we introduce each one of the models shown in Figure 1.1 as well as a complete analysis about the differences between them and how to obtain a more abstract model from other. In a complementary way, it is also explained how the study of these distinct formalisms can be combined to more efficiently study a biological system. In Chapter 3, we start by presenting differential logic and how to apply it to the study of hybrid systems as PWL models naturally are. In Chapter 4 presents some major results as Reactive Boolean networks and their variants. A formal notion reactivity, as introduced by Gabbay and his collaborators, is provided and we study how RBN models fit in the schema of abstraction as presented in Figure 1.2. Chapter 5 presents a theoretical approach to Boolean networks and establishes a correspondence between bisimulation and a major asymptotic property of this kind of model. Then, a new methodology – extended asymptotic graph – is proposed in order to improve the results obtained with the AG method. In this way, we analyze AG models and, based on this study, EAG models are presented as an alternative. In Chapter 6 we discuss which cases should not be studied with deterministic methodologies and, as complement, we propose a reactive formalism along with the software PRISM as an alternative to deterministic models and methodologies. Then, we also introduce the software rPrism as a tool for the study of the reactive and stochastic models previously presented. Finally, Chapter 7 closes the thesis with some conclusions and hints for future work.

Chapter 2

Background on quantitative and qualitative models

When we think about a biological regulatory network, we must consider a set of biological components which are present in a specific intracellular context. These components interact and chemically react according to physical and chemical laws. The global processes occurring within a cell are then responsible for guiding the cell according its functions and cycle. It is important to mention that the physical and chemical laws which are applied to inanimate matter are the same which drive the basic processes of life (see [8]).

Taking into account the large amount of components interacting within a cell – proteins, mRNA, gens, molecules, *etc...* – one cannot study with detail the entire physic-chemical map of a cell. Instead, researchers consider subsets of components according to the specific cell function being studied ([52]). For instance, if the goal is to study the growth and replication mechanisms of a cell, only the components which are believed to take part on the process are considered – this subset of components is mentioned in literature as “module”. However, usually these modules still comprise a large number of distinct components which interact with each other. Then, instead of describing a biological model using physical and chemical equations, the notion of *regulation* is considered. A component a is said to regulate a component b whenever the presence of a significant concentration of a within the cell can influence the concentration of b in the future. For instance, a specific mRNA is said to regulate the protein produced when it is translated into the corresponding organelle. Furthermore when we mention a positive (respectively, negative) regulation of a over b , we refer a regulation where the production/activation of b is induced (respectively, inhibited) by the presence of a or inhibited (respectively induced) by the absence of a . Models rather use this notion of “regulation” to describe interactions between components than the general physical and chemical laws underlying the same regulations. It is important to mention that modules are chosen in such a way that components within the module do not considerably react with components out of it. Also, a module represents itself a biological regulatory network and, often, several submodules are considered for specif biological regulatory networks.

Biological regulatory networks usually contain a large number of interconnections due to regulations. Thus, it is usual to represent them graphically in order to proceed with a preliminary and conceptual analysis. For this, we can use a special class of graphs (V, E) where the set of vertices coincide with the set of components and the set of edges $E \subseteq V \times V \times \{+, -\}$ describe the regulation relations. Here, an edge $(a, b, *)$ describes a positive regulation of a

over b if $*$ = + and a negative one otherwise. Graphically, positive regulations are usually represented in literature by a line ending with either an usual arrow or a black circle; while negative regulations are represented by a line ending with either an orthogonal line or a white circle. As example, in Figure 2.1, we present a diagram for the regulatory network of circadian rhythm of a cyanobacteria, which can be found in [14]. In this model two proteins are considered – KaiA and KaiC. While the component KaiA is represented by A , KaiC admits both an unphosphorilated form – represented by U – and three phosphorilated forms – represented by T , TS , S . Therefore, the graph representation of this regulatory network, shown in Figure 2.1, let us know how these components regulate each other. In particular, we are able to understand that, for instance, T positively regulates TS and S negatively regulates A .

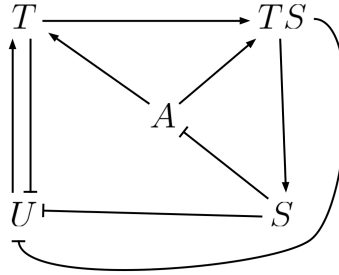


Figure 2.1: Graph diagram of a regulatory network for the circadian rhythm of a cyanobacteria.

Biological regulatory networks as the one illustrated in Figure 2.1 are studied in order to provide insights about how these biological modules operate. In particular, it is common to seek for features like stable states and periodic behaviors.

We begin this document by revisiting the methods, ideas and models already applied to this topic. Recalling the fact that this entire process is ruled by physical and chemical laws, exact models should rather be stochastic than deterministic. However, and since in practical cases, biological systems usually comprise a huge number of elements of each considered component, deterministic models are a reasonable choice due to the law of large numbers. Because of this, it is important to, when studying a biological regulatory network with a deterministic approach, carefully specify the dynamics of the studied system using a suitable model. Hence, in this chapter, several kinds of deterministic models are presented and compared. Moreover, some connections between them are highlighted. Also, for more information about modeling biological regulatory networks we recommend to see [40].

2.1 Quantitative models.

The term “quantitative model” is used to describe a class of models that represents a biological system by assigning a continuous variable to each component considered. These variables will then indicate the absolute concentration of each component within the environment of the system under study. Deterministic models use these variables to describe the evolution of a model over time. This is done by representing the regulation relations by suitable numerical equations like, for instance, differential equations. Moreover, we note that each state of such a system is completely described by the values of all its variables. During

the rest of this section we present two kinds of quantitative models.

2.1.1 System of ordinary differential equations.

As mentioned before, the variables considered for this kind of system take real values. However, since we consider that each variable represents the concentration of a component from a biological system, we must guarantee positivity constraints for all variables, since a negative concentration has no meaning in real cases.

Generally, when one uses a system of differential equations to model a biological regulatory network, it is common to follow the methodology described below.

Let us consider a biological regulatory network where a_1, \dots, a_n is the list of components involved. Then we introduce real variables x_1, \dots, x_n such that x_i represents the concentration of component a_i within the cellular environment. At this point, we recall that deterministic models describe the dynamics of a biologic regulatory networks using the concept of regulation. Thus, the differential equation describing the evolution of some variable x_i depends on the concentration of each variable x_j such that the component a_j regulates the component a_i . These regulations must be described by a quantitative function s describing the level of regulation in terms of the concentration of the regulative protein a_j . In practice, this means that an expression $s^+(x_j)$ (respectively, $s^-(x_j)$) must be integrated in the differential equation describing the evolution of x_i whenever the component a_j regulates positively (respectively, negatively) the component a_i .

Several authors make different choices for the expression of $s^+(x)$. In general, all they agree to consider a sigmoid function such that $s^+(0) = 0$ and $\lim_{x \rightarrow +\infty} s^+(x) = 1$ (see [70]). For instance, a natural choice is to use an expression with $\arctan(x)$ or a logistic function. However, the expression for $s^+(x)$ which is most commonly found in literature is the so-called *Hill function* which takes the form: $s^+(x; \theta, n) = \frac{x^n}{x^n + \theta^n}$, where $\theta, n \in \mathbb{R}$ are parameters. Since this choice for the sigmoid function representing a positive regulation depends on parameters, it has the advantage of being possible to adjust these parameters in order to obtain a model that fits experimental data in a better way. In Figure 2.2 we can see an example of some of these sigmoid functions. While the value of θ determines the point where the sigmoid function takes the value 0.5, the value of n is responsible for the slope around $x = \theta$. Along this thesis we consider the latter definition of $s^+(x; \theta, n) = \frac{x^n}{x^n + \theta^n}$ along with

$$s^-(x; \theta, n) = 1 - s^+(x; \theta, n) = \frac{\theta^n}{x^n + \theta^n}.$$

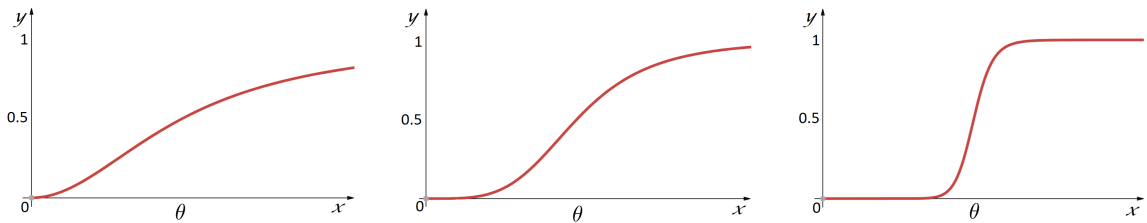


Figure 2.2: Comparison between several Hill functions ($n=2$, left; $n=4$, middle, $n=16$, right).

Note that this kind of expression for s^+ and s^- can only take values between 0 and 1. This is done in order to represent a “normalized” regulation and, in practice, an additional

constant $k_{ij} > 0$ is considered in such a way that a positive regulation of a component j over a component i is described by the differential equation $x'_i = k_{ij}s^+(x_j; \theta, n)$ and a negative regulation between the same components is described by the differential equation $x'_i = k_{ij}s^-(x_j; \theta, n)$. Since s^+ and s^- solely takes values in between 0 and 1, k_{ij} can be understood as the maximum effect of a_j over a_i . Sometimes, more than one component can, cooperatively, regulate another one. If a_1, \dots, a_m cooperatively regulate a_0 , then this is represented by the expression $k_{0,1\dots m} \prod_{l=1}^m s^{*l}(x_l; \theta_{a_0, a_l}, n)$, where $*_l$ can be either $+$ or $-$, according to the case.

Each equation of an ODE model uses sums and products of this kind of expressions to describe all regulations. Furthermore, an additional constant is considered for each component equation because, in a biological context, the effect of natural degradation along time must be considered. Different components degrade at different rates and, therefore, we must consider a degradation constant $\gamma_i > 0$ for each component i .

Putting all this together, we are now able to present an example of an ODE model.

Example 2.1.1 (Circadian rhythm of a cyanobacteria – ODE model).

Recalling the example of circadian rhythm in Figure 2.1, we can obtain a generic model for it as follows. Since the system considers components A , U , T , TS and S , this model must consider variables x_a , x_u , x_t , x_{ts} and x_s , for the respective concentrations. Following the example as introduced in [14], we consider that the total concentration of KaiC (both its unphosphorylated and phosphorylated forms) is constant, *i.e.* $x_u + x_t + x_{ts} + x_s = C$, with C being a constant value. This allows us to reduce one equation to the ODE model, which is described by the following system:

$$\begin{cases} x'_a = k_{a,s} \frac{\theta_{a,s}^n}{x_s^n + \theta_{a,s}^n} - \gamma_a x_a \\ x'_t = k_{t,ua} \frac{x_u^n}{x_u^n + \theta_{t,u}} \cdot \frac{x_a^n}{x_a^n + \theta_{t,a}} - \gamma_t x_t \\ x'_{ts} = k_{ts,ta} \frac{x_t^n}{x_t^n + \theta_{ts,t}} \cdot \frac{x_a^n}{x_a^n + \theta_{ts,a}} - \gamma_{ts} x_{ts} \\ x'_s = k_{s,tsa} \frac{x_{ts}^n}{x_{ts}^n + \theta_{s,ts}} \cdot \frac{\theta_{s,a}^n}{x_a^n + \theta_{s,a}^n} - \gamma_s x_s \end{cases}$$

with $x_u + x_t + x_{ts} + x_s = C$, a constant value.

As mentioned before, one advantage of this kind of model is to allow the estimation of parameters. In particular, in [14], for this circadian rhythm model, parameters were estimated for $n = 4$, resulting in the following model:

$$\begin{cases} x'_a = 10 \frac{5^4}{x_s^4 + 5^4} - 0.45 x_a \\ x'_t = 20.51 \frac{x_u^4}{x_u^4 + 29.95^4} \frac{x_a^4}{x_a^4 + 10^4} - 0.24 x_t \\ x'_{ts} = 10.74 \frac{x_t^4}{x_t^4 + 11.42^4} \frac{x_a^4}{x_a^4 + 10^4} - 0.28 x_{ts} \\ x'_s = 6.61 \frac{x_{ts}^4}{x_{ts}^4 + 10.16^4} \frac{13^4}{x_a^4 + 13^4} - 0.08 x_s \end{cases}$$

with $x_u + x_t + x_{ts} + x_s = C$, a constant value.

With the model presented in this example, we are able to recover the dynamics of the regulatory network of circadian rhythm. In particular, by studying this model, Chaves & Preto, in [14], were able to recover the cyclic behavior, as observed in the real life system, along with realistic values for the cycle period. It is important to mention that, in this context, a state of a biological regulatory network is fully and unequivocally described by the values for the concentration of their components. For instance, considering the ODE model for the circadian rhythm which was presented above, $(x_a, x_t, x_{ts}, x_s) = (0, 0, 0, 0)$ describes a state of the model.

Definition 2.1.1. Let $x = (x_1, \dots, x_n)$ and $x' = F(x)$ be a system of ordinary differential equations, with $x'_i = F_i(x)$, for $i \in \{1, \dots, n\}$. We say that a *flow* from a state $\bar{x} \in \mathbb{R}^n$ to a state $\bar{y} \in \mathbb{R}^n$ is a continuously differentiable function $f : \mathbb{R}_0^+ \rightarrow \mathbb{R}^n$ such that:

- $f(0) = \bar{x}$
- $\exists t' \in \mathbb{R}^+ : f(t') = \bar{y}$
- $f(x)$ verifies the differential equations, *i.e.* $f'_i(t) = F_i(x)$ whenever $f(t) = x$.

For any of these ODE models we always obtain an *invariant region*, *i.e.* a subset of the entire state space such that there is not any flow from a state inside the subset to a state outside it. This is obtained for each variable x_i as the quotient between the sum of maximum effects over a component i , and γ_i , its degradation rate. This will be described further with more detail but, in particular, for the ODE model presented in Example 2.1.1, we obtain that if $(x_a, x_t, x_{ts}, x_s) \in [0, \frac{k_{a,s}}{\gamma_a}] \times [0, \frac{k_{t,ua}}{\gamma_t}] \times [0, \frac{k_{ts,ta}}{\gamma_{ts}}] \times [0, \frac{k_{s,tsa}}{\gamma_s}]$, then (x_a, x_t, x_{ts}, x_s) will always be within that region. The existence of this invariant region is a useful feature since it implies two useful properties:

- Positivity constraints for variables are assured if the initial value is within the invariant region;
- An invariant region often admits a steady state or a closed orbit.

The existence of steady states and closed orbits are important as will be seen forward. Also the positivity of variables is fundamental since it provides meaning and coherence to these models. Actually, it is not difficult to prove that, for any of these ODE models for biological regulatory networks, a flow whose initial state is outside the invariant domain will eventually reach the interior of the invariant region (this can be proven by generalizing the proof of Proposition 2.1.1). Thus, this implies that every steady state and closed orbit is within the mentioned invariant region. This is important since one can restrict the entire state space to a smaller domain.

Although this kind of model represents the dynamics of a biological regulatory network in a realistic way, it is not easily studied and features like closed orbits are difficult to be found. Indeed, these models are often used to perform simulations and give insights of how a system evolve, rather than to study them analytically. This is due to the nonlinearity of the equations and expression involved, and it presents an obstacle to the application of other methodologies such as computational implementations.

2.1.2 Piecewise Linear models.

To overcome the presented drawback of models composed by nonlinear differential equations, some alternatives and simplifications have come out. One of them is a class of models known as Piecewise linear or PWL, for short.

A method for obtaining a PWL model is to take an ODE model as a starting point and to simplify it in order to make more amenable to be studied by analytic and computational processes. The general idea of a PWL model is to consider an ODE model and divide its invariant region into smaller and mutually exclusive domains in order to approximate the nonlinear differential equations by specific linear differential equations within each smaller domain.

It is important to note that PWL models are considered qualitative models by some authors (see [30]). However, taking into account the approach proposed in this thesis, it makes sense to consider it as a quantitative model.

Before entering into details, we introduce some notation in order to make the following description clearer.

Note 1. Consider a ODE model, whose set of components is represented by $A = \{a_1, \dots, a_n\}$. We introduce the following abbreviations:

- k_i represents the sum of all parameters with form $k_{i,j}$ where j can be a single component or several cooperative components. Informally, it represents the sum of all “ k ” parameters of the differential equation relative to x_i .
- γ_1 abbreviates γ_{a_1} .
- For each component $a \in A$, we introduce $\theta_1^a, \dots, \theta_{n_a}^a$ such that $n_a \geq 1$ and $\theta_1^a < \dots < \theta_{n_a}^a$, and for all $b \in A$, if $\theta_{b,a}$ occurs in the ODE model, then exists $i \in \{1, \dots, n_a\}$ satisfying $\theta_{b,a} = \theta_i^a$.

Finally, for every ODE model, defined as described in previous section, we already mentioned that it was always possible to find an invariant region. Using the notation above, we are able to define generically the invariant region as $[0, \frac{k_1}{\gamma_1}] \times \dots \times [0, \frac{k_n}{\gamma_n}]$. The region generically presented here is mentioned simply as the *invariant region* of the corresponding ODE or PWL model in the sequel.

Proposition 2.1.1. *Consider an ODE model for a biological regulatory network using Hill functions and with n components. Then $[0, \frac{k_1}{\gamma_1}] \times \dots \times [0, \frac{k_n}{\gamma_n}]$ is an invariant region.*

Proof. The proof proceeds by analyzing the vector (x'_1, \dots, x'_n) at the boundaries of this region and showing that it is either null or points to inside the region.

Let us now consider a state such that $x_i = \frac{k_i}{\gamma_i}$ for arbitrary $i \in \{1, \dots, n\}$. Then, since Hill functions are never greater than 1, at this state $x'_i \leq k_i - \gamma_i x_i = k_i - \gamma_i \frac{k_i}{\gamma_i} = 0$.

Let us consider a state such that $x_i = 0$ for arbitrary $i \in \{1, \dots, n\}$. Then, since Hill function are never negative and $k_i \geq 0$, at that state $\gamma_i x_i = 0 \Rightarrow x'_i \geq 0$

Thus, since a flow is a continuous function, the region is invariant. \square

As mentioned before, in a ODE model, we must consider thresholds k , θ and n for Hill functions – which are estimated – in order to approximate the reality. PWL models simplify ODE models by taking the estimated values for thresholds k and θ but ignoring the value of n . In order to eliminate nonlinearity in the differential equations, PWL models consider that $n \rightarrow +\infty$. Because of this, sigmoid functions s^+ and s^- become *Heaviside functions* as illustrated by the following computation:

$$s^+(x; \theta, n) = \frac{x^n}{x^n + \theta^n} \xrightarrow{n \rightarrow +\infty} \begin{cases} 0, & \text{if } x < \theta \\ \frac{1}{2}, & \text{if } x = \theta \\ 1, & \text{if } x > \theta \end{cases} \quad (2.1)$$

$$s^-(x; \theta, n) = \frac{\theta^n}{x^n + \theta^n} \xrightarrow{n \rightarrow +\infty} \begin{cases} 1, & \text{if } x < \theta \\ \frac{1}{2}, & \text{if } x = \theta \\ 0, & \text{if } x > \theta \end{cases} \quad (2.2)$$

Thus, we are able to obtain, at each state, a system of linear differential equations. Moreover, given an n -dimensional state space, we can still find n -dimensional domains such that the system of differential equations is the same for all states inside each one of these domains and, as mentioned, linear.

In practice, for each variable x relative to a component a , we define the hyperplanes $x = \theta_1^a, x_j = \theta_2^a, \dots, x_j = \theta_{n_a}^a$ as *boundaries*. Then, considering the invariant region, mentioned before, and excluding all boundaries relative to each component, the remaining state space is composed by disconnected *domains*. In this way, we ignore the cases where the values for s^+ and s^- could take the value $\frac{1}{2}$, which are not considered in PWL model. Given a model with components $\{a_1, \dots, a_n\}$, we denote the set of all these domains by \mathcal{V} which can simply be described as:

$$\left\{ [0, \theta_1^{a_1}[,]\theta_1^{a_1}, \theta_2^{a_1}[, \dots,]\theta_{n_{a_1}}^{a_1}, \frac{k_1}{\gamma_1}] \right\} \times \dots \times \left\{ [0, \theta_1^{a_n}[,]\theta_1^{a_n}, \theta_2^{a_n}[, \dots,]\theta_{n_{a_n}}^{a_n}, \frac{k_n}{\gamma_n}] \right\}$$

For a specific domain, we say that a *state is in its boundary* whenever it is not contained in the domain itself but it is contained on its topological frontier, regarding the state space. Also, two domains are said to be *adjacent* if the intersection of their topological frontier is a $(n - 1)$ -dimensional region of the state space. This region is said to be the *shared or common boundary* between both domains.

The main advantage of this kind of model is that one is able to analytically solve the system of differential equation and, thus, fully describe the flows and dynamics of these simplified models. Moreover, the main dynamics of the original ODE model is preserved whenever the process described before is followed. In particular, these models ease the process of finding steady states and closed orbits since we are able to use exact procedures instead of numerical simulations. The main concern about this is the possible loss of information and the relevance of the results obtained. Nevertheless, studies with numerical simulation in [29, 30, 31], shows that there is no difference in the qualitative properties of the solutions in most cases.

As mentioned, the system of differential equations within each domain $V \in \mathcal{V}$ is linear. Moreover, it can be represented as follows:

$$\begin{cases} x'_1 = k_1^V - \gamma_1 x_1 \\ \dots \\ x'_n = k_n^V - \gamma_n x_n \end{cases}$$

where the value of k_i^V , for each $I \in \{1, \dots, n\}$ is obtained according to the domain.

Given a domain V , it is easy to see that the unique steady state of the respective linear ODE is stable and given by $\bar{x}_V = (\frac{k_1^V}{\gamma_1}, \dots, \frac{k_n^V}{\gamma_n})$. Moreover, the entire domain V is on its basin of attraction. We denote the state \bar{x}_V by *focus* (or *focal point*) of the domain V , since all flows whose initial state is on V asymptotically converge to \bar{x}_V when following the respective system of linear differential equations.

Given this, if the focus of a domain is contained on it, we can conclude that the domain is invariant and the focus is a stable steady state. However, if the focus of a domain is outside of it, in order to compute the evolution of the model starting at some initial state inside that domain, we can think about the flows which start at that initial state. However, if the focus of a domain D is outside D , then there is a flow between the initial state and a state on the boundary of the considered domain D . In this case, in order to compute the further trajectory of the flow, we can concatenate it with a new one starting on that point in the boundary of D and that is directed by the differential equations of the reached adjacent domain. This allows us to obtain flows crossing more than a single domain and, thus, one can retrieve the general dynamics of the entire model. In order to illustrate this, an example is introduced.

Example 2.1.2. In this example, we consider a toy PWL model which is shown in Figure 2.3. This model considers two variables – x and y – describing the concentration of two components which positively regulates each other. The degradation rate is common for both components and it is 1. In this model, two boundaries are considered: $x = \theta_1$ and $y = \theta_2$. In the figure, the state space is represented as well as the invariant region and the boundaries of the model. Furthermore, a system of differential equations is shown within each domain meaning that it guides the flows inside the respective domain.

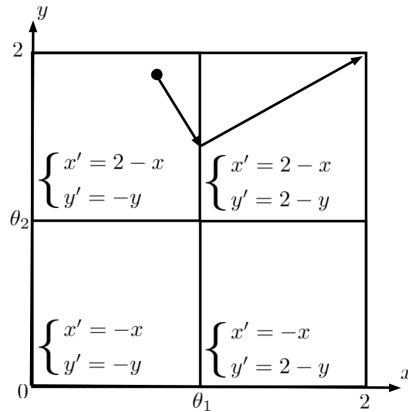


Figure 2.3: Example of a flow crossing more than one domain in a PWL model.

For this model, and given the systems of differential equations for every domain, we are able to compute the focal points. For instance, in the domain where $\theta_1 < x < 2$ and

$\theta_2 < y \leq 2$, the system of differential equations considered is $x' = 2 - x \wedge y' = 2 - y$, thus the focus is $(2, 2)$. Since $(2, 2)$ is within this domain we can immediately conclude that $(x, y) = (2, 2)$ is a stable steady state of the model. This does not occur in the domain where $0 \leq x < \theta_1$ and $\theta_2 < y \leq 2$ since the focus of this domain is $(2, 0)$, which is not contained on it. Hence, as mentioned before, the differential equations ruling the dynamics within this domain will drive all states to some respective state on boundary. This mean that there will always exist a flow between any state of this domain and a state in one of its boundaries. This is exemplified in Figure 2.3 where a state within this domain is represented by an enlarged point. On the same domain an arrow represents the flow from that point to the respective point at the boundary. Thereafter, another arrow represents a flow from same point in the boundary to $(2, 2)$, which is the focus of the domain where $\theta_1 < x \leq 2$ and $\theta_2 < y \leq 2$. In practice, we can concatenate both flows and simply say that there is a flow from the stated represented by the enlarged point and the stable steady state of $\theta_1 < x \leq 2$ and $\theta_2 < y \leq 2$. Therefore, this generalized notion of flow captures the dynamics resulting from the interconnection all domains of the entire system.

About the previous example, some additional explanations are given:

- Although the flows are represented by straight lines in Figure 2.3, we note that, in general, they are described by curves in the state space. However, when all considered degradation rates are identical, these flows become straight, as can be observed by solving the system of differential equations analytically.
- There is some linguistic liberty, so to speak, when saying that there exists a flow from some state to $(2, 2)$ since the convergence to this steady state is asymptotic. This is done in this thesis for simplicity but, formally, we should say that there exists a flow from some state to some other state as close as we want to $(2, 2)$.
- In this example we considered an implicit dynamic at the boundaries in such a way that it was possible to cross them even with no formal dynamics being introduced. In general, these dynamics at boundaries cannot be though in such a simplistic way. Indeed, the study of the dynamics at boundaries is not a simple issue and many degenerate cases can occur (see [23]). During the rest of this subsection we present some of these cases.

Regarding PWL models as simplifications of ODE models, one can think about the dynamics on boundaries as given by the piecewise linear equations obtained as shown in Equations 2.1 and 2.2. However, some inconsistencies can arise.

In Figure 2.4, some special cases are illustrated. All these cases must be carefully treated to obtain coherent results. There, arrows represent flows, lines represent boundaries and points represent focal points.

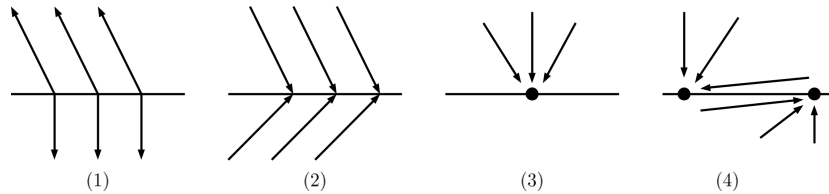


Figure 2.4: Some examples of boundaries with special dynamics.

Case (1) shows two adjacent domains where flows move away from the shared boundary. In this case, if we want to consider a flow starting at that boundary, two cases can occur: i) it will move into one of the domains; ii) it will slide along the boundary (this kind of dynamics at boundaries are known as *sliding modes*). Thinking about Equations 2.1 and 2.2, we can obtain a system of differential equations $x' = F(x) = (f_1(x), \dots, f_n(x))$. Thus, the resulting vector will determine the domain in which the flow will enter. However, it is also possible that the resulting vector is null or pushes the flow along the boundary. In these cases we can have an unstable steady state or an evolution toward the boundary, respectively. Cases of sliding modes are illustrated in Figure 2.5 where the vectors on a small neighborhood of the boundary are represented by a black arrow and the vector at the boundary is represented by a gray arrow. In a similar way, case (2) presents two domains whose flows move to the boundary. In this case, the question is how will the flow evolve once it reaches the boundary. Again, as in (1), one must think about the vector $F(x)$ at the boundary. However, in this case, and since both domains will force the respective flows into the boundary, the evolution must always occur toward the boundary. Also, as before, it can be the case that the resulting vector is null and we obtain a steady state which is, in this case, stable. This is also shown in Figure 2.5. When a flow evolves toward a boundary, more complex dynamics can occur but we will not present them here since we do not need them in the sequel. The case (3) considers a domain whose focus is in its boundary. Thus, no flow starting at that domain can cross the boundary since the convergence for the focus is asymptotic. Moreover, at the boundary itself, the dynamics is described using a strategy similar to the one in cases (1), (2). Finally, in (4), two adjacent domains admit focal points at the common boundary. In this case, the shared boundary cannot be crossed and, moreover the system will asymptotically approach each one of the focal points according to the dynamics of the respective domain. Moreover, within the boundary, the dynamics is obtained as described for cases (1) and (2). In particular, an additional unstable steady state may occur.

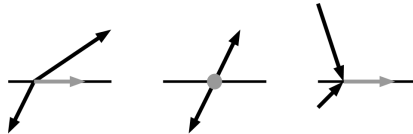


Figure 2.5: Some examples of resulting dynamics at boundaries.

To conclude this remark about dynamics of PWL models on boundaries, we point out that many other cases which are not listed could occur. Some generalizations of PWL models even consider each boundary shared by two domains as a domain with proper dynamics itself. However, we note that all these problems about dynamics at boundaries are solved trying to find the “most coherent” solution to flows (see [23]).

Here we note that the hybrid nature (both continuous and discrete) of this kind of model comes from combining the continuous formalisms within each domain with the discrete transition between these domains.

Finally, we close this subsection about PWL models by revisiting our guiding example of circadian rhythm presented in Figure 2.1.

Example 2.1.3 (Circadian rhythm of a cyanobacteria – PWL model).

For this example, we consider the ODE model introduced in Example 2.1.1. One must consider the parameters presented there to obtain a simplified PWL model.

Thus, we are able to obtain all domains and the respective system of linear differential equations. First, note that the invariant region for this model is given by $(x_a, x_t, x_{ts}, x_s) \in [0, \frac{10}{0.45}] \times [0, \frac{20.51}{0.24}] \times [0, \frac{10.74}{0.28}] \times [0, \frac{6.61}{0.08}]$. Also, the boundaries are hyperplanes described by the equations: $x_a = 10$, $x_a = 13$, $x_t = 11.42$, $x_{ts} = 10.16$, $x_s = 5$ and $x_u = 29.95$. However, since we consider x_u as a dependent variable, we must, instead, consider the equation $C - 29.95 = x_t + x_{ts} + x_s$, where C is the total amount of KaiC (which is considered to be a constant in the model) as an additional boundary.

Since the obtained model is 4-dimensional, it is not fully presented here. However, we exemplify with one domain. Considering $C = 60$ and the domain where $10 < x_a < 13$, $0 \leq x_t < 11.42$, $0 \leq x_{ts} < 10.16$ and $0 \leq x_s < 5$, then $29.95 < x_u \leq 60$ always holds. Within this domain, the system of differential equations obtained is:

$$\begin{cases} x'_a = 10 - 0.45x_a \\ x'_t = -0.24x_t \\ x'_{ts} = -0.28x_{ts} \\ x'_s = -0.08x_s \end{cases}$$

In this way, it is possible to compute the focus of this domain to determine the state to which all flows which start at this domain converge. Since it is $(\frac{200}{9}, 0, 0, 0) \approx (22.22, 0, 0, 0)$ which is out of this domain, all flows starting within this domain will eventually reach a boundary and leave. Since the degradation rates for each component are distinct, the flows within all domains of this model are not straight lines but curves.

PWL models, as simplified ODE models are more easily studied when compared to regular ODE models. In particular, there are no problems caused by nonlinearity of differential equation. However, this comes with the price of obtaining an hybrid model (both continuous and discrete) and losing some quantitative informations, as can be seen by the emergence of strange behaviors at boundaries.

2.2 Qualitative models.

Quantitative models describe a system using continuous variables to express, for instance, the exact value of the concentration of a component. Contrarily, qualitative models do not express variables like concentration by their absolute values but using qualitative terms like “high”, “medium” and “low”, for example. This allows us to use discrete variables and obtain much simpler models, when comparing with the quantitative ones, with continuous variables. We note that, because of this level of abstraction, qualitative models are not the best ones to design or analyze a system with great precision, however we have several advantages. Note that, in order to describe a dynamic system, these models can be depicted as graphs. This kind of representation is very useful since many algorithm and techniques can be used to study them. In this section we present Boolean networks, an important kind of qualitative model, and some variants, which will be used during the rest of the thesis. Finally, we also present a methodology to study large BN models.

2.2.1 Boolean networks.

Boolean networks (BN) are models which consider a variable for the concentration of each component, as before. This kind of model is called “Boolean” because each variable, related to a component of the system, can take either the value “1” or “0”, meaning that its concentration is respectively “high” or “low”. Thus, we do not obtain a precise model but, instead, a simpler one which can be studied in order to obtain a general overview of the dynamics of a system.

Given the nature of this kind of model, the dynamics is described using Boolean expressions. For instance, if two component, whose corresponding variables are A and B , positively regulate a component C , then we can express this by the equation $C^+ = A \vee B$. Also, if instead C is negatively regulated by some component whose correspondent variable is D , we can write $C^+ = \neg D$. We note that we wrote “ C^+ ” instead of “ C ” because these equations do not describe a specific state of the system but its general dynamics. In this way, and regarding that variables represent quantitative values for concentration such as “high” and “low”, “ C^+ ” can be understood as “the quantitative concentration of C in the future”.

Given the Boolean equations describing the dynamics of a model, we can introduce the idea of time steps in order to establish a temporal notion of evolution. In a graph-like representation, we can think that crossing an edge represents the evolution of the system in a time step.

Example 2.2.1 (Bistable switch). Consider a module of a system where only two components, whose corresponding variables are denoted by A and B , are considered. Also, let these two components inhibit each other mutually. This system is known as a bistable switch and is described by the following equations:

$$\begin{cases} A^+ = \neg B \\ B^+ = \neg A \end{cases} \quad (2.3)$$

Note that there are four possible states for this system, since $(A, B) \in \{0, 1\}^2$. Thus, if we introduce the notion of time steps, we can come up with a network which provides insights about the dynamics of the system. Consider, for instance, the state $(A, B) = (0, 0)$, which we simply write as 00, for short. Then, using the Equation 2.3, we can say that the state 00 will eventually evolve to become state 11 (here again, 11 is an abbreviation of $(1, 1)$). If we build a graph taking into account these dynamics, we obtain the graph shown in Figure 2.6. Note that in this figure also 10 and 01 are the respective abbreviations for states $(1, 0)$ and $(0, 1)$.

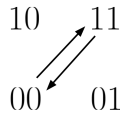


Figure 2.6: Boolean network of (synchronous) Bistable switch.

In this example we introduced the notion of graphs for a BN. The following remarks must be taken into account:

- Abbreviating states like $(1, 0)$ by 10 is very common. Indeed, in the sequel we will use this kind of notation freely even for a larger number of variables. Thus, at each

example an intrinsic order for variables must be considered in order to unambiguously identify states (such as before, in ODE and PWL models). For instance, in the previous example, state 01 is the state where the first variable – A – is 0/“low” and the second one – B – is 1/“high”.

- Two paradigms can be considered in BN models: synchronous and asynchronous. In Example 2.2.1 we considered a synchronous approach, however we will further discuss the difference between these two paradigms.
- Often, one may want to use other qualitative qualifiers different from “high” and “low”. Indeed, one can use other qualifiers such as “medium”, “very high” or “very low” introducing additional Boolean variables. We will also discuss this further in this section.

Asynchronous vs synchronous approach

When studying a BN model and its dynamics, one must think if the formal methods and models used are suitable to correctly describe what is observed in real life. Therefore, we note that, although Boolean networks are a discrete model (with a finite set of states and a finite set of transitions between states), reality is more complex. Indeed, recalling Example 2.2.1, we note that state 00 (eventually) goes to state 11, *i.e.* the state where the concentration of both components are “low” evolves to one where the concentration of the same components is “high”. This is not realistic since if both components have a low concentration, then only one at a time should become high. In fact, when we say that the concentration of a component is “high” or “low” we intrinsically consider a threshold which marks the limit between these quantitative values. It is not realistic that the concentration of both components cross such limit at the same time. This is the basis for the asynchronous paradigm.

Contrarily to the synchronous case where the values of all variables are updated at the same time, the asynchronous approach considers several possible transitions, corresponding to update only one variable (whose value changes) at a time.

To illustrate this, recall the Example 2.2.1 where the state 00 transit to state 11. In an asynchronous approach, 00 could transit to two different states: 10 (where solely the value of the first variable changes) and 01 (where solely the value of the second variable changes). The comparison is shown in Figure 2.7. Further in this section, a more complex example considering asynchronous approach will be presented.

In this thesis we will always consider the asynchronous paradigm for every BN.

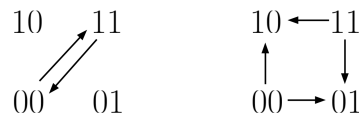


Figure 2.7: Synchronous (left) vs. asynchronous (right) approach.

Multiple qualitative levels

When studying a particular biological regulatory network, we may sometimes observe that a component whose concentration is described by a variable x positively regulates two other ones (let us call them a and b) in different ways: it is possible that there is some intermediate

values for x , between “low” and “high”, such that the corresponding component effectively induces the production of a but its concentration is still too low to effectively induce the production of b . In such cases, it is possible to consider a third qualitative state – “low”, “medium” and “high” – in order to describe the system in a more accurate way.

In terms of Boolean equations, these three qualitative levels can be obtained by considering a variable as a binary tuple (x, y) instead of a single binary value. Thus, $(0,0)$ means “low”, $(0,1)$ means “medium” and $(1,1)$ means “high”. Thus, a component with three qualitative levels needs two Boolean equations to describe its dynamics and special care must be taken in order to avoid non-sense states like $(1,0)$.

Example 2.2.2. Consider a toy model with three components A , B and C with a , b and c being the corresponding variables. When a is not low, it induces the production of B and if a is very high, then it inhibits the production of C . Also, B positively regulates C which, in its turn negatively regulates A . This induces a system which can be represented by the following Boolean equations:

$$\begin{cases} a^+ = \neg c \\ b^+ = a \\ c^+ = \neg a \wedge b \end{cases}$$

However, three qualitative values for the concentration of A can be considered in order to obtain a more precise model. Indeed, if we think that $a = (a_1, a_2) \in \{0, 1\}^2$, we can come up with the following Boolean equation to describe the dynamics of this toy model:

$$\begin{cases} a_1^+ = \neg c \vee a_2 \\ a_2^+ = \neg c \wedge a_1 \\ b^+ = a_1 \\ c^+ = \neg a_2 \wedge b \end{cases} \quad (2.4)$$

Here, we note that nonsense states are avoided by introducing some additional constraints in the equations corresponding to the evolution of a_1 and a_2 . Also, the remaining equations were updated accordingly.

At this point, we note that even more than three qualitative levels can be considered in a straightforward way by adding more additional variables and equations. Moreover, when constructing a graph representing the dynamics of a model where some variable admits more than two qualitative levels, we can simplify the notation by introducing abbreviations. For example, for three qualitative levels we can abbreviate in the following way: $0 = (0,0)$, $1 = (0,1)$ and $2 = (1,1)$. This is illustrated in Figure 2.8 where the graph corresponding to the BN model defined by Equation 2.4. More information about these multilevel approaches can be found in [27, 35].

Taking these aspects into account, we end this subsection recalling our guiding example of circadian rhythm.

Example 2.2.3 (Circadian rhythm of a cyanobacteria – Boolean network).

In [14], we can find a BN model for the circadian rhythm of a cyanobacteria. Again, in this context the model admits variables a , t , ts and s , which now are endorsed with qualitative meaning, described by their Boolean values. In the mentioned paper, the proposed BN model is described by the following Boolean equations:

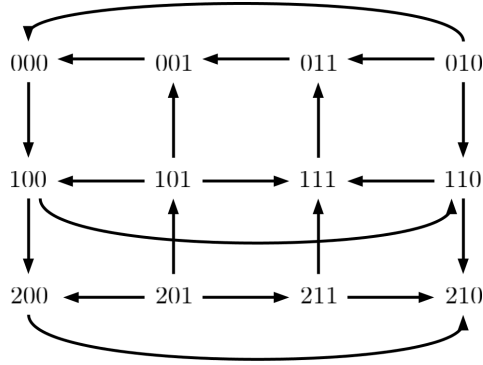


Figure 2.8: Graph of a Boolean network with a variable admitting three qualitative values.

$$\begin{cases} a^+ = \neg s \\ t^+ = u \wedge a \\ ts^+ = t \wedge a \\ s^+ = ts \end{cases}$$

where u can be considered as an abbreviation, such that $u \equiv \neg(t \vee ts \vee s)$. However, for simplicity, in a environment where KaiC is abundant, we can identify $u \equiv 1$. This simplifies the Boolean equation to the one presented in 2.5

$$\begin{cases} a^+ = \neg s \\ t^+ = a \\ ts^+ = t \wedge a \\ s^+ = ts \end{cases} \quad (2.5)$$

For this equation, we can now build the corresponding graph showing the dynamics of the system. It is shown in Figure 2.9.

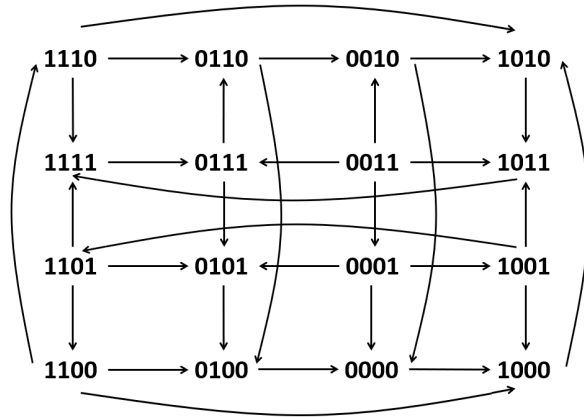


Figure 2.9: Graph of the Boolean network described by Equation 2.5.

2.2.2 From PWL models to Boolean networks.

In the same way that PWL models can be understood as simplifications of ODE models, BN models can be seen as simplifications of PWL models. This connection is not as direct as before but it can be obtained using the graphs of BN models. First of all, we must note that there is a bidirectional correspondence between the Boolean equations of a BN and its graph. Indeed, we can build a graph from the Boolean equations but also recover each one of the Boolean equations by considering the truth table for each variable of the form x^+ (which is implicitly described in the respective graph). This can be attained by using the method of Karnaugh maps (see [41] for details) and is illustrated by the following example:

Example 2.2.4. In this example we consider a graph of a BN model and determine the Boolean equation relative to each variable. Let us consider two variables x and y such that the graph of the corresponding BN model is depicted in Figure 2.10.

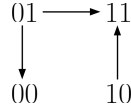


Figure 2.10: Graph of an unknown BN.

From this graph we can now build the truth tables for x^+ and y^+ . These are represented in Table 2.1. Taking the left table (for x^+) as example, when $(x, y) = (0, 0)$ then $x^+ = 0$ because there is not an arrow from 00 for 10. However, for $(x, y) = (0, 1)$ the value of x^+ is now 1 because there is an arrow from 01 to 11. Using a similar reasoning we can obtain all entries for the presented tables.

	y				y		
		0	1			0	1
x	1	1	1	x	1	1	1
	0	0	1		0	0	0

Table 2.1: Truth table for x^+ (left) and y^+ (right).

At this point we can use the method of Karnaugh maps to discover that the Boolean network is described by the following equations:

$$\begin{cases} x^+ = x \vee y \\ y^+ = x \end{cases}$$

Because of the connection between Boolean equations and graphs of BNs, we will often refer to such graphs as BN models in the sequel.

The simplification from PWL models to BN models is thus obtained through the notion of a graph for a BN model. Indeed, given a PWL model we are able to simplify it in order to build a graph. In this way, the obtained graph is now a BN model. The general idea to construct a graph from a PWL model is to consider each domain of a PWL as a vertex/state of the graph. This abstraction results from a qualitative interpretation of the domain.

Recall PWL models where each variable x_i define boundaries of the invariant region $[0, \frac{k_1}{\gamma_1}] \times \dots \times [0, \frac{k_n}{\gamma_n}]$, determined by some thresholds $x_i = \theta_1^{a_i}, \dots, x_i = \theta_k^{a_i}$ with $0 < \theta_1^{a_i} <$

$\dots < \theta_{n_{a_i}}^{a_i} < \frac{k_i}{\gamma_i}$. In a qualitative way, we can think of a new Boolean variable \bar{x}_i with multiple qualitative level such that $\bar{x}_i = 0$ whenever $x_i \in [0, \theta_1^{a_i}[$, $\bar{x}_i = 1$ whenever $x_i \in]\theta_1^{a_i}, \theta_2^{a_i}[$, ... and $\bar{x}_i = n_{a_i}$ whenever $x_i \in]\theta_{n_{a_i}}^{a_i}, \frac{k_i}{\gamma_i}]$ (remember the section about BN with multiple qualitative level).

Doing this to each variable of a PWL model, we obtain discrete variables $\bar{x}_1, \dots, \bar{x}_n$ which provide qualitative information about the concentration of the respective component of the system. In this way, we can establish a direct connection between each domain of a PWL model and each state of the BN model obtained *via* this simplification. To ease the notation, we denote a domain of a PWL model by V_a whenever a is the corresponding state in the simplified BN model obtained.

Then, in order to obtain a graph of a BN model from a PWL model, we must also be able to obtain the set of edges from the dynamics of the PWL model. This is done using the notion of flow. We admit an edge from a state of the BN model a to another state b if V_a and V_b are adjacent and there is a flow starting at some state within a which reaches the boundary shared by V_a and V_b , without crossing any other domain. The adjacency between V_a and V_b guarantees that we follow an asynchronous approach.

Consider the following example:

Example 2.2.5. Let us consider a PWL model illustrated in the left side of Figure 2.11. In this figure, we can observe that the invariant region in $[0, 3] \times [0, 2]$ and the boundaries are defined by the hyperplanes $x = 1$, $x = 2$ and $y = 1$. This generates 6 domains which are represented by 6 vertices in the simplified BN model. Furthermore, the vertices are named according to the qualitative value of state variables, as mentioned before. For instance, the domain $[0, 1[\times [0, 1[$ is named 00, $]1, 2[\times [0, 1[$ is named 10 and $]2, 3[\times [0, 1[$ is named 20.

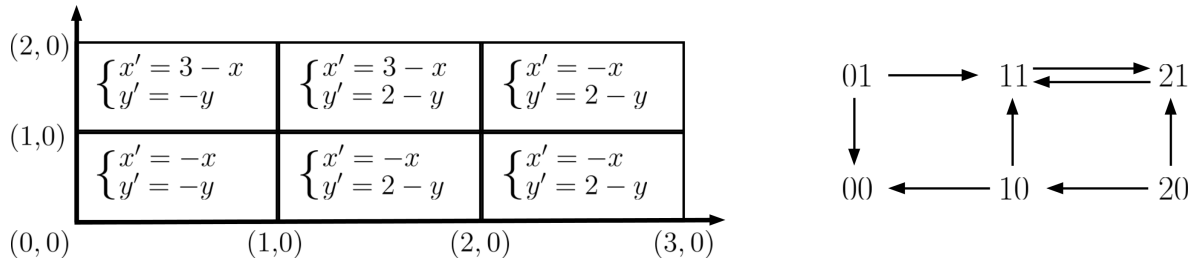


Figure 2.11: A toy PWL model (left) and the corresponding BN model (right).

In order to obtain the edges for the simplified BN model, (shown at the right side of Figure 2.11) one must think about the flow described by the system of linear differential equations within each domain. For instance, there is a flow from the state $(x, y) = (0, \frac{3}{2})$ (which belongs to $[0, 1[\times]1, 2]$) to $(x, y) = (\frac{1}{2}, 1)$ (which belongs to the boundary between $[0, 1[\times]1, 2]$ and $[0, 1[\times [0, 1[$). Thus, the corresponding simplified BN model admits an edge from 01 to 00. Also, since all flows starting at some state in $[0, 1[\times [0, 1[$ asymptotically converge to $(0, 0)$ in a straight trajectory, then there is no edges leaving 00, in the simplified BN model.

Boolean networks with probabilities

This class of simplified BN models includes a special class of graphs. Some authors propose the introduction of probabilities into a graph in order to provide some additional insights about the dynamics of the modeled system since we are able to observe whose transitions occur with greater probability into a graph (see [12]). Another (stochastic) approaches to this kind of model are found in [43, 65].

The idea of this kind of graph is to assign a probability to each edge. This probability is related with the probability of each transition to occur and is computed in the following way:

1. Consider a domain A of a PWL model which admits a flow leaving to some other adjacent domain B .
2. Let $I_B(x) : A \rightarrow \{0,1\}$ be a function which assigns to each state of A the value 1 whenever there is a flow starting at x to a state in the boundary between A and B , and assigns the value 0 otherwise.
3. In the simplified model, let a and b be the vertices corresponding to domains A and B , respectively. The probability assigned to the edge (a, b) is $\frac{\int_A I_B(x) dx}{\int_A dx}$

Example 2.2.6. We present a simple 2-dimensional example. Consider the same PWL as in Example 2.2.5. In Figure 2.12 (left) we can see the same PWL model but with the flow illustrated by edges. On the right side, we can see the respective BN model with probabilities assigned to each edge.

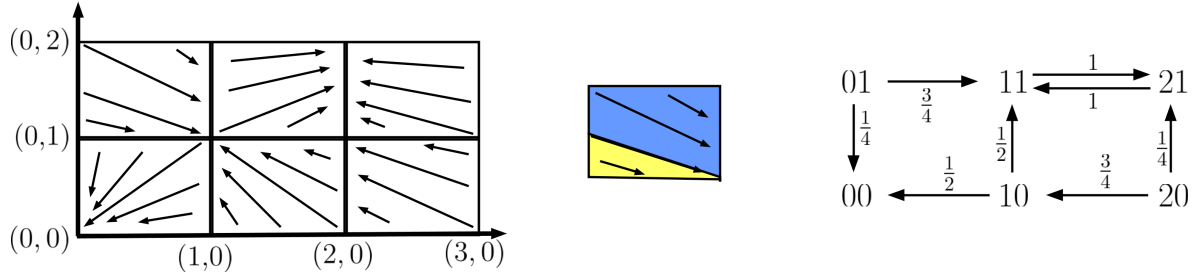


Figure 2.12: BN model with probabilities on edges.

In order to provide an example on how to obtain the probabilities values, we consider the domain $[0, 1[\times]1, 2]$, shown in the Figure 2.12 (middle). With a blue background we illustrate the states of this domain which leave it and enter the domain $]1, 2[\times]1, 2]$ and with a yellow background we illustrate the states which leave it and enter the domain $[0, 1[\times [0, 1[$. If we compute the relative areas, we can note that 75% of the region is blue and 25% is yellow. These are the values assigned to the respective edges.

We note that not every vertex has an outgoing edge (however we could include a loop with probability 1) but for the remaining, the probabilities of the outgoing edges sum to 1.

2.2.3 Asymptotic graph.

In practice, asymptotic graphs (AG) are not a model but a method. Indeed, they were used in several works [68, 11, 15] as a tool for analyzing Boolean models. However, we introduce this

kind of formalism here since it will be needed further. The basic idea for obtaining asymptotic graphs is to divide the set of variables of a BN model in two disjoint sets and study them separately. In practice, this corresponds to consider a biological system composed by two submodules. After this, one can combine the results gathered from studying each submodule separately in order to obtain information about the whole system. In practice, AG provides insights about the asymptotic behavior of the original BN model.

Consider a BN model whose set of Boolean variables is $\{x_1, x_2, \dots, x_n\}$ and is described by the Boolean equation $X^+ = F(X)$ with $X = (x_1, \dots, x_n)$, $X^+ = (x_1^+, \dots, x_n^+)$ and $F(X) = (f_1(X), \dots, f_n(X))$, where each f_i is a Boolean function. Now, separate the elements of X into two disjoint sets A and B such that $A \cup B = X$ and $A \neq \emptyset \neq B$. Thus, we can study each one of the generated “submodels”.

Let us denote the variables of A by x_{a_1}, \dots, x_{a_k} and the variables of B by x_{b_1}, \dots, x_{b_p} (note that $k + p = n$). Thus, the Boolean model for the submodule induced by the components whose variables are contained in A is defined by the Boolean equations $A^+ = F_A(f_{a_1}(A; B), \dots, f_{a_k}(A; B))$. Variables from B must be considered as inputs since they are not considered in this submodel. Similarly, the Boolean model for the other submodule, determined by the set B is given by the Boolean equations $B^+ = F_B(f_{b_1}(B; A), \dots, f_{b_p}(B; A))$ and the variables from A are the inputs of the submodel.

Given this, we can now study the submodel induced by A by considering the variables of B as inputs. In particular, this means that we must consider all possibilities for the value of the variables in B . This would be a debatable move since it would increase the space complexity. However, in practical cases, when we consider a biological system with two submodules, each submodule has a small number of components interacting with components from the other submodel. For instance, this means that a biological system with 30 components can be divided into two submodules where only the concentration of around 3 components must be considered as inputs for each submodule. Nevertheless, note that this is only possible if the modules are previously and correctly identified in the biological system.

Thus, the AG method can be viewed as an analytical tool for reducing BN models. Note that this is really important because a BN model with 20 variables describes a graph with, at least $2^{20} \approx 1000000$ states and, in practice, biological systems often present more than 20 components. The large size of BN models turns difficult its study; however the AG method can reduce them, inducing much smaller models.

Therefore, we can rewrite the BN models for both A and B as $A^+ = F_A(f_{a_1}(A; h_A(B)), \dots, f_{a_k}(A; h_A(B)))$ and $B^+ = F_B(f_{b_1}(B; h_B(A)), \dots, f_{b_p}(B; h_B(A)))$ where h_A and h_B are functions which return the set of variables from B and A , respectively, whose value must be considered as input for the submodels induced by A and B , respectively. We say that $h_B(A)$ is the output of A and $h_A(B)$ is the output of B .

Example 2.2.7. A simple example is provided to illustrate the partition of a BN into two submodels. Consider a simple toy-model in order to make it clear. In this example four variables, x_1, x_2, x_3, x_4 , are considered to take a value in $\{0, 1\}$.

$$\begin{cases} x_1^+ = x_2 \\ x_2^+ = x_1 \\ x_3^+ = \neg x_4 \\ x_4^+ = x_3 \wedge \neg x_1 \end{cases}$$

We divide it into two submodels: one considering the variables in $A = \{x_1, x_2\}$ and other considering the variables in $B = \{x_3, x_4\}$. We can now obtain two BN submodels.

Submodel A

$$\begin{cases} x_1^+ = x_2 \\ x_2^+ = x_1 \end{cases}$$

Submodel B

$$\begin{cases} x_3^+ = \neg x_4 \\ x_4^+ = x_3 \wedge \neg x_1 \end{cases}$$

Then, observe that $h_A(B) = \emptyset$ and $h_B(A) = \{x_1\}$ since the submodel induced by A has no inputs and the submodel obtained by B takes only x_1 as input.

We now introduce some concepts needed for the full construction of the asymptotic graph and explain how to obtain it. Some connections between the introduced concepts and biological problems still will not be explained at this point but further in this section.

Definition 2.2.1. Consider a directed graph (V, E) . We say that a subset $W \subseteq V$ is:

- a *strongly connected component* (SCC) if, for every $x, y \in W$, $\exists v_0, \dots, v_n \in X$ such that $x = v_1$, $v_n = y$ and $(v_{i-1}, v_i) \in E$, for $i \in \{1, \dots, n\}$.
- a *terminal*, if W is a SCC and there is no edge $e = (x, y) \in E$ such that $x \in W$ and $y \notin W$.

Using simpler words, a SCC is a set of vertices such that there is always a path between any two vertices and a terminal is a set of vertices that is a SCC and admits no outgoing edges. In the context of biological examples, we call a terminal as *attractor*.

The construction of the AG uses this concept. Consider a BN model and two submodels whose sets of variables are A and B with $|A| = k$, $|B| = p$. Furthermore, we slightly change the definition of functions h_A and h_B such that $h_A : \{0, 1\}^p \rightarrow \{0, 1\}^{\bar{p}}$ and $h_B : \{0, 1\}^k \rightarrow \{0, 1\}^{\bar{k}}$, *i.e.* given a specific value for the Boolean variables in A (respectively, B), h_A (respectively, h_B) only returns the value of the input variables for A (respectively, B). Thus, the submodel A admits \bar{p} inputs and the submodel B admits \bar{k} inputs. The first step to obtain the AG is to consider all possible combinations of values for the inputs of each submodel and construct the respective BN model for each combination. This generates a collection of Boolean graphs for each submodel, indexed by the respective input value. We denote each of these graphs by G_X^u where X denotes the set of variables considered (in this case, either A or B) and u is the input values considered (in this case, u belongs either to $\{0, 1\}^{\bar{p}}$ or $\{0, 1\}^{\bar{k}}$, according to the submodel considered).

For each graph G_X^u we must compute all attractors X_i^u , where i is a natural number used to enumerate different attractors with a random order. After doing this for all Boolean graphs, we obtain a set $Attr(A)$ and a set $Attr(B)$ containing all attractors in the submodel generated by A and B , respectively. Then, for each attractor X_i^u we must compute the semi-attractors $X_{i,v}^u$. For each attractor X_i^u of a submodel A , $x \in X_{i,v}^u$ if $x \in X_i^u$ and $h_B(x) = v$. That is, semi-attractors split attractors according to the value of their outputs. We denote by $SAttr(A)$ the set of semi-attractors of a submodel A .

The AG is a directed graph such that the set of vertices is $SAttr(A) \times SAttr(B)$. The rule to generate the set of edges is the following:

Consider a vertex $A_{i,v}^u \times B_{\bar{i},\bar{v}}^{\bar{u}}$:

- If $u = \bar{v}$ for all $b \in B_{i,\bar{v}}^{\bar{u}}$ and $\bar{u} = v$ for all $a \in A_{i,v}^u$ then there are no edges coming out from $A_{i,v}^u \times B_{i,\bar{v}}^{\bar{u}}$.
- If $u \neq \bar{v}$, one must consider the graph $G_A^{\bar{v}}$. Then, in that graph, for each admissible m, n check if there is a path from $A_{i,v}^u$ to the semi-attractor $A_{m,n}^{\bar{v}}$. If so, then the AG admits an edge from $A_{i,v}^u \times B_{i,\bar{v}}^{\bar{u}}$ to $A_{m,n}^{\bar{v}} \times B_{i,\bar{v}}^{\bar{u}}$.
- If $\bar{u} \neq v$ for some $a \in A_{i,v}^u$, one must consider the graph G_B^v . Then, in that graph, for each admissible m, n check if there is a path from $B_{i,\bar{v}}^{\bar{u}}$ to the semi-attractor $B_{m,n}^v$. If so, then the AG admits an edge from $A_{i,v}^u \times B_{i,\bar{v}}^{\bar{u}}$ to $A_{i,v}^u \times B_{m,n}^v$.

Let us follow with an example.

Example 2.2.8. Consider the toy model presented in Example 2.2.7. To obtain the AG model from it, we must build all graphs for the submodels induced by A and B by considering all possible input values for each submodel. In particular, these graphs are G_A^\emptyset , G_B^0 and G_B^1 and are illustrated in the Figure 2.13.

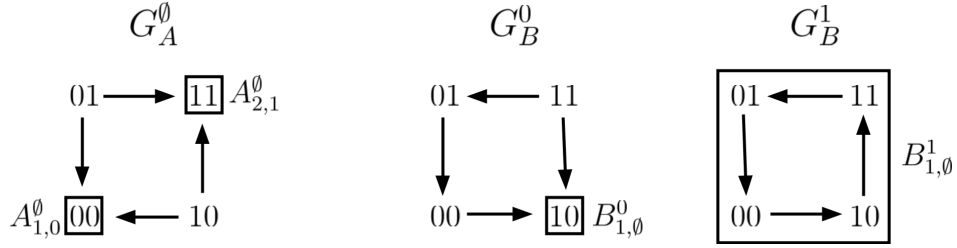


Figure 2.13: Graphs of the submodels for the admissible input values.

As mentioned before, $h_A(B) = \emptyset$ and $h_B(A) = \{x_1\}$. Moreover, the attractors found on each graph are highlighted by a black square: $A_1^\emptyset = \{00\}$ and $A_2^\emptyset = \{11\}$, for the submodel induced by A ; $B_1^0 = \{10\}$ and $B_1^1 = \{00, 01, 10, 11\}$, for the submodel induced by B . Observe that all attractors coincide with semi-attractors, *i.e.* $SAttr(A) = Attr(A)$ and $SAttr(B) = Attr(B)$.

Thus, we can obtain the asymptotic graph. Note that the set of vertices is given by $\{A_1^\emptyset, A_2^\emptyset\} \times \{B_1^0, B_1^1\}$ and the set of edges is obtained using the rule mentioned before. The full asymptotic graph is shown in Figure 2.14.

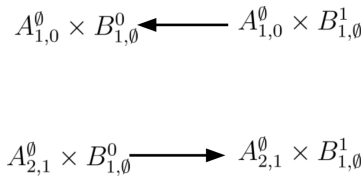


Figure 2.14: The full asymptotic graph of the system in Example 2.2.7.

We close this section about AG with the guiding example of circadian rhythm.

Example 2.2.9 (Circadian rhythm of a cyanobacteria - Asynchronous graph).

In this example, we consider the BN model for the circadian rhythm of a cyanobacteria,

already presented in Example 2.2.3. This is still a simple example with just four variables which take a value in $\{0, 1\}$.

$$\begin{cases} a^+ = \neg s \\ t^+ = a \\ ts^+ = t \wedge a \\ s^+ = ts \end{cases}$$

We divide it into two submodels: one considering variables in $A = \{a, s\}$ and other considering variables in $B = \{t, ts\}$. We obtain two BN submodels.

Submodel A

$$\begin{cases} a^+ = \neg s \\ s^+ = ts \end{cases}$$

Submodel B

$$\begin{cases} t^+ = a \\ ts^+ = t \wedge a \end{cases}$$

Note that the only input of the submodel induced by A is ts and the input for the submodel induced by B is a . To obtain the AG, we must build the graphs for the submodels G_A^0 , G_A^1 , G_B^0 and G_B^1 , as illustrated in the Figure 2.15.

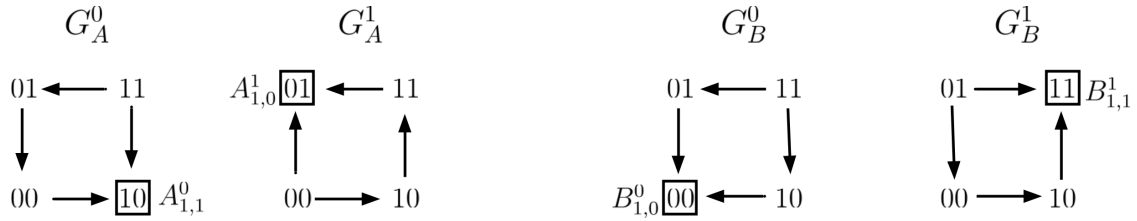


Figure 2.15: Graphs of the submodels for the AG model of circadian rhythm.

The attractors found on each graph are highlighted by a black square and, as in the previous example, the set of semi-attractors coincide with the set of attractors. Thus, we can obtain the asymptotic graph by the process mentioned above. The resulting AG is shown in Figure 2.16.

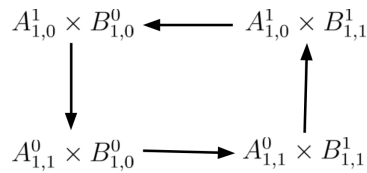


Figure 2.16: The full asymptotic graph.

The utility of AG method and how much information is preserved when compared with BN model was not discussed yet. Although we can note that the cyclic behavior of circadian rhythm system is somehow preserved by the AG in Figure 2.16, we need to introduce some biological concepts in order to have a clear idea of the utility of these models. This is done in the next section of this chapter. However, we can already note that each vertex of a AG represent a specific set of states of the original BN. Given a graph (V, E) of a BN whose

corresponding AG is (\bar{V}, \bar{E}) , we define the embedding function $\pi : \bar{V} \rightarrow 2^V$ in the trivial way. For instance $\pi(\{\{11, 10\}, \{00\}\}) = \{1100, 1000\}$. Moreover, given an attractor \mathcal{A} of a BN, it is possible to find a corresponding attractor in the obtained AG and this corresponding attractor $\bar{\mathcal{A}}$ is such that $\pi(\bar{\mathcal{A}}) := \{x : x \in \pi(a), \text{ for some } a \in \bar{\mathcal{A}}\}$. In particular, we say that each attractor of a BN model is *signaled* by an attractor of the corresponding AG because it is possible to prove that $\pi(\bar{\mathcal{A}}) \subseteq \mathcal{A}$ always hold.

2.3 Stable states in biological context.

As mentioned in the introduction, our goal is to study biological regulatory networks. In this context, an important notion is that of *stable state*. When one studies a ODE model of a biological regulatory network whose differential equation is $X' = F(X)$, the term steady state is not so strange as it refers to states x such that $F(x) = 0$, meaning that the system is at equilibrium. Moreover, these steady states can also be *stable* or *unstable*, depending on the asymptotic dynamics of the system in a neighborhood of the equilibrium state.

In a biological context, the existence of equilibria is thought to be related with the *ways of working* of cells. Indeed, most of the cells of an organism have the same genetic code. However, different cells operate in a different way. One can think, for instance, about a blood cell and a bone cell, which perform distinct functions. Thus, when studying a biological system, the identification of stable steady states or closed orbits is an important task due to its connection with differentiated cells. Depending on the kind of model, these states can take several forms, from quantitative to qualitative values. Thus, we use the term *attractor* to mention these states in different kinds of models.

ODE models are those where this connection is more intuitive. An attractor is identified as a stable equilibrium point or a stable limit cycle. However, ODE models for biological regulatory network consider systems of non linear differential equations, usually a large number of variables. This is a major drawback for its study in a symbolic way and, usually, only numerical simulations are performed. Although these simulations can provide an idea of the dynamics of the biological system, they do not guarantee a global overview of it.

Because of this, PWL models are very useful to simplify the complex dynamic of ODE models. The linearity of the differential equations within each domain allows us to obtain exact solutions. Thus, we can concatenate the flows between distinct domains in order to retrieve the general dynamics of the system. As mentioned before, it is known that not much information is lost with this simplification, however, this abstraction step may lead to some potential problems and inconsistencies as the possibility of contrary flows meeting at boundaries. The reason for these special dynamics at boundaries is, often, the existence of steady states from the initial ODE model (either stable and unstable) being located at the boundaries. This is the reason why such cases must be treated individually and with care. Anyway, it is still possible to look for attractors which are, in this context, represented by stable equilibrium points or stable limit cycles, crossing several domain. Even so, these models can still be large due to the number of components of the system.

Boolean networks are a step further into this simplification process. These models, following the asynchronous paradigm, ignore the dynamics inside each domain and only represent all possible transition between domains. Because of this, the attractors of a modeled system are represented by terminals of the BN model. It is important to highlight that some attractors can be lost when going from a PWL model to its simplified BN model, nevertheless, all

attractors obtained for a BN model *signal* an attractor in the corresponding domains of the PWL model, as discussed later. Moreover, considering BN models with probabilities in the edges we can check questions like: “Which attractor is more likely to be reached from some other domain in the model?”

Finally, a word on the AG method. As mentioned above, this method was developed to decrease the size and difficulty of studying large BN models. Indeed, although the computational power of computers have increased during last years, the complexity of the biological system is still too large. We note that humans are estimated to have more than 20000 genes (see [60]) and the size of a BN model grows exponentially with the number of variables. Thus, the AG method is an important analytical tool to analyze large biological models. The attractors of a AG are terminals, such as in BN models case. Moreover, the process to obtain a AG from a BN model guarantees that all attractors of a BN model are preserved in the corresponding AG model. However, “artificial” attractors can possibly appear, *i.e.* attractors that found no correspondent in the original BN model – they are called *spurious attractors*, in literature. While it is a good new that all attractors can be retrieved, one must check each attractor in order to verify that it is not a spurious attractor. These results about attractors are proved in [68].

Chapter 3

Logic and computational tools to study PWL models

In this chapter we present some theoretical foundations and propose computational tools for the study of PWL models. The basis for this approach relies on the *hybrid* nature of this kind of model. Note that the concept “hybrid” is used to designate dynamics which can be both continuous and discrete. PWL models are naturally hybrid since they combine the continuous dynamics within each domain with the discrete jump between domains. In this chapter we propose differential dynamic logic (see [55]) for studying these systems and show how it can be used to prove some meaningful properties. Most of the work presented in here is published in [21]. Also, some preliminary work can be found in [17].

3.1 Differential dynamic logic.

Differential dynamic logic ($d\mathcal{L}$, for short) is a dynamic logic embedding a first order logic structure and whose set of atomic programs comprises both continuous and discrete behavior. In this chapter we begin by introducing the syntax and semantics of $d\mathcal{L}$ and afterward show how to apply it to biological system. Latter, we compare related works and evaluate their interconnection. For this chapter we assume that the reader has background in dynamic logic (see e.g. [6, 36]) and first-order logic (see e.g. [63]).

3.1.1 Syntax.

The syntax of $d\mathcal{L}$ admits a set V of logical variables and a signature Σ . This set Σ contains:

- Function symbols like $+$, $-$, \cdot , $/$ and constants (functions with arity 0);
- Predicate symbols like $>$, \leq , $=$;
- State variables, which are themselves constants, but whose semantical interpretation is not fixed. We denote by Σ_{fl} the set of state variables.

Since this is a dynamic logic, modalities are induced by programs, built from a set of atomic programs and some structure over them. It is important to point out that the hybrid

dynamics of a systems can be described by this language due to the two kinds of atomic programs:

- Discrete Jump sets – $(a_1 := t_1, \dots, a_n := t_n)$ – which correspond to discrete time assignments.
- Continuous evolutions – $(a'_1 = t_1, \dots, a'_n = t_n \ \& \ \chi)$ – where the value of state variables is continuous, evolving according to a system of differential equations, constrained by an invariant condition, which is a first-order formula χ . One can omit “& χ ” whenever $\chi \equiv \top$.

Using both kinds of atomic programs we can describe both discrete and continuous events. We now follow with the definition of formulas for $d\mathcal{L}$ where this structure is implicitly and formally introduced.

Definition 3.1.1. We define the set of terms $Trm(V, \Sigma)$ as the least set such that $V \subseteq Trm(V, \Sigma)$ and $f(t_1, \dots, t_n) \in Trm(V, \Sigma)$ whenever $f \in \Sigma$ is a function with arity n and $t_1, \dots, t_n \in Trm(V, \Sigma)$.

We define the set of first order formulas $Fml_{FOL}(V, \Sigma)$ recursively as the least set that contains $p(t_1, \dots, t_n)$, \perp , \top , $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$, $\varphi \leftrightarrow \psi$, $\forall x\varphi$ and $\exists x\varphi$ for every proposition symbol $p \in \Sigma$ with arity n , every $x \in V$, every $t_1, \dots, t_n \in Trm(V, \Sigma)$ and every $\varphi, \psi \in Fml_{FOL}$.

We define the set of hybrid programs $HP(V, \Sigma)$ recursively as the least set which contains $(a_1 := t_1, \dots, a_n := t_n)$, $(a'_1 = t_1, \dots, a'_n = t_n \ \& \ \chi)$, $?\chi$, $\alpha; \beta$, $\alpha \cup \beta$ and α^* for every $t_1, \dots, t_n \in Trm(V, \Sigma)$, every $\chi \in Fml_{FOL}(V, \Sigma)$, every distinct $a_1, \dots, a_n \in \Sigma_{fl}$ and every $\alpha, \beta \in HP(V, \Sigma)$.

Finally we define the set $Fml(V, \Sigma)$ of formulas of $d\mathcal{L}$ as the least set that contains $p(t_1, \dots, t_n)$, \perp , \top , $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$, $\varphi \leftrightarrow \psi$, $\forall x\varphi$, $\exists x\varphi$, $[\alpha]\varphi$ and $\langle \alpha \rangle \varphi$ for every proposition symbol $p \in \Sigma$ with arity n , every $x \in V$, every $t_1, \dots, t_n \in Trm(V, \Sigma)$, every $\alpha \in HP(V, \Sigma)$ and every $\varphi, \psi \in Fml(V, \Sigma)$.

3.1.2 Semantics.

The definition of the semantics of $d\mathcal{L}$ states that it may only be interpreted over the set of reals. Thus, all functions and propositions must be interpreted as real functions and propositions over reals. Moreover, this interpretation is *rigid*, *i.e.* symbols like $+$ and propositions like \leq must always be interpreted as the “sum” and “less or equal”, respectively.

Given this, to evaluate a formula of $d\mathcal{L}$ we have to consider:

- An *interpretation* I , which is a (rigid) function, whose domain is $\Sigma \setminus \Sigma_{fl}$, and that interprets the predicate and function symbols as a predicate or real function.
- A *state* v , that is a map $v : \Sigma_{fl} \rightarrow \mathbb{R}$. We denote the set of all states by $Sta(\Sigma)$.
- An *assignment* η , that is a map $\eta : V \rightarrow \mathbb{R}$.

Also, let $x, d \in \mathbb{R}$ and let $f : \mathbb{R} \rightarrow \mathbb{R}$ be a function. We define $f[x \mapsto d]$ as:

$$f[x \mapsto d](y) = \begin{cases} d, & \text{if } y = x \\ f(y), & \text{otherwise.} \end{cases}$$

We firstly define a valuation function for first-order formulas because their valuation is needed to interpret the accessibility relation induced by modalities with hybrid programs. In order to simplify the notation, we also denote the function used to valuate terms with the same symbol.

Definition 3.1.2. Given an interpretation I , an assignment η and a state v , we now define the *valuation function* $val_{I,\eta}(v, \cdot) : Fml(V, \Sigma) \rightarrow \{true, false\}$ for all first-order formulas of $d\mathcal{L}$ recursively as:

- $val_{I,\eta}(v, x) = \eta(x)$, for $x \in V$;
- $val_{I,\eta}(v, u) = v(u)$, for $u \in \Sigma_{fl}$;
- $val_{I,\eta}(v, f(t_1, \dots, t_n)) = I(f)(val_{I,\eta}(v, t_1), \dots, val_{I,\eta}(v, t_n))$, for any function symbol f with arity n and any terms t_i , $i = 1, \dots, n$;
- $val_{I,\eta}(v, p(t_1, \dots, t_n)) = true$ iff $I(p)(val_{I,\eta}(v, t_1), \dots, val_{I,\eta}(v, t_n)) = true$, for any predicate symbol p with arity n and any terms t_i ;
- $val_{I,\eta}(v, \perp) = false$;
- $val_{I,\eta}(v, \top) = true$;
- $val_{I,\eta}(v, \neg\varphi) = true \Leftrightarrow val_{I,\eta}(v, \varphi) = false$, for every $\varphi \in Fml_{FOL}(V, \Sigma)$;
- $val_{I,\eta}(v, \varphi \vee \psi) = true$ iff $val_{I,\eta}(v, \varphi) = true$ or $val_{I,\eta}(v, \psi) = true$, for every $\varphi, \psi \in Fml_{FOL}(V, \Sigma)$;
- $val_{I,\eta}(v, \varphi \wedge \psi) = true$ iff $val_{I,\eta}(v, \varphi) = true$ and $val_{I,\eta}(v, \psi) = true$, for every $\varphi, \psi \in Fml_{FOL}(V, \Sigma)$;
- $val_{I,\eta}(v, \varphi \rightarrow \psi) = true$ iff $val_{I,\eta}(v, \psi) = true$ whenever $val_{I,\eta}(v, \varphi) = true$, for every $\varphi, \psi \in Fml_{FOL}(V, \Sigma)$;
- $val_{I,\eta}(v, \varphi \leftrightarrow \psi) = true$ iff $val_{I,\eta}(v, \psi) = val_{I,\eta}(v, \varphi)$, for every $\varphi, \psi \in Fml_{FOL}(V, \Sigma)$;
- For every $\varphi \in Fml_{FOL}(V, \Sigma)$, $val_{I,\eta}(v, \exists x\varphi) = true$ iff $val_{I,\eta[x \mapsto d]}(v, \varphi) = true$ for some $d \in \mathbb{R}$;
- For every $\varphi \in Fml_{FOL}(V, \Sigma)$, $val_{I,\eta}(v, \forall x\varphi) = true$ iff $val_{I,\eta[x \mapsto d]}(v, \varphi) = true$ for every $d \in \mathbb{R}$.

Before providing the definition of valuation for the full set of formulas of $d\mathcal{L}$, we must define the semantical interpretation for hybrid programs as a relation which introduces the notion of transition between states.

Definition 3.1.3. Given a hybrid program α , the induced transition relation $\rho_{I,\eta}(\alpha) \subseteq \text{Sta}(\Sigma) \times \text{Sta}(\Sigma)$ is defined recursively as:

- $(v, w) \in \rho_{I,\eta}(x_1 := \theta_1, \dots, x_n := \theta_n)$ iff $w = v[x_1 \mapsto val_{I,\eta}(v, \theta_1)] \dots [x_n \mapsto val_{I,\eta}(v, \theta_n)]$;

- $(v, w) \in \rho_{I,\eta}(x'_1 = \theta_1, \dots, x'_n = \theta_n \ \& \ \chi)$ iff $\exists r \in \mathbb{R}_0^+$ and there is a flow $f: [0, r] \rightarrow \text{Sta}(\Sigma)$, such that:
 - $f(0) = v$ and $f(r) = w$;
 - f respects the differential equations, *i.e.* for each variable x_i , the valuation of x_i at the state $f(t)$, $\text{val}_{I,\eta}(f(t), x_i) = f(t)(x_i)$ is continuous and has a derivative of value $\text{val}_{I,\eta}(f(t), \theta_i)$ in the interval $]0, r[$;
 - The value of the variables $z \in \Sigma_{fl} \setminus \{x_1, \dots, x_n\}$ remains the same during the continuous evolution, *i.e.* $f(t)(z) = v(z)$ for any $t \in [0, r]$
 - Every state $f(t)$ verifies χ , *i.e.* $\text{val}_{I,\eta}(f(t), \chi) = \text{true}$ for any $t \in [0, r]$;
- $\rho_{I,\eta}(?\chi) = \{(v, v) \in \text{Sta}(\Sigma) \times \text{Sta}(\Sigma) : \text{val}_{I,\eta}(v, \chi) = \text{true}\}$;
- $\rho_{I,\eta}(\alpha \cup \beta) = \rho_{I,\eta}(\alpha) \cup \rho_{I,\eta}(\beta)$;
- $\rho_{I,\eta}(\alpha ; \beta) = \{(u, w) : \exists v \in \text{Sta}(\Sigma) \text{ such that } (u, v) \in \rho_{I,\eta}(\alpha) \text{ and } (v, w) \in \rho_{I,\eta}(\beta)\}$;
- $\rho_{I,\eta}(\alpha^*) = \{(u, w) : \exists n \geq 0 \text{ integer, } \exists v_0, \dots, v_n \in \text{Sta}(\Sigma) \text{ such that } u = v_0, w = v_n \text{ and } (v_{k-1}, v_k) \in \rho_{I,\eta}(\alpha) \text{ for any } k \in \{1, \dots, n\}\}$.

Note that the notion of “flow” introduced in this definition coincides with the one introduced before for ODE and PWL models, in a biological context. However, in this context, the flow is constrained by a first order condition χ . Figure 3.1 illustrates these transitions by identifying states as circles and transitions induced by a hybrid program as edges.

In a discrete jump set all assignments are instantaneous and simultaneous. It is important to note that all state variables a_1, \dots, a_n must be distinct in order to the definition of these program be coherent. The same occurs for continuous evolution because one state variable cannot evolve following two distinct differential equations.

Definition 3.1.4. We introduce the valuation for all set of formulas recursively as:

- For every formula $\varphi \in Fml_{FOL}$, the valuation is as defined in Definition 3.1.2;
- $\text{val}_{I,\eta}(v, \neg\varphi) = \text{true} \Leftrightarrow \text{val}_{I,\eta}(v, \varphi) = \text{false}$, for every $\varphi \in Fml(V, \Sigma)$;
- $\text{val}_{I,\eta}(v, \varphi \vee \psi) = \text{true}$ iff $\text{val}_{I,\eta}(v, \varphi) = \text{true}$ or $\text{val}_{I,\eta}(v, \psi) = \text{true}$, for every $\varphi, \psi \in Fml(V, \Sigma)$;
- $\text{val}_{I,\eta}(v, \varphi \wedge \psi) = \text{true}$ iff $\text{val}_{I,\eta}(v, \varphi) = \text{true}$ and $\text{val}_{I,\eta}(v, \psi) = \text{true}$, for every $\varphi, \psi \in Fml(V, \Sigma)$;
- $\text{val}_{I,\eta}(v, \varphi \rightarrow \psi) = \text{true}$ iff $\text{val}_{I,\eta}(v, \psi) = \text{true}$ whenever $\text{val}_{I,\eta}(v, \varphi) = \text{true}$, for every $\varphi, \psi \in Fml(V, \Sigma)$;
- $\text{val}_{I,\eta}(v, \varphi \leftrightarrow \psi) = \text{true}$ iff $\text{val}_{I,\eta}(v, \psi) = \text{val}_{I,\eta}(v, \varphi)$, for every $\varphi, \psi \in Fml(V, \Sigma)$;
- For every $\varphi \in Fml(V, \Sigma)$, $\text{val}_{I,\eta}(v, \exists x\varphi) = \text{true}$ iff $\text{val}_{I,\eta[x \mapsto d]}(v, \varphi) = \text{true}$ for some $d \in \mathbb{R}$;
- For every $\varphi \in Fml(V, \Sigma)$, $\text{val}_{I,\eta}(v, \forall x\varphi) = \text{true}$ iff $\text{val}_{I,\eta[x \mapsto d]}(v, \varphi) = \text{true}$ for every $d \in \mathbb{R}$;

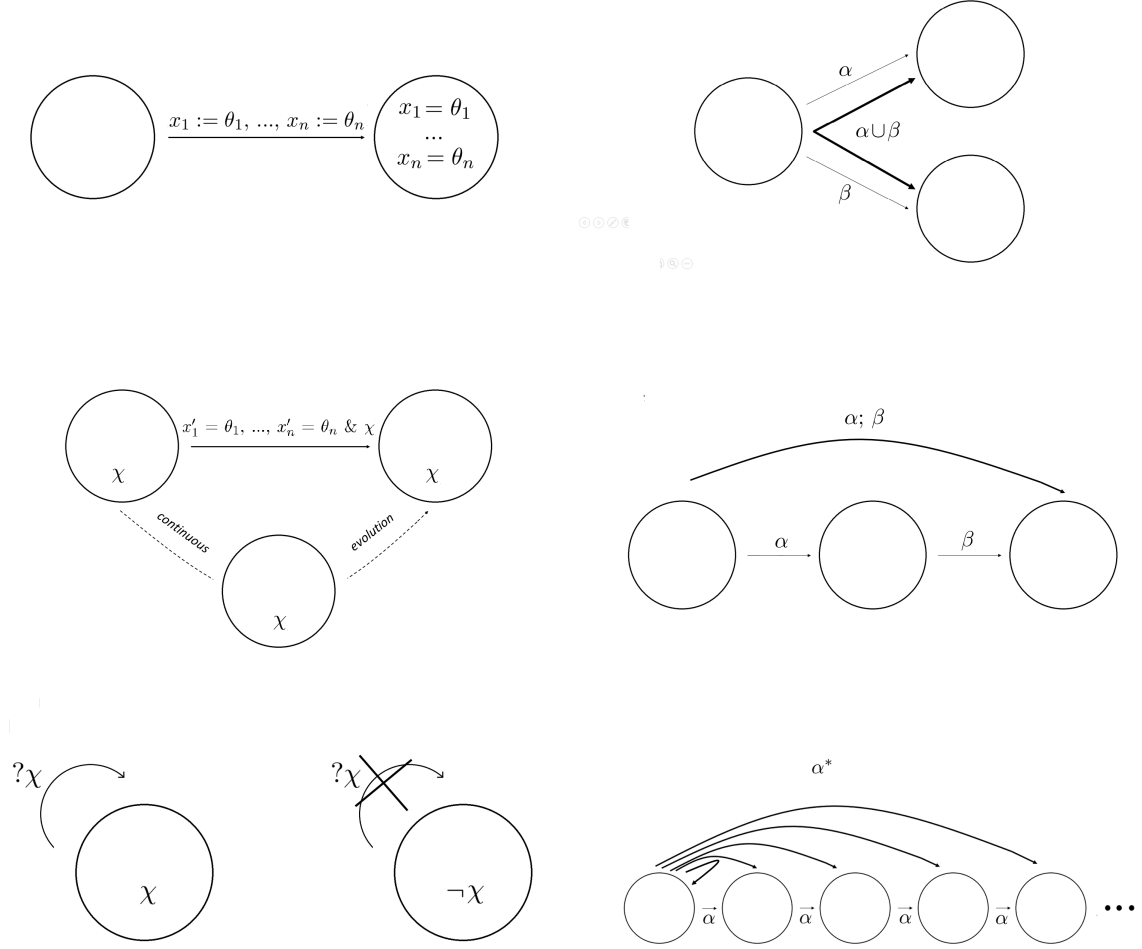


Figure 3.1: Illustration of the hybrid programs induced transitions.

- $val_{I,\eta}(v, \langle \alpha \rangle \varphi) = true$ iff $val_{I,\eta}(w, \varphi) = true$ for some state w such that $(v, w) \in \rho_{I,\eta}(\alpha)$;
- $val_{I,\eta}(v, [\alpha] \varphi) = true$ iff $val_{I,\eta}(w, \varphi) = true$ for any state w such that $(v, w) \in \rho_{I,\eta}(\alpha)$.

If $val_{I,\eta}(v, \varphi) = true$, we say that a formula φ is *satisfied* in I, η, v . Moreover, φ is said to be *valid* if it is satisfied in any triple I, η, v .

We follow with an example on how to evaluate a formula.

Example 3.1.1. Let us consider the following formula $([x_2 := 1]x_1 = 1) \rightarrow \langle x'_1 = x_1 \rangle x_1 = e$. We present a step by step evaluation of this formula for an arbitrary triple I, η, v . For this, we firstly note that, in this case, $(u, v) \in \rho_{I,\eta}(x'_1 = x_1)$ whenever $v = u[x_1 \mapsto u(x_1)e^t]$, for some $t \in \mathbb{R}^+$. This is obtained by solving the differential equation.

$$\begin{aligned}
 & val_{I,\eta}(v, ([x_2 := 1]x_1 = 1) \rightarrow \langle x'_1 = x_1 \rangle x_1 = e) = true \\
 & \Leftrightarrow val_{I,\eta}(v, \langle x'_1 = x_1 \rangle x_1 = e) = true \text{ whenever } val_{I,\eta}(v, [x_2 := 1]x_1 = 1) = true \\
 & \Leftrightarrow val_{I,\eta}(v[x_1 \mapsto v(x_1)e^t]x_1 = e) = true \text{ for some } t \in \mathbb{R}^+ \text{ whenever}
 \end{aligned}$$

$$\begin{aligned}
& \text{val}_{I,\eta}(v[x_2 \mapsto 1], x_1 = 1) = \text{true} \\
& \Leftrightarrow v[x_1 \mapsto v(x_1)e^t](x_1) = e \text{ for some } t \in \mathbb{R}^+ \text{ whenever } v[x_2 \mapsto 1](x_1) = 1 \\
& \Leftrightarrow v(x_1)e^t = e \text{ for some } t \in \mathbb{R}^+ \text{ whenever } v(x_1) = 1 \\
& \Leftrightarrow e^t = e \text{ for some } t \in \mathbb{R}^+, \text{ which is true for } t = 1.
\end{aligned}$$

In particular, we note that, since this evaluation was for an arbitrary tripe I , η and v , then this formula is valid.

It is important to note that the syntax of $d\mathcal{L}$ can be applied to describe usual properties of systems such as reachability. For instance, we can express that a specific state is reachable from another specific state. Actually, this can even be generalized to regions. In fact, we can express the property “regarding a system of differential equations such that $x' = 1 - y$ and $y' = x$, we can reach a state where $x = y$ starting at $(x, y) = (0, 0.1)$ without crossing any state where $x < 0$ or $y < 0$ ” by the formula:

$$\langle x := 0, y := 0.1; (x' = 1 - y, y' = x \ \& \ x \geq 0 \wedge y \geq 0) \rangle x = y$$

One of the main novelties of $d\mathcal{L}$ is the capacity of being able to describe hybrid dynamics. Thus, one of the most important things when trying to describe a property of a system is to describe its dynamics by an hybrid program. Afterward, this hybrid program can be used to index modalities. We follow with an example in order to illustrate how such hybrid programs can be obtained.

Example 3.1.2 (Bouncing Ball from [55]). In Figure 3.2 we can see that the dynamics of the ball in the air follows the laws of physics where the acceleration of the ball (derivative of the velocity v) is $-g$, the gravitational acceleration. The variable h is the height, which depends on the velocity of the ball, and t is the running time. In the instant the ball hits the ground, the velocity is instantaneously updated. At this point, the velocity becomes $-cv$ where the parameter c is an elasticity constant depending on the ball.

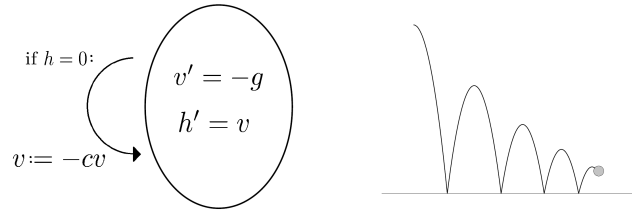


Figure 3.2: Bouncing ball example.

Neither ODEs nor discrete models are able to fully describe the dynamics of such a system but the syntax of $d\mathcal{L}$ is able to do so. In order to do this, we must describe the behavior of the bouncing ball by an hybrid program of $d\mathcal{L}$.

The continuous evolution of the ball in the air is described by the ODE $v' = -g$ and $h' = v$. Thus, we must describe it by the program $(v' = -g, h' = v \ \& \ h \geq 0)$. We note that the continuous evolution is constrained by the condition $h \geq 0$ in order to assure that the ball does not go “below” the ground. Also, when the ball hits the ground we must describe its behavior by a discrete assignment $v := -cv$. Here, we can either consider that c has a fixed value or that it is a free logical variable. In order to be more general, in this example,

we consider c a free logical variable. Then, we can describe the dynamics of bouncing ball by the following hybrid program:

$$((v' = -g, h' = v \ \& \ h \geq 0); ((?(h = 0); (v = -cv)) \cup (?(h \neq 0))))^*$$

This program admits a continuous evolution while $h \geq 0$ and, after that, the execution checks if $h = 0$. If so, it executes the discrete assignment to the velocity v . This procedure is repeated finitely many times in order to allow the ball to bounce arbitrary finitely many times. We note that, in this case we do not really need the constrain into the ODE. Indeed, we can sometimes find an equivalent hybrid program to describe the dynamics of the system:

$$((v' = -g, h' = v); ((?(h = 0); (v = -cv)) \cup (?(h > 0))))^*$$

Actually, this is possible because anytime $h < 0$ after the execution of the continuous evolution, the program will not be able to successfully terminate because it will fail either the test $?(h = 0)$ or $?(h > 0)$ and the execution will fail.

3.1.3 Proof calculus.

As expected, $d\mathcal{L}$ admits a proof calculus. This calculus uses sequents which are expressions with the form $\Phi \vdash \Psi$. In this representation we call to the formulas in Φ the *antecedents* and to the formulas in Ψ the *consequents*. We have the following useful equivalence:

$$\Phi \vdash \Psi \equiv \vdash \bigwedge_{\varphi_i \in \Phi} \varphi_i \rightarrow \bigvee_{\psi_i \in \Psi} \psi_i$$

We do not present the rules of this proof calculus in this thesis because they are not fundamental for the work developed. They can be found in [55], as well as its basic properties. The proof calculus of $d\mathcal{L}$ is sound but incomplete, *i.e.* although all provable formulas are valid, there are valid formulas which are not provable. This fact is a drawback, however, we can work around this problem. Indeed, in [55] a weaker result about completeness is proven. It states that the proof calculus of $d\mathcal{L}$ is complete if all valid formulas of the form $[\alpha]\varphi$, with $\alpha \in HP(V, \Sigma)$ being a continuous evolution and $\varphi \in Fml_{FOL}(V, \Sigma)$, are known.

The process for verifying the validity of a formula is algorithmic. At each step, the algorithm considers a formula φ and obtains one or more simpler formulas which would be enough to derivate φ using the proof calculus of $d\mathcal{L}$. Then, one tries to prove each one of these simpler formulas. Thus, this is a recursive process which finishes when we eventually reach an axiom or some simpler formula which we are not able to prove. Since this process is long and costly, it is useful to have a tool to help us in this process. This tool, which was also developed by A. Platzer is a software called KeYmaera. It helps to verify the validity of a $d\mathcal{L}$ formula by producing its proof in a semi-automatic way. In practice, the software performs all algorithmic steps and asks for human help whenever it is not able to algorithmically check the validity of some simpler formula. More details about KeYmaera software can be found in [55].

Since this proof calculus is incomplete, it can be the case that we are not able to prove some valid formulas of $d\mathcal{L}$ even using KeYmaera software. Thus, when a proof is incomplete and KeYmaera asks for human help, the user can check which formula is causing the problem. This is very useful because the formula returned by KeYmaera is, in fact, a condition for the validity of the input formula for KeYmaera. This means that, if we provide a formula φ to

KeYmaera and it returns some formulas $\varphi_1, \dots, \varphi_n$ that it is unable to prove, then it means that, anyway, we have a proof for the formula $(\varphi_1 \wedge \dots \wedge \varphi_n) \rightarrow \varphi$.

This fact provides some interesting strategies for obtaining sufficient conditions. If one is studying a system and wants to obtain sufficient conditions for some event to happen, one can describe the desired event by a $d\mathcal{L}$ formula and use it as input of KeYmaera. Thus, KeYmaera will check whether the desired property is valid (always hold) or provide some sufficient condition for it to happen. For example, this is very useful when studying safety conditions for safety-critical systems (see [45, 56, 57]).

3.2 Application to PWL models: Biological systems with hybrid dynamics.

The occurrence of hybrid dynamics in biochemical systems is common. In fact, we can consider several scenarios where it is reasonable to study biological systems under a hybrid perspective. In a medical context, this is observed in cases like the use of an insulin pump or a pacemaker, where the continuous¹ dynamics of the organism is, almost instantaneously, affected by an external input. Indeed, several approach in a therapeutic already consider a discrete controller (for example in the case of some electric stimulations or drug administration). Also at a smaller scale, this can be done in a cellular context with microorganism or even cells (for instance, manipulating the environment). Combining these discrete controllers with the natural continuous dynamics originates hybrid dynamics.

A related concept that cannot be ignored is *reconfigurability* (see [50]). This notion refers the capacity of some systems to change the way they operate while running. Although it can be used at a wider context, we note that hybrid systems can be seen as reconfigurable systems where discrete jumps are seen as reconfigurations. Thus, the change in their behavior is caused by some discrete input to the system. There are references to several biological reconfigurable systems. De Silva presents in his book [64] many examples of reconfigurable biological systems. Also, in [51] an insulin infusion pump motivates the study of a reconfigurable design.

3.2.1 Relating hybrid and reconfigurable systems.

In the literature, several approaches can be found to the study of reconfigurable systems. Regarding reconfigurations as transitions, we may propose some sort of modal logic as the language to express them. However the methods must be prepared to deal with whatever mathematical structures (and corresponding logics and languages) are used to formally describe local configurations of a system. Madeira in [50] presents the mathematical foundations of a (family of) logic(s) to reason about reconfigurable systems, modeled as generalized transition systems. In these structures, transitions represent the changes between configurations and each world of the structure is a model of a specific configuration. It is important to refer that a logic common to every state of the transition system must exist. This is precisely the approach we follow in this document. Hence, our approach can be described in a rather straightforward way: models for reconfigurable systems software are structured transition

¹We note that, in fact, the number of each kind of component is a natural number and the interactions between components follow stochastic laws. However, as mentioned before, due to their large number, it is reasonable to treat them as continuous variables.

systems specified by an appropriate logical system. Their states are the individual configurations with whatever structure they need to have to model concrete applications. On the other hand, transitions correspond to the admissible reconfigurations that can be performed in the real world.

There are several ways in which a system can be reconfigurable: discrete jumps imposed by physical constraints, external stimuli, or discontinuities in vector fields are some common occurrences. To identify reconfigurability in biological models, we thus propose a definition that uses the notion of *discrete event* [9]. A discrete event triggers a change in the current continuous system: it can be caused by a variable constraint ($h = 0$ in Example 3.1.2) or by a discrete input with possible values. Each discrete event induces a transition from the current *configuration* to another one: in general, there may be more than one transition allowed for each event. Formally, a reconfigurable system is defined as follows.

Definition 3.2.1 (Reconfigurable System). A reconfigurable system is a tuple $(X, \mathcal{V}, \mathcal{M}, \mathcal{Q})$ of nonempty sets such that:

- X is a nonempty set of *state variables*;
- \mathcal{V} is a nonempty *evaluation set*;
- \mathcal{M} is the set of *configurations* $M : D_M \times \mathbb{T} \rightarrow \text{Sta}(X)$, where
 - \mathbb{T} is an additive monoid such as \mathbb{R}_0^+ (continuous case) or \mathbb{Z}_0^+ (discrete case).
 - $\text{Sta}(X)$ is the set of all *states* $v : X \rightarrow \mathcal{V}$. In case $X = \{x_1, \dots, x_n\}$, we can identify a state v with the tuple $(v(x_1), \dots, v(x_n))$;
 - $D_M \subseteq \text{Sta}(X)$ is the *invariant* of the configuration M ;
 - $M(v, 0) = v$, for any state $v \in D_M$;
 - $M(M(v, t_0), t_1) = M(v; t_0 + t_1)$ for any state v and every $t_0, t_1 \in \mathbb{T}$;
- \mathcal{Q} is a set of *discrete events* or *reconfiguration* (relations) $Q \subseteq (\mathcal{M} \times \text{Sta}(X))^2$, where:
 - if $((M_1, v_1), (M_2, v_2)) \in Q$, then $v_1 \in D_{M_1}$ and $v_2 \in D_{M_2}$;
 - no reflexive pair is contained in Q , *i.e.* with the form $((M, v), (M, v))$.

We say that (v_0, v_1) is an *admissible M -evolution from v_0 to v_1* within the configuration M if $\exists \bar{t}, M(v_0, \bar{t}) = v_1$ and such that $M(v_0, t) \in D$ for all $t \in [0, \bar{t}]$.

Each configuration M is such that $M(v, t)$ represents the state reached from v after t units of time, according to the respective configuration. The condition $M(M(v, t_0), t_1) = M(v; t_0 + t_1)$ guarantees the coherence of each configuration. In the examples discussed here, the configurations will be given using ODEs and discrete updating functions. Thus, when the dynamics within M is ruled by differential equations, admissible M -evolutions coincide with flows. Moreover, (i) reconfigurations which are not triggered at the same configurations are distinct and (ii) reconfigurations which do not lead to the same configuration must be also distinct.

A discrete event $Q = ((A, v), (B, v')) \in \mathcal{Q}$ expresses that it is possible to jump from the state v evolving according to the configuration A to the state v' evolving according to the configuration B by the discrete event Q . Note that no pair with the form $((M, v), (M, v))$ is contained in any discrete event because, since these states changes occur instantaneously,

this kind of pair does not represents any reconfiguration at all. The most tricky part is to identify the appropriate set \mathcal{Q} and the corresponding discrete events Q . The following examples illustrate this.

As mentioned, we can consider a hybrid system as a particular reconfigurable system. This is because the continuous system admits an instantaneous (discrete time) reconfiguration. Bearing this in mind, we can also apply $d\mathcal{L}$ to the study of such systems. In [38], hybrid systems are formalized using the concept of hybrid automaton. Therefore, we use that definition of hybrid automata and explain why they can be considered as a particular case of reconfigurable systems.

Definition 3.2.2 (Hybrid Automata). A hybrid automaton is a tuple $(\mathcal{M}, E, \Sigma, X, init, inv, dyn, asg, grd)$ where:

- X is a finite set of real-valued variables.
- \mathcal{M} is a finite set of discrete states, E is a transition relation $E \subseteq \mathcal{M} \times \Sigma \times \mathcal{M}$ and Σ a set of labels.
- $init$ and inv are functions which associate to each state a predicate over the variables in X . Letter \mathcal{D} denotes the set $\{(M, v) \in \mathcal{M} \times \mathbb{R}^{|X|} : v \models inv(M)\}$ where the expression $v \models inv(M)$ means that the predicate $inv(M)$ is satisfied in v .
- dyn is a function that associates to each state a predicate over the variables in X and their first derivatives. It is used to define the set of continuous evolutions which occur at each state (system of ODEs).
- asg is a function that, given an edge $(e = (M_1, l, M_2) \in E)$, returns a predicate over v_0, v_1 , states of M_0 and M_1 , respectively, after a discrete jump. This provides an assignment to each edge. Finally, the function grd associates each edge with a guard, i.e. a predicate over X .

We can see that this family of hybrid automata is a reconfigurable system where:

- The set X is the set of state variables.
- The set \mathcal{V} is \mathbb{R} .
- Each state $M \in \mathcal{M}$ represents a configuration given by the solution of the system of ODEs obtained with $dyn(m)$ and its invariant D_M is the set $\{v : (M, v) \in \mathcal{D}\}$ and, therefore, $\mathbb{T} = \mathbb{R}_0^+$. In this way $M(v_0, t)$ is the state which assigns to each state variable the value of the respective solution for the Cauchy problem with the ODEs obtained from $dyn(M)$ and initial state v_0 , after t units of time.
- The set \mathcal{Q} of discrete events is given by the set containing all discrete events Q defined for each $l \in \Sigma$ and each pair $(M_0, M_1) \in \mathcal{M} \times \mathcal{M}$ as:
 - $\{((M_0, v_0), (M_1, v_1)) : e = (M_0, l, M_1) \in E, \text{ the predicate } asg(e) \text{ is valid over } v_0, v_1 \text{ and the predicate } grd(e) \text{ is true over } v_0\}$.

Finally, we note that the hybrid automaton consider $init$ as the possible initial states and values for the state variables. Although possible, we did not consider such set in our definition of reconfigurable systems since we will not use it.

The bouncing ball is a classical example of a hybrid system and we present it to illustrate how to consider it as a reconfigurable system.

Example 3.2.1 (Bouncing Ball). A bouncing ball can be seen as a reconfigurable system and, as explained above, as a system with hybrid dynamics. Anytime the ball hits the ground and bounces back there is a change in the differential equations which drives the evolution of the position of the ball along time. Since this change occurs at discrete time instants we

observe a reconfiguration at those instants. We recall the Figure 3.2 in order to describe the bouncing ball as a reconfigurable system $(X, \mathcal{V}, \mathcal{M}, \mathcal{Q})$ where:

- $X = \{v, h\}$
- $\mathcal{V} = \mathbb{R}$
- $\mathcal{M} = \{M\}$ is a singleton and $\mathbb{T} = \mathbb{R}_0^+$. For each t , $M((v_0, h_0), t)$ is the solution for the Cauchy problem: $v' = -g \wedge h' = v$, with initial values (v_0, h_0) at t . The invariant $D_M = \{(v, h) \mid h \geq 0\}$.
- $\mathcal{Q} = \{Q\}$ is a singleton such that $Q = \{((M, (\bar{v}, 0)), (M, (-c\bar{v}, 0))) \mid \bar{v} < 0\}$.

Now we focus on how to use d \mathcal{L} proof calculus to study and prove properties of hybrid systems. In order to do this, we recall the hybrid program obtained in Example 3.1.2 for the bouncing ball using the syntax of d \mathcal{L} . Considering H as the initial height and assuming the law relating kinetic and potential energy – $v^2 \leq 2g(H - h)$ – we can prove that the height h of the ball will always be between 0 and the initial height H . Figure 3.3 shows the initial screen of KeYmaera for the proof of this property. Indeed, we only need to assume that the gravitational acceleration g is positive, the initial height H is positive, and that the value of c , an elasticity constant, is between 0 and 1.

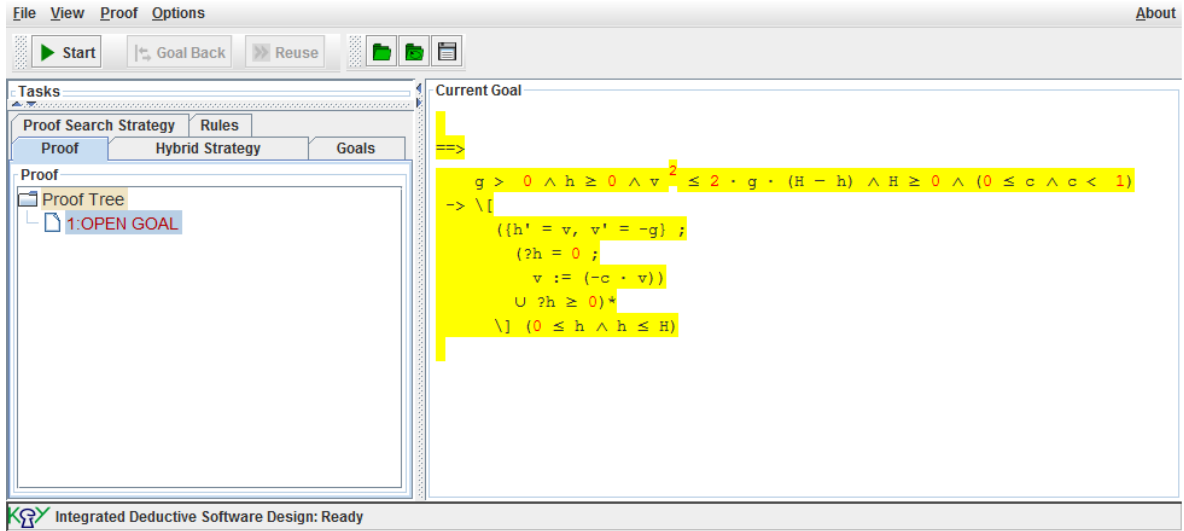


Figure 3.3: Initial screen of the bouncing ball problem in KeYmaera.

We note that, in this example, the exact values of g (gravitational acceleration) and c (elasticity constant) were not specified. In fact, d \mathcal{L} proof calculus is able to prove properties and formulas containing variables (unlike, for example, model checking). References [49, 55, 56, 57] collect many other examples essentially from mechanics.

3.2.2 PWL models as hybrid models.

In this document, we focus our attention in PWL models for biological systems as hybrid systems. As illustrated next, PWL models appear naturally as reconfigurable structures.

Example 3.2.2. Consider a PWL model where

$$\mathcal{D} = \left\{ [0, \theta_1^{a_1}[,]\theta_1^{a_1}, \theta_2^{a_1}[, \dots,]\theta_{n_{a_1}}^{a_1}, \frac{k_1}{\gamma_1}] \right\} \times \dots \times \left\{ [0, \theta_1^{a_n}[,]\theta_1^{a_n}, \theta_2^{a_n}[, \dots,]\theta_{n_{a_n}}^{a_n}, \frac{k_n}{\gamma_n}] \right\}$$

is the set of all domains. Also, let $F^D(x_1, \dots, x_n)$ be system of differential equations associated with the domain $D \in \mathcal{D}$.

This general class of systems can be interpreted as a class of reconfigurable systems with $X = \{x_1, \dots, x_n\}$ being the set of state variables, $\mathcal{V} = \mathbb{R}$ and the set \mathcal{M} containing the configurations M^D , relative to each domain D . $M_D : \bar{D} \times \mathbb{R} \rightarrow Sta(X)$ is such that $M_D(x_0, 0) = x_0$ and $\frac{dM_D(x,t)}{dt} = f_D(x)$. The set of discrete events \mathcal{Q} is defined with respect to boundaries between adjacent domains. When the flows of two adjacent domains can be concatenated – *i.e.* they do not move away nor to the boundary simultaneously – then a discrete event $Q = ((D, v), (D', v))$ is in \mathcal{Q} whenever D and D' are adjacent and v is in the boundary between D and D' .

As mentioned above, there may be some special cases (such as sliding dynamics or fixed points along the boundaries) where a transition is not so straightforward to compute. However, this does not affect reconfigurability: depending on the case, a new configuration corresponding to the sliding mode or fixed point might be added to \mathcal{M} .

Following this idea, we can use the syntax of $d\mathcal{L}$ to specify the dynamics of PWL models. This is done using the hybrid language of $d\mathcal{L}$: the differential equations specify the dynamics within each domain and the discrete jump sets along with the remaining operators of dynamic logic are used to specify the changes between domains described by the discrete events. A given system may exhibit several reconfigurability agents. For instance, a discrete controller can be also considered in a PWL model, generating a new class of discrete events.

In this case, given a set X of state variables, the PWL model obtained is similar but each f_D depends on external parameters, $\mu = (\mu_1, \dots, \mu_r)$, taking values in a discrete set: $\mu \in U \subseteq \mathbb{R}^r$ and $f_D : \bar{D} \times U \rightarrow Sta(X)$. Thus, a broader set of configurations and additional discrete events must be considered. Let ℓ be the number of domains and note that, at each domain of a PWL model, we have a system of linear equations with inputs:

$$X' = f_D(X; \mu), \text{ for } X \in D, \mu \in U$$

If we consider $u = |U|$, then there are $\ell \times u$ configurations $M_{D,\mu}$ associated to the system of linear differential equations $f_D(X; \mu)$.

The set of discrete events contains all elements for transitions between boundaries, as for usual PWL models, and some additional elements for each change in the input value. Thus, the set \mathcal{Q} of discrete event contains the elements of type:

- (1) $((M_{D,\mu}, v), (M_{D',\mu}, v))$ where D and D' are adjacent and v belong to the boundary between D and D' ;
- (2) $((M_{D,\mu}, v), (M_{D,\mu'}, v))$ for every state u , *i.e.* whenever the value of any input variable changes.

Here, we note that: in (1), we still must be careful with sliding modes and; in (2), discrete controllers may be constrained by some rule or strategy – for instance, a pacemaker should only be triggered at specific points. This is indeed intrinsically described in the structure of reconfigurable system by the definitions of discrete events and invariants. In this way, the dynamics of a PWL model with inputs is a reconfigurable system and can also be described by hybrid programs in the syntax of $d\mathcal{L}$. This is illustrated in the next example.

Example 3.2.3 (PWL model with inputs).

Let us consider a theoretical regulatory network composed of two components which regulates each other as well as themselves. Also, consider an external discrete controller μ to regulate the expression of y . In a practical case μ can be, for instance, mRNA:

$$\begin{cases} x' = 5 \frac{y^m}{2^m + y^m} + \frac{x^m}{3^m + x^m} - x \\ y' = 5 \frac{3^m}{3^m + x^m} + \frac{y^m}{y^m + 2^m} + \mu - y. \end{cases}$$

To simplify, we consider, in the sequel, that the degradation rate of the substance represented by μ is extremely high and is reduced to zero quickly if the control fades.

The corresponding piecewise linear model is the one presented in Table 3.1, according to its four domains labeled B_i , $i = 1, \dots, 4$ with $B_1 = \{0 \leq x < 3, 0 \leq y < 2\}$, $B_2 = \{0 \leq x < 3, 2 < y \leq 6\}$, $B_3 = \{3 < x \leq 6, 2 < y \leq 6\}$, and $B_4 = \{3 < x \leq 6, 0 \leq y < 2\}$.

$\begin{cases} x' = 5 - x \\ y' = 6 + \mu - y \end{cases}$ $0 \leq x < 3 \wedge 2 < y \leq 6$	$\begin{cases} x' = 6 - x \\ y' = 1 + \mu - y \end{cases}$ $3 < x \leq 6 \wedge 2 < y \leq 6$
$\begin{cases} x' = -x \\ y' = 5 + \mu - y \end{cases}$ $0 \leq x < 3 \wedge 0 \leq y < 2$	$\begin{cases} x' = 1 - x \\ y' = \mu - y \end{cases}$ $3 < x \leq 6 \wedge 0 \leq y < 2$

Table 3.1: Piecewise linear model with a discrete control variable μ .

This model presents two forms of reconfigurable behavior. Reconfigurations occur whenever the system transits between domains or whenever the discrete input μ changes its value. In this example we define the possible discrete values for the input $\mu \in \{0, 2\}$.

A careful analysis of the model in Table 3.1 shows that, in the case inputs have no effect – *i.e.* $\mu \equiv 0$ –, the system asymptotically converges to an orbit centered at $(x, y) = (3, 2)$ as shown in Figure 3.4.

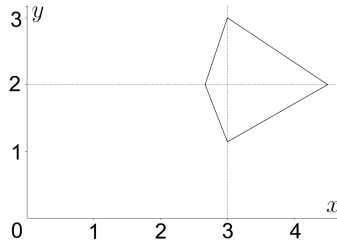


Figure 3.4: Orbit of the system.

However, suppose that the objective is to drive the system toward a state where both proteins are highly expressed, that is, the system remains in a state which is at domain B_3 .

This can be done if we set the control as $\mu = 2$ whenever $x \geq 3$ and $y \geq 2$, and $\mu = 0$ otherwise. Indeed, we can use $d\mathcal{L}$ to prove that the resulting system has a stable steady state at $(x, y) = (6, 3)$. In order to be able to do this, we need to describe this property by a formula of $d\mathcal{L}$ syntax. Firstly, we must obtain the hybrid program that describes the dynamics of this system in the semantics of $d\mathcal{L}$. Since there are four domains, we describe the evolution on each, as different configurations. Note that in this case, no complex dynamics appears at the boundaries. We also introduce a time counter τ whose purpose will be shown further:

$$ctrl_1 \equiv (?x \leq 3 \wedge y \leq 2; \mu := 0; (x' = -x, y' = 5 - y + \mu, \tau' = 1 \ \& \ x \leq 3 \wedge y \leq 2))$$

$$ctrl_2 \equiv (?x \leq 3 \wedge y \geq 2; \mu := 0; (x' = 5 - x, y' = 6 - y + \mu, \tau' = 1 \ \& \ x \leq 3 \wedge y \geq 2))$$

$$ctrl_3 \equiv (?x \geq 3 \wedge y \leq 2; \mu := 0; (x' = 1 - x, y' = -y + \mu, \tau' = 1 \ \& \ x \geq 3 \wedge y \leq 2))$$

$$ctrl_4 \equiv (?x \geq 3 \wedge y \geq 2; \mu := 2; (x' = 6 - x, y' = 1 - y + \mu, \tau' = 1 \ \& \ x \geq 3 \wedge y \geq 2))$$

For instance, the execution of program $ctrl_1$ first verifies if the actual state is in the respective domain B_1 , defined by $x \leq 3 \wedge y \leq 2$; if so, then it executes $\mu := 0$ and proceeds with the continuous evolution defined by the respective system of differential equations. In the system of differential equations, we add a differential equation capturing to the time counter τ . Also, this evolution can only be executed while the system is still in the same domain (this constraint is represented by “ $\& \ x \leq 3 \wedge y \leq 2$ ”). The executions of $ctrl_2$, $ctrl_3$ and $ctrl_4$ are analogous.

Then, the evolution of the complete system is described by:

$$bioctrl \equiv (ctrl_1 \cup ctrl_2 \cup ctrl_3 \cup ctrl_4)^*$$

The execution of $bioctrl$ can determine which subprogram must be executed. In order to consider behaviors that come from one domain to another, the Kleene operator is added, for (finite) iteration.

If a state is a steady state, then there exists a neighborhood such that any other state inside that neighborhood converges asymptotically to the steady state. In particular, we can conclude that some state is a steady state if there is a disk centered in that state such that if we choose another state x inside it, the distance of this state to the center will decrease along the trajectory obtained by the evolution of the system. We can describe this property by the following formula of $d\mathcal{L}$:

$$\begin{aligned} \varphi \equiv \exists c > 0 (\forall 0 < k < c((x - 6)^2 + (y - 3)^2 = k \wedge \tau = 0 \\ \rightarrow [bioctrl](\tau = 0 \vee (x - 6)^2 + (y - 3)^2 < k))) \end{aligned}$$

where c and k are logical variables and τ, x, y and u are state variables.

We point out that the time counter τ is important here because, by definition, the hybrid program $bioctrl$ can terminate without executing the continuous evolution. In this case, the state would not vary and the distance to the stable steady state $(x, y) = (2, 3)$ would not decrease. Therefore, after the execution of $bioctrl$, either no continuous evolution occurred (and, thus, $\tau = 0$) or the distance to the state $(x, y) = (2, 3)$ decreased.

We must note that, in general, there are asymptotic stable steady states which do not verify the property above. Indeed, in a general case, the distance can increase during some periods of time along the trajectory to an attractor. However, since we are considering a piecewise linear models, we know that these phenomena do not occur.

In fact, we can construct a proof of φ within the $d\mathcal{L}$ proof calculus to show that the property above holds, as reported in [17].

Next example presents the ODE model for circadian rhythm of a cyanobacteria with a discrete input variable and use $d\mathcal{L}$ syntax to describe some properties.

Example 3.2.4 (Cyanobacteria circadian rhythm with discrete controller).

Recover the ODE model introduced in Example 2.1.1 for the circadian rhythm of a cyanobacteria:

$$\begin{cases} x'_a = 10 \frac{5^4}{x_s^4 + 5^4} - 0.45x_a \\ x'_t = 20.51 \frac{(205.43 - x_t - x_{ts} - x_s)^4}{(205.43 - x_t - x_{ts} - x_s)^4 + 29.95^4} \cdot \frac{x_a^4}{x_a^4 + 10^4} - 0.24x_t \\ x'_{ts} = 10.74 \frac{x_t^4}{x_t^4 + 11.42^4} \cdot \frac{x_a^4}{x_a^4 + 10^4} - 0.28x_{ts} \\ x'_s = 6.61 \frac{x_{ts}^4}{x_{ts}^4 + 10.16^4} \cdot \frac{13^4}{x_a^4 + 13^4} - 0.081x_s \end{cases}$$

As it is this model does not describe a reconfigurable system since it can be described by a single configuration. Nevertheless, it is possible to obtain hybrid dynamics when considering a discrete control variable u . Let u be a control which induces the production of the protein KaiA whose concentration is represented by x_a (for instance, mRNA). In order to include this variable we change the differential equation associated to x_a to $x'_a = 10 \frac{5^4}{x_s^4 + 5^4} + u - 0.45x_a$. Moreover, we consider an additional differential equation with the following control strategy:

$$u' = \begin{cases} 1 - u, & \text{if } x_a \leq 15 \\ -u, & \text{otherwise.} \end{cases}$$

Here, the term “ $-u$ ” describes the degradation rate for the control substance (which can be mRNA, for instance).

Let $X = (x_a, x_t, x_{ts}, x_s, u)$ and $F : \mathbb{R}^5 \rightarrow \mathbb{R}^5$ such that $F(X) = (F_1, F_2, F_3, F_4, F_5)$, where F_i corresponds to the i^{th} equation of the ODE model with the control u , $i \in \{1, 2, 3, 4, 5\}$. We can represent our model by the expression $X' = F(X)$. Additionally, if we introduce an additional state variable a such that $F_5 = a - u$, where a can be either 0 or 1, according to the control strategy, we can describe the dynamics of this model by a hybrid program of $d\mathcal{L}$. To be realistic, a discrete controller would check the value of x_a at regular intervals in order to decide which action to take. Let us consider that the discrete controller checks the state of the model every 0.1 units of time. We can describe the dynamics of this model by the following hybrid program, which we denote by *hybdpgrm*:

$$\begin{aligned} subpgrm \equiv & \left(\tau := 0; ((?x_a < 15); a := 1 \cup (?x_a \geq 15); a := 0); \right. \\ & \left. (\tau' = 1, X' = F(X) \ \& \ \tau \leq 0.1) \right) \end{aligned}$$

$$hybdpgrm \equiv subpgrm; ((\tau = 0.1); subpgrm)^*$$

The hybrid program denoted by *subpgrm* starts by considering a time counter state variable τ which will be needed further. Then it verifies if either $x_a < 15$ or $x_a \geq 15$ and adjusts the value of a , accordingly. Thereafter, it runs the continuous evolution constrained by $\tau \leq 0.1$ in order to guarantee that the continuous evolution will not run longer than what is allowed between two successive verifications by the discrete controller.

Hence, the program *hybdpgrm* executes the program *subpgrm* successively. It begins by executing it once. Afterward, it verifies if the discrete controller must alter the value of a (this is known by performing the test $?\tau = 0.1$). If so, it runs *subpgrm* again. We introduce the operator $*$ to obtain a hybrid program which successfully terminates on all reachable states of the original model.

Now that we have a hybrid program describing the evolution of the hybrid model (ODE with discrete controller), we can test formulas expressing properties we want to prove. For instance, we can ask if, for the initial point $X = (17, 10, 10, 10, 0)$, it is true that x_a is always greater than 13. This property is expressed by the formula:

$$?(x_a = 17 \wedge x_t = 10 \wedge x_{ts} = 10 \wedge x_s = 10 \wedge u = 0) \rightarrow [\textit{hybdpgrm}]x_a > 13$$

3.2.3 Limitations and alternatives.

As mentioned, the main advantage of $d\mathcal{L}$ is to have a sound proof calculus. However, when one is proving a formula containing a modality with continuous evolutions – given as systems of ODEs –, the proof calculus of $d\mathcal{L}$ requires the analytical solution of those systems of ODEs in order to formally prove the formula.

This can be problematic since some differential equations do not admit an analytical solution or are very hard to solve. In particular, one can think about systems of nonlinear differential equations. Although $d\mathcal{L}$ is able to specify properties of a system by a formula, sometimes that formula cannot be formally proven. In fact, even KeYmaera is not able to deal with most of these problematic cases. KeYmaera calls (Wolfram) Mathematica in order to obtain a symbolic solution for the system of ODE, which is not always possible.

Consider, for example, the final formula in Example 3.2.4. Although there is no problem specifying the system, KeYmaera would not be able to prove that formula since it cannot handle the system of nonlinear differential equations which guide the dynamics of the ODE model for circadian rhythm.

Furthermore, KeYmaera is particularly designed for proving safety-properties but does not perform so well when proving reachability properties. This is explained by the fact that safety properties are expressed by “Box” modalities ($[\pi]$), while reachability properties are expressed by “Diamond” modalities ($\langle\pi\rangle$) and KeYmaera performs better when proving formulas solely with “Boxes” (we note that since $\neg[\pi]\neg$ is semantically equivalent to $\langle\pi\rangle$, we mean formulas with “Boxes” which are not in the scope of a negation).

dReach - A complement to KeYmaera.

The conception of dReach is related to dReal, whose theoretical foundations are based in the notion of δ -satisfiability. dReal checks the satisfiability of formulas like φ^δ , where φ is a first order formula and δ is a real. Thus, φ^δ is a syntactic variant of φ that encodes a notion of numerical perturbation on logic formulas. Essentially, the notion of satisfiability is relaxed

to admit δ -bounded errors. With this relaxation, δ -complete decision procedures can fully exploit the power of numerical approximations without losing formal correctness guarantees.

Thus, dReach [44] is a tool based on δ -reachability, *i.e.* reachability under some error or perturbation δ , which works over dReal [28]. This tool is designed to study hybrid systems. However, we can even present it as a tool to study reconfigurable system, described by tuples $(X, Q, flow, jump, inv, init)$ such that X is a set of real state variables, Q is a set of configurations and $flow, jump, inv, init$ are functions that assign SMT formulas that dReal can handle (first-order formulas over the reals including polynomials, trigonometric functions, exponential functions, Lipschitz-continuous ODEs, etc.) to each configuration in Q . In this case $flow$ determines the differential equations guiding the continuous evolution in each configuration, $jump$ provides conditions to jump between configurations, inv states an invariant for the configuration and $init$ determines the initial states.

Thus, this tool is able to deal with systems whose dynamics is described using many nonlinear differential equations if they are Lipschitz-continuous ODEs. Moreover, under a bounded error δ , it can solve reachability problems, which are not easy to study with $d\mathcal{L}$ and Keymaera.

However, dReach also presents some practical limitations when comparing to $d\mathcal{L}$ and KeYmaera. The fact that we are dealing with hybrid systems is an issue since we can have a jump condition which is too general. In this way, it is possible that we obtain a system which can jump almost anytime and to anywhere. This generates an infinite number of trajectories which are, in practice, impossible to follow. Anyway, usually in reachability problems we are only interested in finding a witness rather than all possible solutions, which is more important for safety.

Example 3.2.5. Consider again the ODE model for circadian rhythm which is presented in Example 2.1.1 and, therefore, to study it using $d\mathcal{L}$ is difficult. Moreover, we can obtain a hybrid model if we introduce a control variable such as in Example 3.2.4. Indeed, till now we considered that the total concentration of KaiC was constant but we can add the unphosphorylated form of KaiC using a control variable u . This variable determines the rate with which KaiC is introduced in the system. In order to keep the model simple, we note that this increment in x_u can be represent by the respective increase in C , the total amount of the protein KaiC. Therefore, in the model presented in [14], we note that $x_u = C - x_t - x_s - x_{ts}$ and we add the ODE $C' = u$ where u , the control variable, can take the values 0 (no control), 1 (small increment), or 4 (strong increment). We consider that the value of u can be changed at each time unit, thus, obtaining a hybrid model.

$$\begin{cases} x'_a = 10 \frac{5^4}{x_s^4 + 5^4} - 0.45x_a \\ x'_t = 20.51 \frac{(C - x_t - x_s - x_{ts})^4}{(C - x_t - x_s - x_{ts})^4 + 29.95^4} \frac{x_a^4}{x_a^4 + 10^4} - 0.24x_t \\ x'_{ts} = 10.74 \frac{x_t^4}{x_t^4 + 11.42^4} \frac{x_a^4}{x_a^4 + 10^4} - 0.28x_{ts} \\ x'_s = 6.61 \frac{x_{ts}^4}{x_{ts}^4 + 10.16^4} \frac{13^4}{x_a^4 + 13^4} - 0.081x_s \\ C' = u \end{cases}$$

As a reconfigurable model, we consider three configurations corresponding to the three control values. In each case, we consider a different value for the control variable u . Furthermore, we consider a terminal configuration which is reached whenever the goal is attained. In the context of dReach, tuple $(X, Q, flow, jump, inv, init)$ is such that $X = \{x_a, x_s, x_{ts}, x_t, C\}$

and Q consider the four configurations. Moreover *flow* assigns to each configuration the corresponding ODE with the corresponding value for u along with a time counter τ whose corresponding differential equation is $\tau' = 1$. For the final configuration, we consider a ODE with null equations, in such a way that there is no evolution along time. The jump condition must be chosen carefully in order to obtain a treatable model. We consider that the system can change its configuration at 5, 10 and 15 units of time. This will be explained further. In [14], the condition used to guarantee the positivity of x_u is $C \geq \frac{20.51}{0.24} + \frac{10.74}{0.28} + \frac{6.61}{0.081}$. However, in this case we allow smaller values for the initial value of C and establish the *inv* condition as $x_s + x_t + x_{ts} \leq C$ on each configuration to guarantee the positivity of x_u .

Simulating numerically the system without controls during 20 units of time, the value of x_t will never go over 26. We check if it is possible to obtain a trajectory where x_t reaches the value of 30 during the same interval of time for the initial conditions $C^0 = 80$, $x_a^0 = 10$, $x_s^0 = 10$, $x_t^0 = 10$ and $x_{ts}^0 = 10$ using our discrete control u . Thus, for *init* we consider the configuration with no control and $C^0 = 80$, $x_a^0 = 10$, $x_s^0 = 10$, $x_t^0 = 10$ and $x_{ts}^0 = 10$ as initial condition.

In order to use dReach, we must design a control strategy. Therefore, we let our system start at the state with no control and we allow the system to change its state at 5, 10, and 15 units of time. We also add the jump condition “ $x_t = 30$ ” in such a way that, when it is satisfied, the system is allowed to change to the terminal configuration.

Running dReach, we conclude that our control strategy assures that the system is controllable from the initial state $C^0 = 80$, $x_a^0 = 10$, $x_s^0 = 10$, $x_t^0 = 10$ and $x_{ts}^0 = 10$ to a state where $x_t = 30$ within 20 units of time. Moreover, a numerical solution (witness) for this problem is generated and the graphics of it can be seen in Figure 3.5. The solution given by dReach corresponds to letting the system evolve during 15 units of time with no control ($u = 0$) and then change to a strong control (with $u = 4$).

Note that, in fact we do not have reachability but δ -reachability. However, we can think about a very small value for δ in such way that it is not significant.

The purpose of this example is to illustrate how reachability problems for reconfigurable and, in particular, hybrid systems can be studied using dReach, as an alternative to KeYmaera. Indeed, combining these two tools we can have a solid methodology to study properties of hybrid systems.

3.3 Final remarks.

Hybrid and, more generally, reconfigurable behaviors are common in many areas including in biological contexts after the explicit introduction of discrete controllers. This kind of behavior also finds a natural correspondence with PWL models where the continuous dynamics inside each domain is mixed with discrete time reconfigurations at the boundaries. Our contribution is the application of $d\mathcal{L}$ and the tool KeYmaera to the study of biological models which embed hybrid feature – such as PWL models. $d\mathcal{L}$ is a language whose syntax is naturally able to specify properties of this kind of system. Moreover, the proof calculus of $d\mathcal{L}$ can be used to obtain safety conditions for a system operation mode or controlled procedure. This can be attained using the KeYmaera computational tool which algorithmically obtains a large part of a proof.

Moreover, we observe that KeYmaera presents some limitations when dealing with reachability problems. In this case, we propose dReach as an alternative since it was specifically

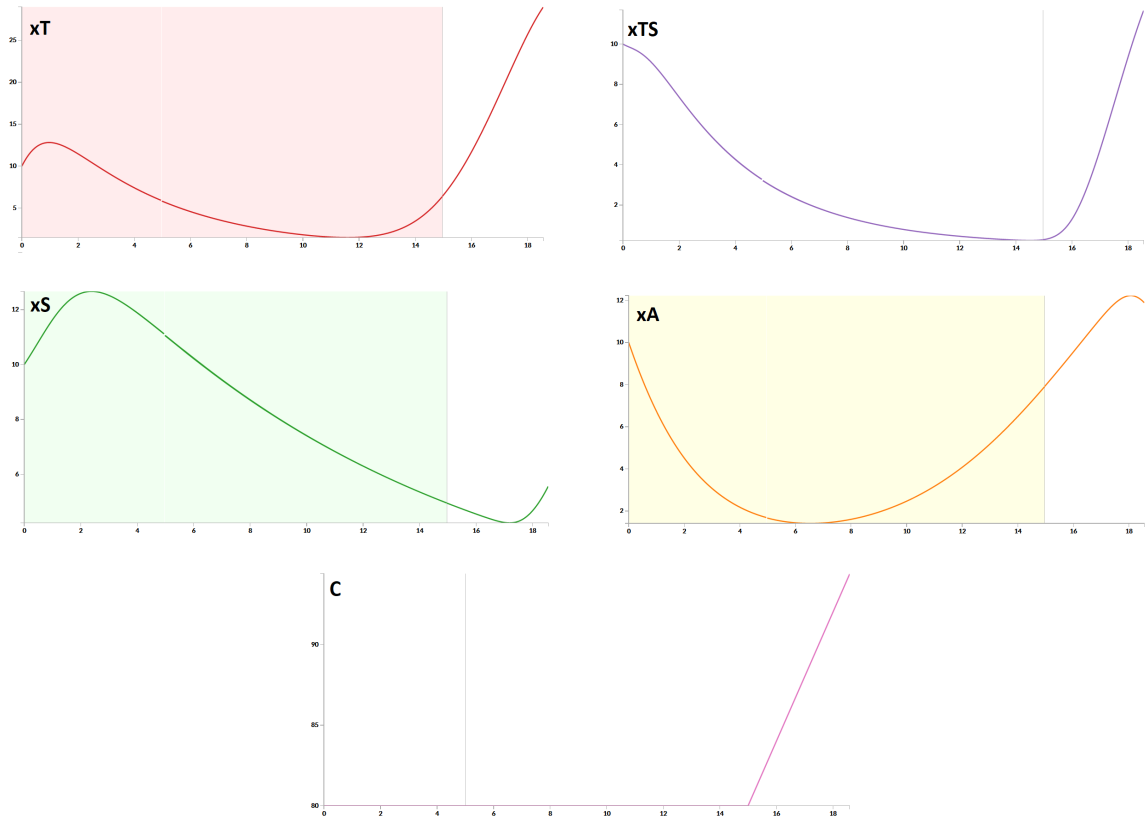


Figure 3.5: Solution for the reachability problem.

developed to work with δ -reachability. Therefore, our work proposes that one can use these two approaches according to the appropriated context in order to obtain a powerful tool to the study of hybrid systems.

Chapter 4

Reactive Boolean networks: An intermediary step between PWL and BN models

In this chapter we introduce a new kind of model for biological regulatory networks which we call reactive Boolean networks (RBN). The model is based on switch graphs [26], which are reactive structures, *i.e.*, discrete structures whose configuration can change whenever a discrete event occur. Thus, these models are still reconfigurable structures and can be seen as a discretization of hybrid models. Indeed, this will be highlighted when establishing a connection between PWL models and BN models, in the context of biological regulatory network models. In this chapter we introduce new structures/models as generalization of switch graphs; concepts such as bisimulation for these newly introduced structures/models; and prove some important results about them. Thus, we introduce RBN models and compare their utility to the already existing kinds of models. Finally, we generalize the concept of switch graphs to include weights and discuss their utility. During the rest of this chapter, we assume that the reader is familiarized with the concepts of modal logic and bisimulation [6]. The work presented here was published in [19, 20, 22]. Some of the work can be found in another submitted paper [59].

4.1 Switch graphs.

The term reactive models is applied to graph-like models whose set of edges may be altered whenever an edge is crossed. This notion was introduced by Gabbay but some examples of this kind of structure had already been presented earlier. For instance by van Benthem [5] and Areces [1, 2] introduced the notions of sabotage and swap logic, respectively. Particularly, in [2], the authors propose to enrich modal logics with relation-changing operators. In this way, these operators describe how the frame would change, *i.e.* if we conceive a frame as a graph, the introduced operators describe which edges should be removed or added at each step. Switch graphs were introduced by Marcelino and Gabbay in [26] as a model to reactive systems. Contrarily to what was proposed by Areces, a switch graph contains in its definition every possible future configurations, instead of having a language with specific operators to act over its structure. In this way, the cause for the reactivity is moved from the language to the model itself. Thus, our work is model-oriented and we introduce a usual modal language

and reactivity is occurs naturally due to the structure of a switch graphs.

Definition 4.1.1. A switch graph is defined as a pair (W, S) where W is a set of *states* or *worlds* and S is a set of generalized edges such that:

- $S_0 \subseteq W \times W$;
- $S_{n+1} \subseteq S_0 \times S_n \times \{\circ, \bullet\}$.
- $S = \bigcup_{i \geq 0} S_i$

An edge $s \in S_n$ is said to be a *n-level edge*. Moreover, if $n \geq 1$, then s is said to be an *higher-level edge*.

Switch graphs are generalizations of graphs in such a way that usual graphs only contain 0-level edges. Moreover, in switch graphs have multiple edges but only 0-level edges can be crossed. Higher-level edges are uniquely included in order to describe reactive dynamics. Higher-level edges can either *inhibit* (i.e. temporarily remove) an edge from the model or *activate* (i.e. restore) an edge into the model. In this way, an higher-level edge $(s_0, s_n, *)$ can be either an *activator* or an *inhibitor* according to the value of $*$: it is an activator if $* = \bullet$ and an inhibitor if $* = \circ$. Semantically, if an higher-level edge $(s_0, s_n, *)$ is present in a switch graph, then the edge s_n is activated (respectively, inhibited) whenever s_0 is crossed and $* = \bullet$ (respectively, $* = \circ$). In order to formally describe these change on a switch graph (W, S) , we introduce the notion of *instantiation* as a function $I : S \rightarrow \{0, 1\}$ in such a way that $I(s) = 1$ means that the edge $s \in S$ is present in the model and $I(s) = 0$ means that s is temporarily removed. Whenever an edge of a switch graph is crossed, the instantiation I must be updated in order to reflect all possible activations/inhibitions caused by the higher-level edges. In this thesis, in graphical representations, we illustrate worlds and 0-level edges as usual vertices and arrows of an oriented graph. An higher-level edge $(s_0, s_n, *)$ is illustrated as an arrow going from the edge s_0 to the edge s_n whose head is black whenever $* = \bullet$ and white whenever $* = \circ$. Finally, an instantiation I is illustrated by representing an edge s with a dashed line whenever $I(s) = 0$ (i.e. s is temporarily removed) and with a solid line otherwise.

The update of the instantiation function must be coherent with the changes induced by higher-level edges. Thus, given an instantiation I and a 0-level edge s_0 we denote by I^{s_0} the updated instantiation function after crossing s_0 and define it as:

$$I^{s_0}(s) = \begin{cases} 1, & \text{if } (s_0, s, \bullet) \in S \text{ and } I((s_0, s, \bullet)) = 1 \\ 0, & \text{if } (s_0, s, \circ) \in S \text{ and } I((s_0, s, \circ)) = 1 \\ I(s), & \text{otherwise} \end{cases}$$

We note that some inconsistencies could arise if two parallel higher-edges exist. For instance we could have an activator and an inhibitor edge acting over the edge at the same time. We will not consider these cases here but they could be solved by providing a rule like “inhibitor edges prevail over activator edges”. We also note that, given a particular switch graph, if there is an integer N such that there is at least one N -level edge and no $(N + 1)$ -level edges are considered, then N -level edges are either active or can be ignored, since there is not any $(N + 1)$ -level edge to activate them.

We follow with a simple example.

Example 4.1.1. Consider the switch graph illustrated in Figure 4.1, with the following abbreviations: $e_2 = ((w, w), (w, w), \circ)$ and $e_1 = ((w, w), ((w, w), (w, w), \circ), \bullet)$. This corresponds to a switch graph (W, S) such that $W = \{w\}$ and $S = \{(w, w), e_1, e_2\}$.

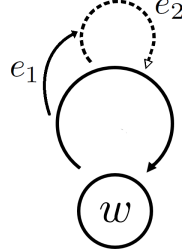


Figure 4.1: Example of a switch graph representing a counter.

In this figure, the world w and the 0-level edge (w, w) are depicted as usual. For each higher-level edge $(s, s', *)$, they are graphically represented as going from s to s' and with either a white head (whenever $* = \circ$) or a black head (whenever $* = \bullet$). Every edge s is represented as dashed arrow whenever $I(s) = 0$ and as solid one otherwise, implicitly describing the considered instantiation. Putting all together, higher-level edges with white head are inhibitors and edges with black head are activators. On other hand, dashed arrows should not be considered or crossed because they are temporarily removed.

Finally, we describe how a graph can be altered whenever an edge is crossed. The switch graph above, only admits an edge that can be crossed: (w, w) . Thus, we can successively cross it until it is no more possible, we obtaining the sequence of graphs depicted in Figure 4.2.

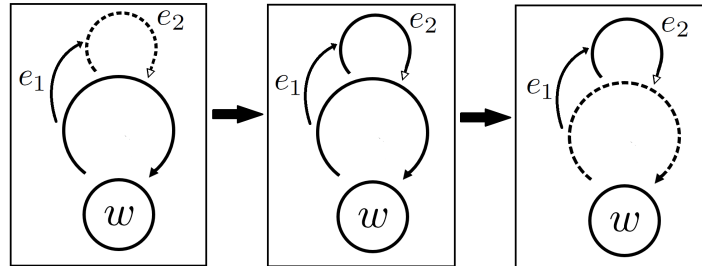


Figure 4.2: Evolution of a switch graph representing a counter.

Note that when crossing (w, w) for the first time, there is an inhibitor edge e_2 that would be triggered. However, this time, it is temporarily removed. Nevertheless, e_1 , which is an activator edge, will be triggered and restore e_2 . Because of this, when (w, w) is crossed for the second time, e_2 is already active and acts over (w, w) , temporarily removing it. Thus, we cannot make any move more since (w, w) , which is now removed from the graph, is the only crossable edge. Since this switch graph presents an edge that can be crossed exactly two times, it is called a *counter*, in the literature.

4.1.1 Reactive frames, logic and bisimulation.

In this section we propose modal logic for switch graph models and discuss its suitability. Moreover, we introduce a notion of bisimulation for switch graphs resorting to an auxiliary

structure called reactive frame, introduced by Gabbay & Marcelino in [25].

Definition 4.1.2. A pair (W, Δ) is said to be a reactive frame where W is the set of worlds (or states) and Δ is the set of *paths* if and only if:

- $W \subseteq \Delta$
- $\Delta \subseteq \bigcup_{0 < n < \infty} W^n$
- Δ is closed for *subpaths*, i.e. $(w_1, \dots, w_n) \in \Delta$ whenever $(w_1, \dots, w_n, w_{n+1}) \in \Delta$, for every $w_1, \dots, w_{n+1} \in W$.

In this context, we define a function $t : \Delta \rightarrow W$ such that $t(w_1, \dots, w_n) = w_n$, i.e. t returns the last world of a path. We introduce some simplifications to notation: given a path $\lambda = (w_1, \dots, w_n)$, we can represent it by $w_1 \dots w_n$ and λw represents the path $w_1 \dots w_n w$. A path γ *extends* or *is an extension* of a path λ if there exist $w_0, \dots, w_n \in W$ such that $\gamma = \lambda w_0 \dots w_n$. Moreover, every path is an extension of itself. Finally, the *length* of a path is the number of worlds it contains (non necessarily distinct). For instance, the length of (w_1, \dots, w_n) is n and (w_1, \dots, w_n, w_1) is $n + 1$.

Reactive frames represent reactive transition systems in an implicit way, by describing all possible generated paths. This allows us to explicitly describe all admissible evolutions of a system even if it is not easy to obtain a switch graph that generates it. In a dual way, switch graphs provide a structure with implicit reactive rules which generates the corresponding set of paths. As proven by Marcelino & Gabbay in [26], every reactive behavior can be described by each approach. In particular, that proof implies that, for each switch graph, we can obtain an equivalent reactive frame.

Reactive frames can be seen as Kripke frames where paths represent worlds and the accessibility relation is obtained for paths. In particular, Gabbay & Marcelino introduce the notion of reactive model and propose a bimodal logic to study these models. This is called the *logic of reactive models*.

Definition 4.1.3. Let Ω be a set of *atomic propositions*. Define the set of formulas for the logic of reactive models, $RFml(\Omega)$, as the least set such that:

- $\Omega \subseteq RFml(\Omega)$
- $\neg \varphi \in RFml(\Omega)$ whenever $\varphi \in RFml(\Omega)$
- $\varphi \wedge \psi \in RFml(\Omega)$ whenever $\varphi, \psi \in RFml(\Omega)$
- $\Diamond_R \varphi \in RFml(\Omega)$ whenever $\varphi \in RFml(\Omega)$ and $a \in A$
- $\Diamond_P \varphi \in RFml(\Omega)$ whenever $\varphi \in RFml(\Omega)$

Other operators like \perp , \top , \vee , \rightarrow , \Box_R and \Box_P are introduced as abbreviations, as usual.

Informally, the modality R represents the usual accessibility relation and P is an equivalence relation which relates paths ending at the same world.

Definition 4.1.4. Let Ω be a set of *atomic propositions*. Then a tuple (W, Δ, V) is a *reactive model* when (W, Δ) is a reactive frame and $V : \Omega \rightarrow 2^\Delta$ is a function assigning to each atomic proposition the set of paths satisfying it.

We note that, in this context, the evaluation of formulas is done with respect to a path rather than a state/world. The satisfiability of a formula φ at a path λ of a reactive model $\mathcal{M} = (W, \Delta, V)$ is denoted by $\mathcal{M}, \lambda \models \varphi$ and defined it recursively as:

- $\mathcal{M}, \lambda \models \varphi \Leftrightarrow \lambda \in V(\varphi)$, whenever $\varphi \in \Omega$.
- $\mathcal{M}, \lambda \models \neg\varphi \Leftrightarrow \mathcal{M}, \lambda \not\models \varphi$
- $\mathcal{M}, \lambda \models \varphi \wedge \psi \Leftrightarrow \mathcal{M}, \lambda \models \varphi$ and $\mathcal{M}, \lambda \models \psi$
- $\mathcal{M}, \lambda \models \Diamond_R \varphi \Leftrightarrow \mathcal{M}, \lambda w \models \varphi$ for some $w \in W$ such that $\lambda w \in \Delta$
- $\mathcal{M}, \lambda \models \Diamond_P \varphi \Leftrightarrow \mathcal{M}, \gamma \models \varphi$ for some $\gamma \in \Delta$ such that $t(\gamma) = t(\lambda)$

This definition can be extended to the remaining operators introduced by abbreviation.

We now introduce a definition of bisimulation for reactive models. Indeed, as would be expected, this coincides with the usual definition of bisimulation for bimodal logic.

Definition 4.1.5. Let (W, Δ, V) and (W', Δ', V') be two reactive models. A relation $\mathcal{S} \subseteq \Delta \times \Delta'$ is a bisimulation if and only if, for all $\lambda \in \Delta, \lambda' \in \Delta'$, such that $(\lambda, \lambda') \in \mathcal{S}$,

(*R-zig*) $\forall w \in W (\lambda w \in \Delta \Rightarrow \exists w' \in W', \lambda' w' \in \Delta' \text{ such that } (\lambda w, \lambda' w') \in \mathcal{S})$

(*R-zag*) $\forall w' \in W' (\lambda' w' \in \Delta' \Rightarrow \exists w \in W, \lambda w \in \Delta \text{ such that } (\lambda w, \lambda' w') \in \mathcal{S})$

(*P-zig*) $\forall \gamma \in \Delta (t(\lambda) = t(\gamma) \Rightarrow \exists \gamma' \in \Delta' (t(\lambda') = t(\gamma') \text{ and } (\gamma, \gamma') \in \mathcal{S}))$

(*P-zag*) $\forall \gamma' \in \Delta' (t(\lambda') = t(\gamma') \Rightarrow \exists \gamma \in \Delta (t(\lambda) = t(\gamma) \text{ and } (\gamma, \gamma') \in \mathcal{S}))$

(*atom*) For each $p \in \Omega$, $\lambda \in V(p) \Leftrightarrow \lambda' \in V(p)$

Example 4.1.2. Consider a set of atomic propositions $\Omega = \{p\}$ and two reactive models $\mathcal{M}_1 = (W_1, \Delta_1, V_1)$ and $\mathcal{M}_2 = (W_2, \Delta_2, V_2)$ where: $W_1 = \{w_1, w_2, w_3\}$, $\Delta_1 = \{w_1, w_2, w_3, w_1 w_2, w_1 w_2 w_1, w_1 w_3, w_1 w_3 w_1\}$ and $V_1(p) = \{w_2, w_3, w_1 w_2, w_1, w_3\}$; and $W_2 = \{v_1, v_2\}$, $\Delta_2 = \{v_1, v_2, v_1 v_2, v_1 v_2 v_1\}$ and $V_2(p) = \{v_2, v_1 v_2\}$. Then $\mathcal{S} = \{(w_1, v_1), (w_2, v_2), (w_3, v_2), (w_1 w_2, v_1 v_2), (w_1 w_2 w_1, v_1 v_2 v_1), (w_1 w_3, v_1 v_2), (w_1 w_3 w_1, v_1 v_2 v_1)\}$ is a bisimulation.

We point out the importance of the modality indexed by P for the coherence of the notion of bisimulation in reactive models. Otherwise, note that if $\Omega \equiv \emptyset$, then $\{(w_1, v_1), (w_1 w_2, v_1 v_1), (w_1 w_2 w_2, v_1 v_1 v_2)\}$ would be a bisimulation between $(\{w_1, w_2\}, \{w_1, w_1 w_2, w_1 w_2 w_2\}, V)$ and $(\{v_1, v_2\}, \{v_1, v_1 v_1, v_1 v_1 v_2\}, V)$. This makes no sense since it would relate two paths whose terminal state is distinct.

We now present a proof of the complete Hennessy-Milner theorem under some conditions. We start by proving modal equivalence.

Lemma 4.1.1. Let (W, Δ, V) and (W', Δ', V') be reactive models, let $\lambda \in \Delta, \lambda' \in \Delta'$ and let $\mathcal{S} \subseteq \Delta \times \Delta'$ be a bisimulation of reactive models. Then $(\lambda, \lambda') \in \mathcal{S}$ implies $\mathcal{M}, \lambda \models \varphi \Leftrightarrow \mathcal{M}', \lambda' \models \varphi$ for every formula $\varphi \in R\mathcal{Fml}(\Omega)$.

Proof. The proof is by induction over the structure of formulas. If $\varphi \in \Omega$, then $\mathcal{M}, \lambda \models \varphi \Leftrightarrow \mathcal{M}', \lambda' \models \varphi$ by definition of bisimulation. The non-atomic cases are presented below, under the hypothesis that $(\lambda, \lambda') \in \mathcal{S}$.

- $M, \lambda \models \neg\varphi$
 $\Leftrightarrow M, \lambda \not\models \varphi$ definition
 $\Leftrightarrow M', \lambda' \not\models \varphi$ inductive hypothesis
 $\Leftrightarrow M', \lambda' \models \neg\varphi$ definition
- $M, \lambda \models \varphi \wedge \psi$
 $\Leftrightarrow M, \lambda \models \varphi$ and $M, \lambda \models \psi$ definition
 $\Leftrightarrow M', \lambda' \models \varphi$ and $M', \lambda' \models \psi$ inductive hypothesis
 $\Leftrightarrow M', \lambda' \models \varphi \wedge \psi$ definition
- $M, \lambda \models \Diamond_R \varphi$
 $\Rightarrow \exists w \in W, \lambda w \in \Delta$ and $M, \lambda w \models \varphi$ definition
 $\Rightarrow \exists w' \in W', \lambda' w' \in \Delta'$ such that λw and $\lambda' w'$ are bisimilar bisimulation definition
 $\Rightarrow \exists w' \in W', \lambda' w' \in \Delta'$ and $M', \lambda' w' \models \varphi$ inductive hypothesis
 $\Rightarrow M', \lambda' \models \Diamond_R \varphi$ definition

The reciprocal condition is proved analogously.

- $M, \lambda \models \Diamond_P \varphi$
 $\Rightarrow \exists \gamma \in \Delta, t(\gamma) = t(\lambda)$ and $M, \gamma \models \varphi$ definition
 $\Rightarrow \exists \gamma' \in \Delta'$ with $t(\gamma') = t(\lambda')$ such that γ' and γ are bisimilar bisimulation definition
 $\Rightarrow \exists \gamma' \in \Delta'$ with $t(\gamma') = t(\lambda')$ and $M', \gamma' \models \varphi$ inductive hypothesis
 $\Rightarrow M', \lambda' \models \Diamond_P \varphi$ definition

The reciprocal condition is proved analogously

□

The reciprocal of the Lemma 4.1.1 is only valid for a restricted class of models. In classical modal logic, image-finiteness is usually imposed. We resort to a slightly more relaxed notion, that is the one of saturated model, based in a similar notion presented in [2]. With this restriction, Theorem 4.1.2 below explains how a bisimulation relating paths indistinguishable by formulas of $RFml(\Omega)$ could be built. In the sequel, for a relation $Z \subseteq \Delta \times \Delta$ on paths, notation $Z[\lambda]$ abbreviates the set $\{\gamma \in \Delta : \lambda Z \gamma\}$.

Definition 4.1.6. Let Σ be a set of formulas and $M = (W, \Delta, V)$ a reactive model.

- Σ is *satisfiable* over a set of paths $\Lambda \subseteq \Delta$ if there is a path $\lambda \in \Lambda$ such that $M, \lambda \models \varphi$ for every $\varphi \in \Sigma$.
- Σ is *finitely satisfiable* over a set of paths $\Lambda \subseteq \Delta$ if, for every finite subset $\bar{\Sigma} \subseteq \Sigma$, there is a path $\lambda \in \Lambda$ such that $\lambda \models \varphi$ for every $\varphi \in \bar{\Sigma}$.
- A model is *Z-saturated* over a relation $Z \subseteq \Delta \times \Delta$, if, for all λ , every set Σ is satisfiable over $Z[\lambda]$ whenever Σ is finitely satisfiable over $Z[\lambda]$.

Before presenting the next theorem, we introduce some more concepts. Given a reactive frame (W, Δ) , the relation $R \subseteq \Delta \times \Delta$ is defined by the condition: $(\lambda, \gamma) \in R$ iff $\exists w \in W, \gamma = \lambda w$. Similarly, we define the relation $P \subseteq \Delta \times \Delta$ by the condition: $(\lambda, \gamma) \in P$ iff $t(\gamma) = t(\lambda)$.

We state and prove the next theorem in order to complete the Hennessy-Milner theorem for the presented logic. It is not proved for all reactive models but comprise an embracing class of them.

Theorem 4.1.2. *Let M and M' be two P -saturated and R -saturated reactive models. A non-empty relation $\mathcal{S} \subseteq \Delta \times \Delta'$ such that $(\lambda, \lambda') \in \mathcal{S}$ iff for any formula φ , $M, \lambda \models \varphi \Leftrightarrow M', \lambda' \models \varphi$, is a bisimulation.*

Proof. Consider $(\lambda, \lambda') \in \mathcal{S}$. Suppose that $(\lambda, \gamma) \in R$, for some $\gamma \in \Delta$ and let $Sat(\gamma) = \{\varphi : M, \gamma \models \varphi\}$. Then, for each finite subset $\Sigma' \subseteq Sat(\gamma)$, $M, \lambda \models \Diamond_R \bigwedge_{\varphi \in \Sigma'} \varphi$ holds and, therefore,

$M', \lambda' \models \Diamond_R \bigwedge_{\varphi \in \Sigma'} \varphi$. This means that $Sat(\gamma)$ is finitely satisfiable over $R[\lambda']$, and since M' is

R -saturated, $Sat(\gamma)$ is satisfied over $R[\lambda']$. Thus, there exists a state γ' such that $(\lambda', \gamma') \in R$ and $(\gamma, \gamma') \in \mathcal{S}$. The reciprocal is proven analogously: if $(\lambda, \lambda') \in \mathcal{S}$ and $(\lambda', \gamma') \in R$, then there exists some $w \in W$ such that $(\lambda w, \gamma') \in \mathcal{S}$.

Suppose now that $(\lambda, \gamma) \in P$, for some $\gamma \in \Delta$ and consider $Sat(\gamma) = \{\varphi : M, \gamma \models \varphi\}$. Then, for each finite subset $\Sigma' \subseteq Sat(\gamma)$, $M, \lambda \models \Diamond_P \bigwedge_{\varphi \in \Sigma'} \varphi$ and, therefore, $M', \lambda' \models \Diamond_P \bigwedge_{\varphi \in \Sigma'} \varphi$.

This means that $Sat(\gamma)$ is finitely satisfiable over $P_{\lambda'}$, and since M' is P -saturated, $Sat(\gamma)$ is satisfied over $P_{\lambda'}$. Again, there exists a state γ' such that $(\lambda', \gamma') \in P$ and $(\gamma, \gamma') \in \mathcal{S}$. The reciprocal is proven analogously: if $(\lambda, \lambda') \in \mathcal{S}$ and $(\lambda', \gamma') \in P$, then there exists some $\gamma \in \Delta$ such that $(\lambda, \gamma) \in P$ and $(\gamma, \gamma') \in \mathcal{S}$. \square

The theorem would fail for non R -saturated models. The following proposition gives a sufficient condition for a model to be R -saturated.

Proposition 4.1.3. *Let $M = (W, \Delta, V)$ be a reactive model and $R[\lambda]$ as defined above. If $|R[\lambda]| < \infty$, for every $\lambda \in \Delta$, then M is R -saturated.*

Proof. Suppose that for every $\lambda \in \Delta$, $|R[\lambda]| < \infty$ holds for M but the model is not R -saturated. This means that there exists $\lambda \in \Delta$ and a set Σ of formulas such that Σ is finitely satisfiable over $R[\lambda]$ but not satisfiable over $R[\lambda]$.

Clearly, any formula $\varphi \in \Sigma$, $\{\varphi\}$ is satisfiable over $R[\lambda]$ which means that $\Diamond_R \Sigma = \{\Diamond_R \varphi : \varphi \in \Sigma\}$ is satisfied in λ . Since $|R[\lambda]| < \infty$, every path in $R[\lambda]$ can be enumerated as $\gamma_1, \dots, \gamma_n$. Since Σ is not satisfiable over $R_a(\lambda)$, there is a formula $\varphi_i \in \Sigma$ for each γ_i , $i \in \{1, \dots, n\}$, such that φ_i is not satisfied in γ_i . However, for any $i \in \{1, \dots, n\}$, $\Diamond_R \varphi_i$ is satisfied in λ . Thus, the set $\Phi = \{\varphi_1, \dots, \varphi_n\} \subseteq \Sigma$ is finite and, therefore, satisfiable over $R[\lambda]$. This leads to a contradiction since each path $\gamma_i \in R[\lambda]$ does not verify $\varphi_i \in \Phi$. \square

An analogous result for relation P is obtained along similar lines, however, if it holds, then it implies the absence of cycles in the reactive model.

Example 4.1.3. We present an example of a model which is not R -saturated and, therefore, does not verify the Theorem 4.1.2. This example is based in a similar one presented in [7] for non finite Kripke models.

We consider two Kripke frames as depicted in Figure 4.3. Note that these Kripke frames are particular cases of reactive structures and, therefore, we can also obtain a reactive frame from them. Both of them have a “root” (v_0 and w_0) and an infinite number of paths starting at that root (such as $v_0 v_1^1$, $v_0 v_1^2 v_2^2$ and $w_0 w_1^3 w_2^3 w_3^3$) can be obtained. All these paths start at the respective root and have a finite and increasing length. However, the reactive frame obtained from the Kripke frame on the right-side admits an additional path with an infinite number of successors ($w_0 w_1^0 w_2^0 w_3^0 \dots$). Considering the corresponding reactive models \mathcal{V} (on the left) and

\mathcal{W} (on the right) with a single proposition p which is valid along all paths of both models, we can verify that the paths v_0 and w_0 satisfy the same formulas of $RFml(\Omega)$. However, for these models, the relation \mathcal{S} that relates worlds which admit the same valid formulas is not a bisimulation. In order to demonstrate this, we note that $v_0 \mathcal{S} w_0$. Moreover, $\mathcal{W}, (w_0 w_1^1) \models \Diamond_R^n p$ for all n , where \Diamond_R^n denotes the sequence of n symbols \Diamond_R . Now, if we consider a world v_1^k , for arbitrary $k \in \mathbb{N}$, we note that $\mathcal{V}, v_0 v_1^k \not\models \Diamond_R^k p$ since the respective length of that path is exactly k (including the world v_0^k). This means that none of the successors of v_0 verifies the same formulas as $w_0 w_1^0$. Therefore, does not exist a path λ such that $\lambda = v_0 v$ and $\lambda \mathcal{S} w_0 w_1^0$.

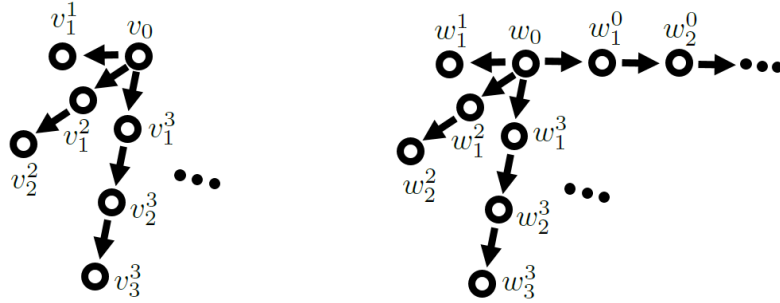


Figure 4.3: The reactive frame of two non bisimilar reactive models.

Indeed, the reactive model on the left side is not R -saturated. Note that the set of formulas $\Sigma = \{\Diamond_R^n p : n \geq 0\} \subseteq RFml(\Omega)$ is finitely satisfiable over \mathcal{V} because, for each finite set $\bar{\Sigma}$, we can find $N \in \mathbb{Z}^+$ such that, for each formula $\Diamond_R^n p \in \bar{\Sigma}$, we have $N > n$. Thus, $\mathcal{V}, v_0 \models \Diamond_R^n p \Leftrightarrow \mathcal{V}, v_0 v_1^N \dots v_n^N \models p$, which is true for each formula $\Diamond_R^n p \in \bar{\Sigma}$. However, the full set Σ is not satisfiable over \mathcal{V} because each path has finite length.

We were able to establish a notion of bisimulation for reactive systems. However, one can find some “lack of information” about the definition of bisimulation for reactive models. Although this definition is based on the usual one for bimodal logic, it is not intuitive when one thinks about the global structure of a reactive system. The reason is that we are defining bisimulation as a relation over paths instead of worlds. One example of this is the bisimulation presented in Example 4.1.2. Note that makes sense to relate the state w_1 with v_1 and states w_2, w_3 with v_2 . Although this connection can be retrieved, it is not intuitively obtained and an overview about the structure of the reactive system itself is not provided. Therefore, we conceive a notion of bisimulation for switch graphs by connecting them with reactive frames and proving its coherence with the one introduced for usual graphs.

Definition 4.1.7. Given a set of atomic propositions Ω . We say that $\mathcal{M} = (W, S, I, V)$ is a switch graph model whenever (W, S) is a switch graph, I is an (initial) instantiation and $V : \Omega \rightarrow W$ is a satisfiability function.

With this, we generalize the notion of switch graph to include propositions. The notion of instantiation in switch graph models is the same as before, since it was introduced for switch graphs and a switch graph model is itself a switch graph with an additional function. We now introduce the notion of bisimulation for switch graph model.

Definition 4.1.8. Consider a switch graph model (W, S, I, V) . We say that a reactive model (W', Δ, V') is *induced* by (W, S, V) when the following holds:

- $W = W'$
- Δ is the set of all possible paths generated by the generalized set of edges S , *i.e.*:
 - $(w) \in \Delta$ for any $w \in W$
 - $(w_0, w_1, \dots, w_n) \in \Delta$ if and only if, for every $n \geq 2, \forall i \in \{1, \dots, n-1\}$, $(w_i, w_{i+1}) \in S$ and $((I^{(w_0, w_1)}) \dots)^{(w_{i-1}, w_i)}(w_i, w_{i+1}) = 1$.
- V' is defined in order to be coherent with the notion of valuation for switch graph models: $\forall p \in \Omega, \lambda \in V(p)$ iff $t(\lambda) \in V'(p)$

Since W and W' coincide in this definition, we do not distinguish them in the rest of this thesis.

Definition 4.1.9. Given two switch graph models (W, S, I, V) , (W', S', I', V') whose induced reactive models are (W, Δ, \bar{V}) and (W', Δ', \bar{V}') , and a relation $\mathcal{R} \subseteq W \times W'$, we say that a relation $\mathcal{B} \subseteq \Delta \times \Delta'$ is *induced* by \mathcal{R} when \mathcal{B} can be obtained recursively by the following rules:

- $(w, w') \in \mathcal{R} \Leftrightarrow ((w), (w')) \in \mathcal{B}$ for every $w \in W, w' \in W'$.
- Let $\lambda \in \Delta$ and $\lambda' \in \Delta'$ be such that $(\lambda, \lambda') \in \mathcal{B}$. For every $w \in W$ and $w' \in W'$ such that $\lambda w \in \Delta$ and $\lambda' w' \in \Delta'$, we have $(\lambda w, \lambda' w') \in \mathcal{B}$ iff $(w, w') \in \mathcal{R}$.

Moreover, we say that \mathcal{R} is a *bisimulation for switch graph models* if the induced relation \mathcal{B} is a bisimulation for reactive models. Two switch graph models are said to be *bisimilar* if there is a bisimulation relating them.

Example 4.1.4. In the first example, we consider two switch graph models, as illustrated in Figure 4.4. For these two model we can find a relation $\mathcal{B} = \{(w_1, v_1), (w_2, v_2), (w_3, v_3), (w_4, v_3)\}$ which is a bisimulation. This can be observed by considering the corresponding reactive models. Since one can note that the relation induced by \mathcal{B} is $\{(w_1, v_1), (w_2, v_2), (w_3, v_3), (w_4, v_3), (w_1 w_3, v_1 v_3), (w_2 w_4, v_2 v_3), (w_3 w_4, v_3 v_3), (w_4 w_3, v_3 v_3)\}$. It is easy to check that this relation is, indeed, a bisimulation for reactive models.

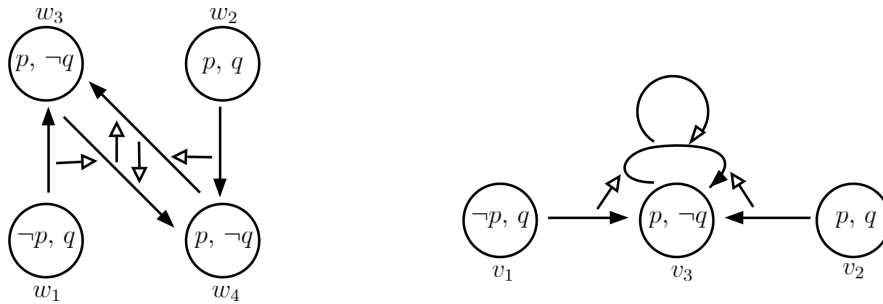


Figure 4.4: Two bisimilar switch graph models.

Indeed, the notion of bisimulation provides an intrinsic notion on how to reduce large models. This can be seen in this example, since we have two bisimilar models with a distinct number of states and edges.

Although the definition of bisimulation for switch graph models resorts to the notion of bisimulation for reactive models, it is coherent with the expected one. Indeed, next results show that this definition of bisimulation for switch graph models generalizes the usual one for Kripke models. Note that, in the following propositions, a Kripke model (W, R, V) can be seen as a switch graph model where the set of edges is R and the instantiation I is the constant function $I \equiv 1$, since no edge can be temporarily removed.

Proposition 4.1.4. *Let (W_K, R, V_K) , (W'_K, R', V'_K) be Kripke models and $\mathcal{B} \subseteq W \times W'$ a bisimulation. Let (W, Δ, V) (respectively, (W', Δ', V')) be the induced reactive model with respect to (W, R, V) (respectively, (W', R', V')). Let the relation $\mathcal{S} \subseteq \Delta \times \Delta'$ be the relation induced by \mathcal{B} . Then \mathcal{S} is a bisimulation of reactive models.*

Proof.

($R - \text{zig}$) Let us consider $\lambda \in \Delta$, $\lambda' \in \Delta'$ and $w \in W$ such that $(\lambda, \lambda') \in \mathcal{S}$ and $\lambda w \in \Delta$. By definition of \mathcal{S} , we conclude that $(t(\lambda), t(\lambda')) \in \mathcal{B}$ and $(t(\lambda), w) \in R$. Therefore, since \mathcal{B} is a bisimulation, there exists $w' \in W'$ such that $(t(\lambda'), w') \in R'$ and $(w, w') \in \mathcal{B}$. Thus, $\exists w' \in W', (t(\lambda w), t(\lambda' w')) \in \mathcal{B}$ which implies $\exists w' \in W', (\lambda w, \lambda' w') \in \mathcal{S}$.

($R - \text{zag}$) Analogous to $R - \text{zig}$.

($P - \text{zig}$) Let us consider $\lambda, \gamma \in \Delta$ and $\lambda' \in \Delta'$ such that $t(\lambda) = t(\gamma)$ and $(\lambda, \lambda') \in \mathcal{S}$. Therefore $(t(\lambda), t(\lambda')) \in \mathcal{B}$ implies $(t(\gamma), t(\lambda')) \in \mathcal{B}$ and, thus, $(\gamma, \lambda') \in \mathcal{S}$. The result follows because, trivially, $t(\lambda') = t(\lambda')$.

($P - \text{zag}$) Analogous to $P - \text{zag}$.

(atom) $M, \lambda \models_X p \Leftrightarrow M, t(\lambda) \models p$ for any $p \in \Pi$. Thus, if $(\lambda, \lambda') \in \mathcal{S}$, then $M, \lambda \models_X p \Leftrightarrow M, t(\lambda) \models p \Leftrightarrow M', t(\lambda') \models p \Leftrightarrow M', \lambda' \models_X p$

□

In the opposite direction a similar result comes out:

Proposition 4.1.5. *Let (W_K, R, V_K) , (W'_K, R', V'_K) be Kripke models and $\mathcal{S} \subseteq \Delta \times \Delta'$ a bisimulation between paths of the corresponding induced reactive models. Define relation $\mathcal{B} \subseteq W \times W'$ by the condition: $(w, w') \in \mathcal{B}$ iff there exists $(\lambda, \lambda') \in \mathcal{S}$ such that $t(\lambda) = w$ and $t(\lambda') = w'$. Then \mathcal{B} is a bisimulation between the original Kripke models.*

Proof. We prove the *zig* and *zag* conditions, as well as the equivalence between atomic propositions.

(*zig*) Let us suppose $(w, w') \in \mathcal{B}$ and that there exists $v \in W$ such that $(w, v) \in R$. Then, there exists $(\lambda, \lambda') \in \mathcal{S}$ such that $t(\lambda) = w$ and $t(\lambda') = w'$. Furthermore, $\lambda w \in \Delta$. Since \mathcal{S} is a bisimulation, there exists $v' \in W'$ such that $\lambda' w' \in \Delta'$ and $(\lambda w, \lambda' v') \in \mathcal{S}$. Hence, $t(\lambda v) = v$, $t(\lambda' v') = v'$ and, therefore, $v \mathcal{B} v'$.

(*zag*) Analogous to *zig*.

(*atom*) Finally, we note that $M, \lambda \models_X p \Leftrightarrow M, t(\lambda) \models p$ for any $p \in \Pi$ because $X = \Pi$. If $(w, w') \in \mathcal{B}$, then there exists $(\lambda, \lambda') \in \mathcal{S}$ such that $t(\lambda) = w$ and $t(\lambda') = w'$. Because $M, \lambda \models_X p \Leftrightarrow M', \lambda' \models_X p$, we conclude that $M, t(\lambda) \models p \Leftrightarrow M', t(\lambda') \models p$, i.e. $M, v \models p \Leftrightarrow M', v' \models p$.

□

We now introduce a language to describe properties of switch graph models. Although the proposed structure is more complex than usual Kripke models, it is pure modal logic. The reason for this is based on two main factors. Firstly, as mentioned before, we are focused in a model-based approach. Secondly, because we can, in some sense, obtain an equivalent state transition model with no higher-level edges. To do this, we follow an approach based in [2]. Let (W, S, I, V) be a switch graph model. Then we obtain an equivalent state transition model (W, R, \mathcal{V}) with no higher-level edges such that:

- $\mathcal{W} = W \times \mathcal{I}$, where \mathcal{I} is the set of all attainable instantiations of the switch graph model.
- $((w_1, I_2), (w_2, I_2)) \in R$ if $I_1(w, w') = 1$ and $I_2 = I_1^{(w, w')}$.
- $\mathcal{V} : \Omega \rightarrow 2^{\mathcal{W}}$ is such that $(w_1, I_1) \in \mathcal{V}(p) \Leftrightarrow w_1 \in V(p)$.

Nevertheless, in this representation we must consider a set of initial states or, equivalently, an initial instantiation to determine which was the initial instantiation of the corresponding switch graph model. We say that this is the *plain representation* of a switch graph model.

Note that both a plain representation of a switch graph model and its induced reactive model are unfolded models. However, a plain representation is not necessarily a tree-like model. Moreover, given a switch graph model (W, S, I, V) with $|W|, |S| < \infty$, we know that the respective plain representation $(\mathcal{W}, R, \mathcal{V})$ is necessarily finite because there are at most $2^{|S|}$ different instantiations. Thus $|\mathcal{W}| < |W| \cdot 2^{|S|} < \infty$ and $|R| \leq |\mathcal{W}|^2 < \infty$. This often is not the case with reactive models, since a loop automatically generates an infinite path.

We introduce the syntax of modal logic:

Definition 4.1.10. Given a set Ω of atomic proposition, we define the set $Fml(\Omega)$ of formulas for modal logic as the least set such that $\Omega \subseteq Fml(\Omega)$ and $\neg\varphi, \varphi \vee \psi, \Box\varphi \in Fml(\Omega)$ whenever $\varphi, \psi \in Fml(\Omega)$.

We introduce the notion of satisfiability for a formula at a state of a model. Given a switch graph model $\mathcal{M} = (W, S, I, V)$ and a formula $\varphi \in Fml(\Omega)$, we write $\mathcal{M}, w \models \varphi$ to state that a formula is true at a state $w \in W$. Also, for every $(w, w') \in S$ such that $I(w, w') = 1$, $\mathcal{M}^{(w, w')} = (W, S, I^{(w, w')}, V)$, i.e. $\mathcal{M}^{(w, w')}$ denotes the updated model after crossing the edge (w, w') .

Definition 4.1.11. The satisfiability relation is defined recursively as follows:

- $\mathcal{M}, w \models \varphi \Leftrightarrow w \in V(\varphi)$, whenever $\varphi \in \Omega$.
- $\mathcal{M}, w \models \neg\varphi \Leftrightarrow \mathcal{M}, w \not\models \varphi$
- $\mathcal{M}, w \models \varphi \vee \psi \Leftrightarrow \mathcal{M}, w \models \varphi$ or $\mathcal{M}, w \models \psi$
- $\mathcal{M}, w \models \Box\varphi \Leftrightarrow \mathcal{M}^{(w, w')}, w' \models \varphi$ for every $w' \in W$ such that $(w, w') \in S$ and $I(w, w') = 1$

Note that this is pure modal logic but the effects of reactivity are reflected when we evaluate formulas with the operator “ \Box ”. Indeed, contrarily to what happens in [2], the changes in the structure of the model are not caused by the operators but are already encoded on the model itself, due to the higher-level edges. We now prove the satisfiability theorem for these models.

Definition 4.1.12. Let $tr : Fml(\Omega) \rightarrow RFml(\Omega)$ be a translator function defined recursively as:

- $tr(p) = p$, for any $p \in \Omega$
- $tr(\neg\varphi) = \neg tr(\varphi)$ for $\varphi \in Fml(\Omega)$
- $tr(\varphi \vee \psi) = tr(\varphi) \vee tr(\psi)$ for $\varphi, \psi \in Fml(\Omega)$
- $tr(\Box\varphi) = \Box_R tr(\varphi)$, for $\varphi \in Fml(\Omega)$

To simplify the notation, we use the same symbol (\models) to represent satisfaction for both reactive models and switch graphs models. However, it should be clear by the context which is the case.

Lemma 4.1.6. Let $\mathcal{M} = (W, S, I, V)$ be a switch graph model and $\bar{\mathcal{M}} = (\bar{W}, \Delta, \bar{V})$ be the corresponding induced reactive model. Then $\mathcal{M}, w \models \varphi \Leftrightarrow \bar{\mathcal{M}}, (w) \models tr(\varphi)$, for any formula $\varphi \in Fml(\Omega)$

Proof. We prove this lemma by induction over formulas:

- If $p \in \Omega$, then $\mathcal{M}, w \models p$
 $\Leftrightarrow \bar{\mathcal{M}}, (w) \models p$ (def. of induced reactive model)
- $\mathcal{M}, w \models \neg\varphi$
 $\Leftrightarrow \mathcal{M}, w \not\models \varphi$ (definition)
 $\Leftrightarrow \bar{\mathcal{M}}, (w) \not\models \varphi$ (inductive hypothesis)
 $\Leftrightarrow \bar{\mathcal{M}}, (w) \models \neg\varphi$ (definition)
- $\mathcal{M}, w \models \varphi \vee \psi$
 $\Leftrightarrow \mathcal{M}, w \models \varphi$ or $\mathcal{M}, w \models \psi$ (definition)
 $\Leftrightarrow \bar{\mathcal{M}}, (w) \models \varphi$ or $\bar{\mathcal{M}}, (w) \models \psi$ (inductive hypothesis)
 $\Leftrightarrow \bar{\mathcal{M}}, (w) \models \varphi \vee \psi$ (definition)
- $\mathcal{M}, w \models \Box\varphi$
 $\Leftrightarrow \mathcal{M}^{(w, w')}, w' \models \varphi$ for every $w' \in W$ such that
 $(w, w') \in S$ and $I(w, w') = 1$ (definition)
 $\Leftrightarrow \bar{\mathcal{M}}, (w, w') \models \varphi$ for every $w' \in W$ such that
 $(w, w') \in \Delta$ (inductive hypothesis and *)
 $\Leftrightarrow \bar{\mathcal{M}}, (w) \models \Box_R \varphi$ (definition)

* observing that $\mathcal{M}^{(w, w')}$ is itself a switch graph model, one can use the inductive hypothesis. This step is valid because, if $\tilde{\Delta}$ is the set of paths for the reactive model induced by $\mathcal{M}^{(w, w')}$, then $(w, w', v_0, \dots, v_n) \in \Delta \Leftrightarrow (w', v_0, \dots, v_n) \in \tilde{\Delta}$ \square

Theorem 4.1.7. Let $\mathcal{M} = (W, S, I, V)$ and $\mathcal{M}' = (W', S', I', V')$ be two switch graph models, $w \in W$, $w' \in W'$, $\varphi \in Fml(\Omega)$ and $R \subseteq W \times W'$ be a bisimulation. Then, $\mathcal{M}, w \models \varphi \Leftrightarrow \mathcal{M}', w' \models \varphi$ whenever $(w, w') \in R$.

Proof. Consider $\bar{\mathcal{M}} = (W, \Delta, \bar{V})$ and $\bar{\mathcal{M}}' = (W', \Delta', \bar{V}')$, the reactive models induced by \mathcal{M} and \mathcal{M}' , respectively.

Consider $(w, w') \in R$. We prove this theorem by induction over formulas:

- If $p \in \Omega$, then $\mathcal{M}, w \models p$
 $\Leftrightarrow \bar{\mathcal{M}}, (w) \models p$ (Lemma 4.1.6)
 $\Leftrightarrow \bar{\mathcal{M}}', (w') \models p$ (*¹)
 $\Leftrightarrow \mathcal{M}', w' \models p$ (Lemma 4.1.6)
- $\mathcal{M}, w \models \neg\varphi$
 $\Leftrightarrow \mathcal{M}, w \not\models \varphi$ (definition)
 $\Leftrightarrow \mathcal{M}', w' \not\models \varphi$ (inductive hypothesis)
 $\Leftrightarrow \mathcal{M}', w' \models \neg\varphi$ (definition)
- $\mathcal{M}, w \models \varphi \vee \psi$
 $\Leftrightarrow \mathcal{M}, w \models \varphi$ or $\mathcal{M}, w \models \psi$ (definition)
 $\Leftrightarrow \mathcal{M}', w' \models \varphi$ or $\mathcal{M}', w' \models \psi$ (inductive hypothesis)
 $\Leftrightarrow \mathcal{M}', w' \models \varphi \vee \psi$ (definition)
- $\mathcal{M}, w \models \Box\varphi$
 $\Leftrightarrow \bar{\mathcal{M}}, (w) \models \Box_R\varphi$ (Lemma 4.1.6)
 $\Leftrightarrow \bar{\mathcal{M}}, (w, v) \models \varphi$, for every v such that $(w, v) \in \Delta$ (definition of induced model)
 $\Leftrightarrow \bar{\mathcal{M}}', (w', v') \models \varphi$, for every v' such that *¹ and Lemma 4.1.1
 $(w', v') \in \Delta'$
 $\bar{\mathcal{M}}', (w') \models \Box_R\varphi$ (definition)
 $\mathcal{M}', w' \models \Box\varphi$ (Lemma 4.1.6)

*¹ Definition of bisimulation for reactive models □

4.1.2 Weighted switch graphs.

We now proceed further our generalization effort by introducing weighted switch graphs. Not much theoretical study is done in this topic since our interest is to provide an alternative representation for reactive system which comprise weights. In the literature weights can be assigned to edges of regular graphs in order to describe quantitative features like costs, probabilities, levels of existence, among others. Since our aim is to apply these techniques to biochemical systems such as biological regulatory networks, it makes sense to consider this approach.

Definition 4.1.13. A *weighted switch graph* is a pair (W, S) together with an instantiation $I : S \rightarrow \mathcal{W} \cup \{\odot\}$ where \mathcal{W} is the *set of weights*, and can be chosen according to the context.

In weighted switch graphs, instead of simply considering that an edge is active or inhibited, each edge has a weight. In order to express this, we generalize the notion of instantiation by considering an instantiation to be a function whose codomain is a set of weights \mathcal{W} which contains an additional element \odot . Thus, if s is an edge of the model and I an instantiation, $I(s) = \odot$ means that the edge s is inhibited (temporarily removed from the model). Otherwise, we say that the edge s is active and has weight $I(s)$.

Given this definition, we can describe the evolution of a weighted switch graph when some edge $s \in W \times W$ is crossed in the following way:

$$I^s(t) = \begin{cases} I((s, t, \bullet)), & \text{if } (s, t, \bullet) \in S \text{ and } I((s, t, \bullet)) \neq \odot \\ \odot, & \text{if } (s, t, \circ) \in S \text{ and } I((s, t, \circ)) \neq \odot \\ I(t), & \text{otherwise.} \end{cases}$$

This means that an activator edge assigns its value to the edge it activates. Also, note that, in general, \odot is semantically different from 0 (for example in the context of costs). However, in some cases like probabilistic or fuzzy they coincide, *i.e.* $\odot \equiv 0$.

Weighted switch graphs also admit a plain representation, obtained through a process similar to the usual one. Moreover they are finite when both the set of states and the set of generalized edges are finite. Note that this does not depend on $|\mathcal{W}|$. Indeed, even with \mathcal{W} containing a non finite number of elements (for instance, if $\mathcal{W} = \mathbb{Z}_0^+$) this still holds. Note that each activator edge assigns its own weight to the activated edge. Thus, if I is the initial instantiation and S is the set of edges of the model, and $\bar{\mathcal{W}} = \{I(s) : s \in S\}$, then $|\bar{\mathcal{W}}| < \infty$ and there are, at maximum, $|\bar{\mathcal{W}}|^{|S|} < \infty$ attainable instantiations.

Fuzzy switch graphs.

Fuzzy switch graphs are weighted switch graphs whose set of weights \mathcal{W} is the interval $[0, 1]$. In this context, a weight is understood as the level of certainty about the existence of the respective edge. Thus, semantically we have $\odot \equiv 0$.

In a fuzzy context, one must consider adapted semantical interpretations of conjunction, disjunction, implication and negation. The first two are generalized by T-norms and T-conorms, respectively [42], whereas fuzzy implication and negation have been also widely studied [3].

Definition 4.1.14. A function $U : [0, 1] \times [0, 1] \rightarrow [0, 1]$, is called *isotonic* if $U(x, y) \leq U(x', y')$ whenever $x \leq x'$ and $y \leq y'$. Moreover, it is called *uninorm* if it is isotonic, commutative, associative and admits a neutral element $e \in [0, 1]$. If $e = 1$, then U is called a *T-norm*, and if $e = 0$, then U is called a *T-conorm* or a *S-norm*.

For instance, $\min(x, y)$ is a T-norm and $\max(x, y)$ an S-norm. For details see [42].

Definition 4.1.15. A unary operation $N : [0, 1] \rightarrow [0, 1]$ is called *antitonic* if $N(x) \geq N(x')$ whenever $x \leq x'$. Moreover it is called *fuzzy negation*, if it is antitonic, $N(0) = 1$ and $N(1) = 0$. N is *strong*, whenever $N(N(x)) = x$.

An example of a fuzzy negation is the Gödel Negation defined in such a way that:

$$N_G(x) = \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{otherwise.} \end{cases}$$

Definition 4.1.16. A function, $I : [0, 1] \times [0, 1] \rightarrow [0, 1]$, is called a *fuzzy implication*, whenever it satisfies the following properties:

1. $I(1, 0) = 0$ and $I(0, 0) = I(0, 1) = I(1, 1) = 1$;
2. if $x \leq z$, then $I(x, y) \geq I(z, y)$;
3. if $y \leq z$ then $I(x, y) \leq I(x, z)$.

Moreover, it satisfies the *order property*, whenever $x \leq y$ implies $I(x, y) = 1$.

One example is the Gödel implication defined in such a way that:

$$I_G(x, y) = \begin{cases} 1, & \text{if } x \leq y \\ y, & \text{otherwise.} \end{cases}$$

With respect to bi-implication there is no universal agreement on a fuzzy counterpart. The most well-known class of fuzzy bi-implications was investigated by Fodor & Roubens [24] who suggested the following definition:

Definition 4.1.17. A function $B : [0, 1] \times [0, 1] \rightarrow [0, 1]$, is called a *fuzzy bi-implication*, whenever it satisfies the following properties:

1. $B(x, y) = B(y, x)$;
2. $B(x, x) = 1$;
3. $B(0, 1) = 0$;
4. If $w \leq x \leq y \leq z$, then $B(w, z) \leq B(x, y)$.

Proposition 4.1.8. Given a T -norm, T and an implication I satisfying the order property, then $B(x, y) = T(I(x, y), I(y, x))$ is a fuzzy bi-implication.

Proof.

1. $B(x, y) = T(I(x, y), I(y, x)) = B(y, x)$.
2. $B(0, 1) = T(I(0, 1), I(1, 0)) = T(1, 0) = 0$.
3. $B(x, x) = T(I(x, x), I(x, x)) = T(1, 1) = 1$.
4. Suppose $w \leq x \leq y \leq z$, then $B(w, z) = T(I(w, z), I(z, w)) = T(1, I(z, w)) = I(z, w) \leq I(z, x) \leq I(y, x) = T(1, I(y, x)) = T(I(x, y), I(y, x)) = B(x, y)$.

□

$B_G(x, y) = T_G(I_G(x, y), I_G(y, x))$ is a bi-implication because the Gödel implication satisfies the order property.

Given these notions, we can introduce a logical language to reason about fuzzy switch graphs. It should be based in what was introduced previously for usual switch graphs but generalize it in order to admit a fuzzy formalism.

First of all, one must note that, in a fuzzy context, syntactical abbreviations like $p \rightarrow q \equiv \neg p \vee q$ make no sense, since fuzzy semantics may consider proper functions for disjunction, conjunction, negation, implication and bi-implication.

Definition 4.1.18. Given a set of atomic propositions Ω , the set of formulas $FFml(\Omega)$ is recursively defined as the least set such that $\Omega \subseteq FFml(\Omega)$, and $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$, $\varphi \leftrightarrow \psi$, $\Box\varphi$, $\Diamond\varphi \in FFml(\Omega)$ whenever $\varphi, \psi \in FFml(\Omega)$.

The notion of satisfiability is also different in a fuzzy context. Instead of a valuation function whose codomain is $\{0, 1\}$ we must consider a generalized one whose codomain is $[0, 1]$. Moreover, one must consider a fuzzy semantics $F = (T, S, N, U, B)$ consisting of a T-norm T , an S-norm S , a negation N , an implication U and a bi-implication B . This is described in the next definition. In this context, given a model $\mathcal{M} = (W, S, I, V)$, we also write $\mathcal{M}^{(w, w')}$ to represent $(W, S, I^{(w, w')}, V)$.

Definition 4.1.19. Given a set of atomic propositions Ω , a fuzzy switch graph model \mathcal{M} is a tuple (W, S, I, V) where $V : W \times \Omega \rightarrow [0, 1]$ is a fuzzy valuation function and (W, S) is a fuzzy switch graph together with an instantiation $I : S \rightarrow [0, 1]$.

Moreover, for such model and given a fuzzy semantics $F = (T, S, N, U, B)$, satisfiability over the set of formulas $FFml(\Omega)$ is defined recursively as:

- $\mathcal{M}, w \models_F p = V(w, p)$, for $p \in AtomProp$.
- $\mathcal{M}, w \models_F \varphi \wedge \psi = T(\mathcal{M}, w \models_F \varphi, \mathcal{M}, w \models_F \psi)$.
- $\mathcal{M}, w \models_F \varphi \vee \psi = S(\mathcal{M}, w \models_F \varphi, \mathcal{M}, w \models_F \psi)$.
- $\mathcal{M}, w \models_F \varphi \rightarrow \psi = U(\mathcal{M}, w \models_F \varphi, \mathcal{M}, w \models_F \psi)$.
- $\mathcal{M}, w \models_F \varphi \leftrightarrow \psi = B(\mathcal{M}, w \models_F \varphi, \mathcal{M}, w \models_F \psi)$.
- $\mathcal{M}, w \models_F \neg \varphi = N(\mathcal{M}, w \models_F \varphi)$.
- $\mathcal{M}, w \models_F \Box \varphi = \underset{(w, w') \in S_0}{T} \left(U \left(I(w, w'), \mathcal{M}^{(w, w')}, w' \models_F \varphi \right) \right)$.
- $\mathcal{M}, w \models_F \Diamond \varphi = \underset{(w, w') \in S_0}{S} \left(T \left(I(w, w'), \mathcal{M}^{(w, w')}, w' \models_F \varphi \right) \right)$.

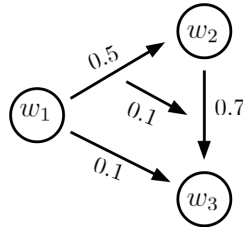


Figure 4.5: Example of a Fuzzy switch graph.

Example 4.1.5. Let $\mathcal{M} = (W, S, I, V)$ be a fuzzy switch graph model obtained from the fuzzy switch graph depicted in Figure 4.5 and $V : \{p\} \rightarrow [0, 1]$ be such that $V(w_2, p) = 1$ and $V(w_3, p) = 0.1$. Consider the Gödel fuzzy semantics, $G = (\min, \max, N_G, I_G, B_G)$ where N_G, I_G and B_G are Gödel's negation, implication and bi-implication, respectively. We illustrate how to evaluate a formula from $FFml(\{p\})$.

$$\begin{aligned}
 \mathcal{M}, w_1 \models_G \Diamond \Diamond p &= \max \left(\min (0.5, \mathcal{M}^{(w_1, w_2)}, w_2 \models_G \Diamond p), \min (0.1, \mathcal{M}^{(w_1, w_3)}, w_3 \models_G \Diamond p) \right) \\
 &= \max \left(\min (0.5, \mathcal{M}^{(w_1, w_2)}, w_2 \models_G \Diamond p), \min (0.1, 0) \right) \\
 &= \max \left(\min (0.5, \mathcal{M}^{(w_1, w_2)}, w_2 \models_G \Diamond p), 0 \right)
 \end{aligned}$$

$$\begin{aligned}
&= \min(0.5, \mathcal{M}^{(w_1, w_2)}, w_2 \models_G \Diamond p) \\
&= \min(0.5, \max(\min(0.1, (\mathcal{M}^{(w_1, w_2)})^{(w_2, w_3)}, w_3 \models_G p))) \\
&= \min(0.5, \min(0.1, 0.1)) = \min(0.5, 0.1) = 0.1
\end{aligned}$$

During the evaluation above, $\mathcal{M}^{(w_1, w_3)}, w_3 \models_G \Diamond p = 0$ because the maximum of an empty set is 0 and the weight of the edge (w_2, w_3) is 0.1 after (w_1, w_2) being crossed.

We point out that our definition of fuzzy switch graph could be even more general. In [59], where this concept is presented, aggregation functions (or simply *aggregations*) are considered. These are, generally, used to produce a representative data of its inputs. For example, *arithmetic, weighted and geometric means*, are canonical examples of aggregation functions; so are T-norms and S-norms [4]. In the context of fuzzy switch graph, its use can be justified to adjust the effect of one edge over another according to their fuzzy value (level of existence).

Definition 4.1.20. An n -ary function $A : [0, 1]^n \rightarrow [0, 1]$ is called an *aggregation function*, if it is isotonic, $A(0, \dots, 0) = 0$ and $A(1, \dots, 1) = 1$.

This notion is only considered when an edge is crossed and one needs to update a model. In particular, aggregations are used to come up with a more general procedure for obtaining an updated fuzzy weight for each edge. Given an aggregation A and an instantiation I , when the edge s is crossed, I is updated to:

$$I^s(t) = \begin{cases} A(I(s), I(t), I((s, t, \bullet))), & \text{if } (s, t, \bullet) \in S \text{ and } I((s, t, \bullet)) \neq \odot \\ \odot, & \text{if } (s, t, \circ) \in S \text{ and } I((s, t, \circ)) \neq \odot \\ I(t), & \text{otherwise.} \end{cases}$$

Note that the approach which has been used in this section is a particular case since projection functions: $\pi_j : A_1 \times \dots \times A_j \times \dots \times A_n \rightarrow A_j$, such that $\pi_j(x_1, \dots, x_j, \dots, x_n) = x_j$ are aggregation functions.

Finally, note that, structurally, probabilistic switch graphs can be seen as particular cases of fuzzy switch graphs because in a probabilistic approach, the set of weights is still $[0, 1]$. However, in these cases one additional condition must be imposed stating that, for each vertex, the weights of all outgoing edges sum 1. Nevertheless, probabilistic and fuzzy approaches are conceptually different. While a fuzzy weight can be understood as the degree of existence of an edge, a probabilistic one represents how likely it is to cross the respective edge. This conceptual difference will be taken into account in further sections where a probabilistic reactive structure is introduced and used.

4.2 Reactive Boolean networks.

In this section, a new qualitative model is introduced – *reactive Boolean networks* – which is particularly interesting to help in the quest for steady states. We start by presenting them, detailing how they can be obtained as a simplification of PWL models and then we prove that they preserve more information from the PWL model than the BN model obtained from the same model. For simplicity and to avoid inconsistencies, we only consider PWL models whose dynamics do not cause flows from two adjacent domains to converge to their common boundary.

4.2.1 Switch graph as a discrete reconfigurable system.

In Chapter 3, PWL models are presented as a reconfigurable and, in particular, hybrid models. When we consider a simplified BN model, we forgot both the reconfigurability and continuous dynamics. Thus, switch graphs are an intermediary step because they lose the continuous dynamics of a hybrid system but maintain the reconfigurability. Note that switch graphs can be seen as discrete transition systems (*i.e.* usual graphs) whose set of edges reconfigures after the discrete event of crossing an edge.

This kind of dynamics occurs frequently in real-life systems and even in games. For instance, castling is a chess special move involving the King and a Rook which can only be performed if none of the pieces taking part has been moved before. Therefore, in identical board configurations (which are states, in this context), different moves may be possible.

Next we describe the process to obtain a RBN from a PWL. In the following definition, consider a PWL model and the corresponding Boolean network obtained from it. There, for simplicity, when we mention a vertex x of the Boolean network we also mention the corresponding domain of the PWL model and *vice-versa*.

Definition 4.2.1. Given a PWL model M whose corresponding Boolean network is N , a *reactive Boolean network* is a switch graph (W, S) where $(W, S_0) = N$, $S_n = \emptyset$ for $n < 1$ and S_1 is obtained according to the following rules:

1. For any domain k of M such that $u = (j, k) \in S_0$, then $(v, u, \circ) \in S_1$ with $v = (i, j)$ if a flow which enters in region j *via* the boundary between regions i and j never leaves it *via* the boundary between regions j and k .
2. For each $(v, u, \circ) \in S_1$ with $u = (j, k)$ and $v = (i, j)$, then $(w, u, \bullet) \in S_1$ if there exists $w = (l, j) \in S_0$ for some region l of M such that there is a flow entering in region j *via* the boundary between regions l and j and leaving it *via* the boundary between j and k .

In practice, this new kind of models can temporarily delete edges that would represent non-realistic behaviors from the state transition graph. In practice, since we can compute the flow given by a linear differential equation and an initial state, the inclusion of an edge of the type $((i, j), (j, k), \circ) \in S_1$ means that is not possible to obtain, in the PWL model, a flow with initial state in the region i that enters in the region j and leads us to region k .

Example 4.2.1. In this example we illustrate how higher level edges are obtained. Consider the Figure 4.6 where three domains of a theoretical PWL model are found on the left side. In the figure, arrows represent flows. On the right side we can see the three corresponding vertices represented by black dots and usual edges representing the transitions between domains. Furthermore, two additional 1-level edges are considered, following the rules described in Definition 4.2.1.

In this case, we note that the flows in the domain C always leave this domain by the lower boundary whenever their initial state is in the boundary between A and C. This is the reason why the inhibitor higher-level edge is introduced. However, a flow within C whose initial state is in the boundary between the domains B and C can leave this domain by either the lower or right boundary. Thus, we introduce an activator higher-level edge to restore the temporarily removed edge. Clearly, if the edge is already active/present, the activator edge has no effect.

We only define reactive Boolean networks with switch graphs such that $S_n = \emptyset$ for $n > 1$. In fact, this definition could be generalized to embed higher-level edges.

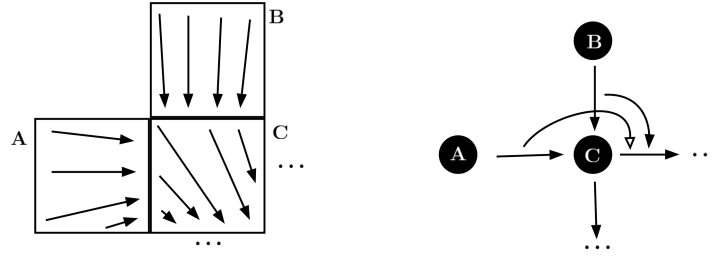


Figure 4.6: Obtaining higher-level edges for a RBN model.

4.2.2 Recovering attractors.

For the rest of this chapter, we only consider finite RBN models. Indeed, note that the biological systems we consider admit a finite number of components. Moreover, given the rules in Definition 4.2.1, the set of edges of every resulting RBN model is finite.

As mentioned before, it is known that all attractors found in simplified BN models signal steady states in the corresponding PWL models. In fact, this statement can be also obtained as corollary of Propositions 4.2.2 and 4.2.3 bellow. However, the opposite does not hold. Indeed, there are some steady states in PWL models that are not present in the respective simplified BN model. This is illustrated in the next example.

Example 4.2.2. Consider the following system of differential equations:

$$\begin{cases} x' = 5 \frac{x^2}{x^2 + 2^2} \cdot \frac{2^2}{y^2 + 2^2} - x \\ y' = 3 \frac{x^2}{x^2 + 4^2} - y \end{cases}$$

Making $n \rightarrow +\infty$ leads to

$\begin{cases} x' = -x \\ y' = -y \\ x < 2 \\ 2 < y \end{cases}$	$\begin{cases} x' = -x \\ y' = -y \\ 2 < x < 4 \\ 2 < y \end{cases}$	$\begin{cases} x' = -x \\ y' = 3 - y \\ 4 < x \\ 2 < y \end{cases}$
$\begin{cases} x' = -x \\ y' = -y \\ x < 2 \\ y < 2 \end{cases}$	$\begin{cases} x' = 5 - x \\ y' = -y \\ 2 < x < 4 \\ y < 2 \end{cases}$	$\begin{cases} x' = 5 - x \\ y' = 3 - y \\ 4 < x \\ y < 2 \end{cases}$

Analytically, two stable steady states can be identified at point $(0, 0)$ and an orbit which asymptotically converges to $(4, 2)$, as illustrated in Figure 4.7.

Figure 4.8 shows the simplified BN model corresponding to the PWL model described. We use the notation introduced before in such a way that, for example, the state 00 of the BN model represents the PWL model domain described by condition $0 \leq x < 2 \wedge 0 \leq y < 2$.

Steady states are signaled by attractors – strongly connected components with no outgoing edges. The BN depicted on this example admits two strongly connected components – $\{00\}$ and $\{11, 21, 20, 10\}$ – but only one of them is an attractor and, thus, signals a single steady state, because $\{11, 21, 20, 10\}$ admits an outgoing edge from 11 to 01. Thus, we lose

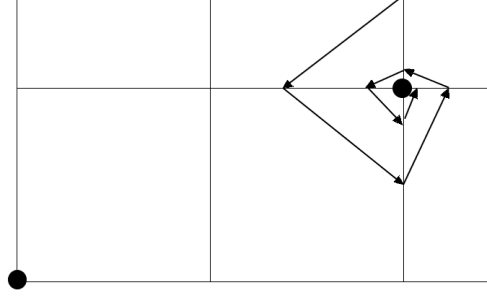


Figure 4.7: Illustrating the stable steady states.

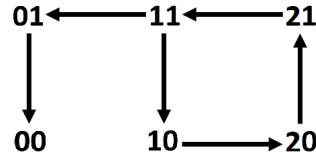


Figure 4.8: Simplified Boolean network.

information about the initial system since we are only able to retrieve a steady state of the corresponding PWL model.

RBN models are an intermediary step. Since RBN are also switch graphs, we can use some terms such as “induced reactive frame” and “plain representation” when discussing them. In this context, steady states are also identified by attractors, whose definition is generalized as follows.

Definition 4.2.2. Given a reactive Boolean network (W, S) whose set of paths of the induced reactive frame is Δ , we say that a set $V \subseteq W$ forms a *strongly connected component relatively to a path* $\lambda \in \Delta$ (SCC_λ) if, for every $v \in V$ and every path $\rho \in \Delta$ which extends λ , there exists $\gamma \in \Delta$ such that $t(\gamma) = v$ and γ extends ρ .

Proposition 4.2.1. *If V is a SCC_λ , it is always possible to find a path between two states $u, v \in V$ after the reconfiguration on edges induced by the path λ .*

Proof. From the definition, for all extensions ρ of λ , one can find γ_u an extension of ρ such that $t(\gamma_u) = u$. Again, by definition, and since γ_u itself extends λ , it is possible to find γ_v which extends γ_u and such that $t(\gamma_v) = v$. \square

Definition 4.2.3. Given a reactive Boolean network (W, S) whose set of paths of the induced reactive frame is Δ , a set $V \subseteq W$ is an *attractor* if it is a SCC_λ , for some path λ , and every path γ extending λ verifies $t(\gamma) \in V$.

These definitions extend the notions of SCC and attractor, which are defined for regular graphs (and BNs), to switch graphs (and RBNs). Informally, this means that a set V is an attractor if there is a path γ such that, after walking along it, we can always find a path between any two states of V and there is not any path guiding us to a final state outside the set V (*i.e.* the usual definition of an attractor in a graph).

Example 4.2.3. Recall Example 4.2.2 and the simplified RBN model for the PWL model introduced on that example. Firstly take the Boolean network introduced there and consider it as (W, S_0) , for our reactive Boolean network. From the PWL model we can follow the rules introduced in Definition 4.2.1 and obtain the set S_1 of higher-level edges, which contains a single element: $\{((21, 11), (11, 01), \circ)\}$. Thus, the resulting RBN model is depicted in Figure 4.9.

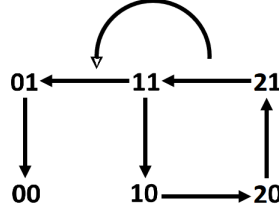


Figure 4.9: A reactive Boolean network.

For this reactive network, given 21 as the initial state, we obtain the following set of paths: $\{(21), (21, 11), (21, 11, 10), (21, 11, 10, 20), (21, 11, 10, 20, 21), \dots\}$. Therefore, according to Definition 4.2.3, $\{11, 10, 20, 21\}$ is an $\text{SCC}_{(21)}$ and, moreover, an attractor for this reactive Boolean network. Similarly, taking 00 as the initial state, $\{00\}$ emerges as an attractor as well.

Thus, we note that, from the RBN model, we are able to retrieve two attractors, as in the original PWL model

Our next results relate attractors from BN models with attractors of RBN models, in order to show that RBN models, in general, preserve more attractors than BN models. Consider a BN or RBN model (V, E) and a RBN model (V', E') . In this context, given an attractor $A \subseteq V$ of the model (V, E) , we say that it is *signaled* by the model (V', E') if there is an algorithm that allows to obtain a set $B \subseteq V'$ from A in an unambiguous way and such that B is an attractor of (V', E') . Using simpler words, if a model “A” signals the attractors of another model “B”, we can recover all attractors of the model “B” from the attractors of the model “A”.

Proposition 4.2.2. *Given a PWL model, the corresponding reactive Boolean network identifies, in general, a larger set of attractors than the simpler Boolean network. Moreover, all attractors of a BN are signaled in the corresponding RBN.*

Proof. The fact that, in general, a RBN admits a larger set of attractors than a BN was already shown in Example 4.2.3. Note that we were able to recover both attractors of the original PWL model, while the BN model was able to only recover one.

Now, consider a BN with an attractor V . If $|V| = 1$, then, trivially, V is also an attractor of the corresponding RBN. Otherwise, let $\lambda = (v)$ be a path with $v \in V$. Since V is an attractor in a BN, all extensions γ of λ are such that $t(\gamma) \in V$. Consider the following algorithmic procedure: Choose, if possible, $v \in V$ for which it is possible to consider an extension γ of λ such that it is no more possible to extend it to path ρ where $t(\rho) = v$. If such a path exists, update λ to γ and V to $V \setminus \{v\}$. Repeat this process while it is possible to choose such a v . Note that, since V and S are finite (*i.e.* there is a finite number of configurations for the S_0 edges), this algorithm terminates. Note that, after this process, for

any $v \in V$ such that there is an extension γ_v of λ such that $t(\gamma_v) = v$, and for every $w \in V$, there is an extension ρ_w of γ_v such that $t(\rho_w) = w$. This proves that V is a SCC_λ and, since each extension γ of λ is such that $t(\gamma) \in V$, the RBN signals an attractor. \square

Proposition 4.2.3. *All attractors in a RBN are steady states of the corresponding piecewise linear model.*

Proof. A steady state of a PWL can be found as either an invariant region or a cyclic behavior which asymptotically converges to a point or orbit. Let V be an attractor in a RBN, and consider a region T resulting from the union of every domain represented by $i \in V$. Since V is a SCC_λ for some path λ , this means that there is a flow in the piecewise linear model which makes impossible to leave region T . Thus, there exists an invariant subregion T' of T and, therefore, since the differential equations considered are linear, it contains a steady state. \square

Finally, we recover the notion of bisimulation and present it as a method to coherently reduce large models. Indeed, we prove that bisimulation preserves attractors. Bisimulation between RBN model is defined for switch graph models by considering an empty set of atomic propositions. We prove the following results over switch graphs in order to keep them general (we recall that RBN models do not admit n -level edges with $n > 1$).

From a model (W, S) , we can soundly obtain a reduced model (W', S') if there exists a bisimulation R verifying $\forall w \in W \exists w', wRw'$. In this case, we say R is *total*.

Lemma 4.2.4. *Let (W, S) and (W', S') be two switch graphs, (W, Δ) and (W', Δ') the corresponding reactive frames. Let \mathcal{R} be a total bisimulation between (W, S) and (W', S') and \mathcal{B} the induced relation from \mathcal{R} . Then \mathcal{B} is total whenever \mathcal{R} is total.*

Proof. We prove this lemma by induction over paths.

Let $\lambda \in \Delta$ be a path. If $\lambda = (w)$, for some $w \in W$, then, since R is total, there is $w' \in W'$ such that $(w, w') \in \mathcal{R}$ and, therefore $(w)\mathcal{B}(w')$.

Let us now consider a path $\gamma = \lambda w$ for some $w \in W$ and $\lambda \in \Delta$. Then, by induction hypothesis, there is $\lambda' \in \Delta'$ such that $\lambda\mathcal{B}\lambda'$. Then, since \mathcal{B} is a bisimulation and by definition, $\exists w' \in W'$ such that $\lambda w\mathcal{B}\lambda'w'$. \square

Proposition 4.2.5. *Let (W, S) and (W', S') be two bisimilar switch graphs. Each attractor of (W, S) is signaled by some attractor of (W', S') .*

Proof. Let (W, S) and (W', S') be two switch graphs whose corresponding reactive frames are (W, Δ) and (W', Δ') , respectively. Let also $R \subseteq W \times W'$ be a total bisimulation and T the corresponding bisimulation for reactive frames.

Consider A , an attractor of (W, S) . Thus, there is some path $\lambda \in \Delta$ such that A is a SCC_λ . According to the previous lemma and since \mathcal{R} is total, then \mathcal{B} is also total. Then, by definition, there is $\lambda' \in \Delta'$ such that $(\lambda, \lambda') \in T$. Let $\bar{B} = \{t(\gamma') : \gamma' \in \Delta' \text{ be an extension of } \lambda'\}$. By the definition of \bar{B} , and using a process analogous to the one presented in the proof of Proposition 4.2.2, we obtain an attractor $B \subseteq \bar{B}$. We will show that the states of B are related with the states of A .

If $b \in B$, then it means that there is an extension γ' of λ' such that $(\gamma') = b$, i.e. $\exists w'_0, \dots, w'_n$ such that $\gamma' = \lambda'w'_0\dots w'_n$. Since R is a bisimulation, we know that $t(\lambda)Rt(\lambda')$, $t(\lambda w_0)Rt(\lambda'w'_0)$, \dots , $t(\lambda w_0\dots w_n)Rt(\lambda'w'_0\dots w'_n)$, where $w_0, \dots, w_n \in W$ are such that $w_0Rw'_0, \dots, w_nRw'_n$. Since A is an attractor $w_0, \dots, w_n \in A$. \square

Finally, we note that, computationally, RBN is still a discrete model. Thus, many already developed approaches can be use on its study. We can think about their plain representation which is, in particular, a discrete automata with initial states. Following this line of thought, note that the size of these automata grows exponentially along with the number of higher-level edges considered. However, this is an expected drawback of increasing the recovered information from PWL models when comparing to the usual BN models.

4.3 Reactive Boolean networks with probabilities.

In this section we generalize the notion of Boolean networks with probabilities to include reactive dynamics. While RBN models are based on switch graphs, RBN models with probabilities are based on weighted switch graphs.

Definition 4.3.1. A RBN model with probabilities is a switch graph (W, S) with an instantiation $I : S \rightarrow [0, 1]$ such that:

- No inhibitor edges are considered.
- There are no n -level edges for $n > 1$.
- For each $w \in W$, either $\sum_{(w, w') \in S} I(w, w') = 1$ or w admits no outgoing edges.
- For each $w \in W$, if $(v, (w, w'), \bullet) \in S$ for some $v \in S, w' \in W$, then, for each fixed $v \in S$, $\sum_{(v, (w, w'), \bullet) \in S} I((v, (w, w'), \bullet)) = 1$.

This guarantees that the sum of the weights assigned to all edges leaving the same state is 1, as required in probabilistic contexts. Moreover we note that, in this context, $\odot \equiv 0$ and this is the reason why we do not need to consider inhibitor edges. As in RBN models, we do not consider edges whose level is higher than 1. This is, in fact possible, but a more careful definition would be needed to guarantee the coherence of a probabilistic model.

RBN models with probabilities can also be conceived as simplifications of PWL models and, in their turn, as generalization of BNs with probabilities. This is done by the process described next. We recall that when we mention a vertex X of the Boolean network we also mention the corresponding domain of the PWL model and *vice-versa*. Moreover, we denote by $B(X, Y)$ the boundary between the domains X and Y .

1. Obtain the usual BN model with probabilities as described in Section 2.2.2.
2. Consider a domain Y admitting more than one outgoing edge. For each state X such that (X, Y) is present in the model:
 - Consider flows starting at some state in $B(X, Y)$, then, for each edge (Y, Z) of the simplified RBN model, we define a function $\mathbb{I}_Z^{(X, Y)} : B(X, Y) \rightarrow \{0, 1\}$ such that, for each $x \in B(x, y)$, $\mathbb{I}_Z^{(X, Y)}(x) = 1$ if and only if there is a flow from x to a state in $B(Y, Z)$ that does not cross other boundaries.

- Introduce an edge $((X, Y), (Y, Z), \bullet)$ whose weight is $\frac{\int_{x \in B(X, Y)} \mathbb{I}_Z^{(X, Y)}(x) dx}{\int_{x \in B(X, Y)} dx}$

This provides a more accurate model when compared to the usual BN model with probabilities since we consider an additional layer of edges representing reactive behaviors obtained from a PWL model. This model can then be studied using its plain representation and some suitable methods, concepts and tools like Markov-chains.

Example 4.3.1. Consider the full PWL model from Example 2.1.3. When simplifying into a discrete BN model one must consider 3 qualitative levels for the variable x_a (contrarily to what is done in Example 2.2.3. The simplified BN model (without weights) is shown in Figure 4.10.

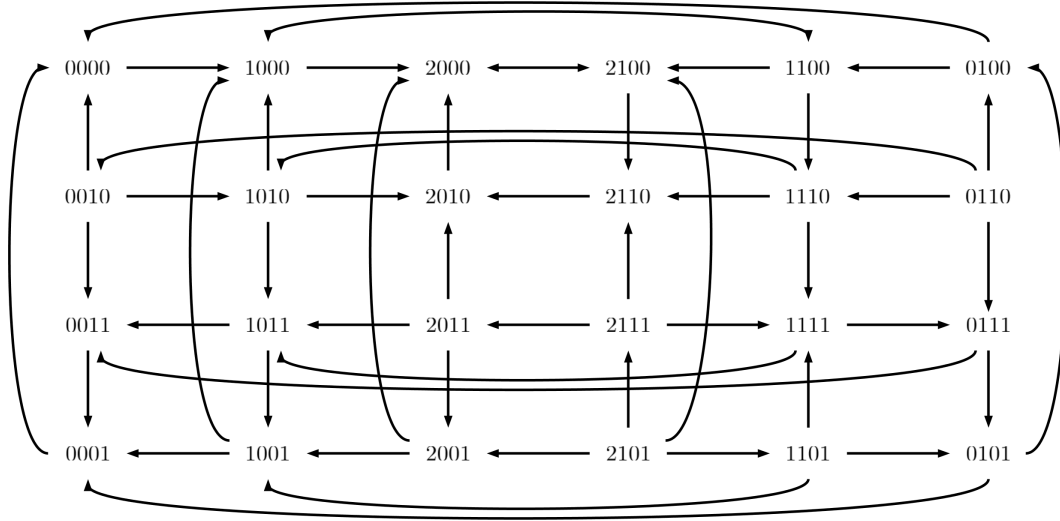


Figure 4.10: BN model for the PWL model in Example 2.1.3.

Let us consider a small part of this network. In Figure 4.11 we see a small part of the full BN model with probabilities. These probabilities were obtained as described in Section 2.2.2. Formally, the probabilities shown near each edge represent the value that the considered instantiation assigns to it. These probabilities are approximations and were obtained computationally.

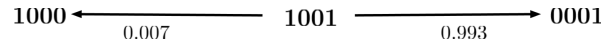


Figure 4.11: Part of a BN model with probabilities.

We note that, in the full model, there are three edges targeting state 1001 – (1011,1001), (1101,1001) and (2001,1001). Thus, we can follow the process described before to find the 1-level weighted edges and their weight. Again, the probabilities shown near each higher-level edge represent the value assigned to it by the considered instantiation. On the other hand, this is also the value it can assign to the target 0-level edge. The resulting partial RBN model with probabilities is shown in Figure 4.12. We note that when we enter in the PWL domain 1001 coming from either 1011 or 1101, we almost surely leave this domain to domain 0001. Nevertheless, when entering in 1001 by the adjacent domain 2001, the probability of leaving to 1000 doubles, despite still being small.

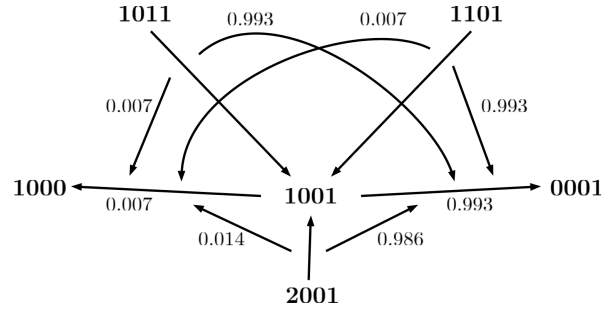


Figure 4.12: Part of a RBN model with probabilities.

4.4 Final remarks.

Reactivity is found in a wide number of contexts. Indeed, reactive systems are discrete reconfigurable systems in such a way that they can be seen as discretization of some classes of hybrid models. Thus, reactive models can describe a large number of systems in a natural way.

We focused our attention in switch graphs and switch graph models and we introduced a concept of bisimulation for these structures. Moreover, we compared it to the already existing notions of bisimulation for classical models, in order to show its coherence. Finally, we applied these concepts to develop a new kind of model for biological regulatory networks – reactive Boolean networks. We showed how to obtain this kind of model as a simplification of a PWL model and proved that more features can be preserved when compared to usual BN model. Although reconfiguration adds some complexity to the model, they are still discrete models and many tools used to study discrete models can be applied, because reactive models admit a plain representation, which consists of a regular automaton.

Chapter 5

Studying asymptotic dynamics in Boolean networks

In this chapter we focus our attention on the study of BN models and, in particular, on the process of looking for attractors. When dealing with large models, the asymptotic graph (AG) method is very useful. However, some problems can arise since false attractors (also called *spurious*) can appear in the asymptotic graph. In this chapter we study why this spurious attractors occur and improve this method in order to reduce their occurrence. The resulting method, called extended asymptotic graph (EAG) is then compared to the usual AG one. The work presented in this chapter was published in [13, 18].

5.1 Bisimulation and attractors.

In the previous chapter we proved that a bisimulation of BN models (in the particular case of RBN models) preserves attractors. We now will present a result which directly relates attractors with bisimulation.

In this chapter we consider only finite graphs. We also consider a bisimulation as a relation between states of the same graph. Therefore, we can impose an extra condition – a bisimulation must be an equivalence relation. This will be needed when introducing the concept of *quotient graph*.

Definition 5.1.1. Let (V, E) be a graph. We say that $\mathcal{B} \subseteq V \times V$ is a complete bisimulation if it is a bisimulation and there exists $B \subseteq V$ such that $\mathcal{B} = B \times B$ (any two elements of B are related).

A complete bisimulation \mathcal{B} is minimal if there is not any other complete bisimulation \mathcal{B}' such that $\mathcal{B}' \subsetneq \mathcal{B}$.

Lemma 5.1.1. Let (V, E) be a directed graph, $B \subseteq V$ and $\mathcal{B} = B \times B$ a minimal complete bisimulation. For any $A \subsetneq B$, $\exists a \in A, v \in B \setminus A$ such that $(a, v) \in E$.

Proof. Let us assume that there exists $A \subsetneq B$ such that, for any $a \in A, v \in B \setminus A$, $(a, v) \notin E$. In this case, we can easily verify that $A \times A$ is an equivalence relation since all states of A are related. By hypothesis, for any $(a, a') \in A \times A \subseteq \mathcal{B}$ such that $(a, b) \in E$, there exists some b' which verifies $(a', b') \in E$ and $(b, b') \in \mathcal{B}$. Since for any $a \in A, v \in B \setminus A$, $(a, v) \notin E$, we conclude that $b, b' \in A$ and, therefore, $(b, b') \in A \times A$. Thus, $A \times A$ is a complete bisimulation and this contradicts the minimality of \mathcal{B} . \square

Theorem 5.1.2. *Let (V, E) be a graph. $\mathcal{B} = B \times B \subseteq V \times V$ is a minimal complete bisimulation $\Leftrightarrow B$ is a terminal of V .*

Proof. “ \Rightarrow ”

We start by proving that if \mathcal{B} is a minimal complete bisimulation, then there exists a path between any two elements of B . We prove that B is a terminal afterward.

We consider $u, v \in B$. By Lemma 5.1.1, we know that there is an edge (u, u_1) from the vertex u to a vertex $u_1 \in B \setminus \{u\}$. If $u_1 = v$, we are done. Otherwise, because of Lemma 5.1.1, we know that there is an edge from a vertex in $\{u, u_1\}$ to some $u_2 \in B \setminus \{u, u_1\}$. Here, either (u, u_2) or (u_1, u_2) . In any case, there is a path from u to u_2 . Again, if $u_2 = v$ we are done. Otherwise, we can continue to apply this procedure until finding a path between u and v (this procedure will end in finite time since we are only considering finite graphs). As u and v were arbitrary, we can conclude that B is a SCC.

Finally, if $u \in B$ and (u, v) , then $(u, u) \in \mathcal{B}$ and, by definition of complete bisimulation, $(v, v) \in \mathcal{B}$. Then $v \in B$ and, thus, B is a terminal.

“ \Leftarrow ”

We now assume that B is a terminal of V . If B only contains an element, the result holds trivially. Otherwise, we can easily see that $\mathcal{B} = B \times B$ is a equivalence relation since all states are related. Consider $(u, v) \in \mathcal{B}$ and $(u, u') \in E$. Since B is a terminal, $u' \in B$ and $\exists v' \in B$ such that $(v, v') \in E$. Furthermore, $(v, v') \in \mathcal{B}$, by definition and, therefore, \mathcal{B} is a complete bisimulation.

Let us assume that \mathcal{B} is not minimal, *i.e.* there is a complete bisimulation $\mathcal{A} := A \times A \subsetneq \mathcal{B}$. Since B is terminal, it is possible to find a path from any $a \in A$ for any $b \in B \setminus A$. Thus, $\exists a' \in A, b' \in B \setminus A$ such that $(a', b') \in E$. But this contradicts the fact of \mathcal{A} being bisimulation because $b' \notin A$. \square

This theorem helps one to understand some properties of attractors which will be used further. It also provides a model reduction method which preserves attractors individually by clustering the states of an attractor into a single state. Given a graph (V, E) and a equivalence relation R , the *quotient graph* (\bar{V}, \bar{E}) is such that $\bar{V} = V/R$, *i.e.* \bar{V} is the set of equivalence classes and $([v], [w]) \in \bar{E}$ if $\exists x \in [v], y \in [w]$ such that $(x, y) \in E$.

If we find a bisimulation \mathcal{B} such that any complete bisimulation contained in \mathcal{B} is minimal, we can compute the quotient directed graph in order to obtain a minimized model in which the states of the attractors may be clustered. Nevertheless, these attractors are individually preserved.

Example 5.1.1 (Reducing a BN model for circadian rhythm). We recover the BN model in Example 2.2.3, relative to a model for circadian rhythm which is given by the following Boolean equations:

$$\begin{cases} a := \neg s \\ s := ts \\ t := a \\ ts := t \wedge a \end{cases}$$

With these equations we can construct the corresponding directed graph and, then, look for their attractors. The resulting asynchronous Boolean network, whose only attractor is highlighted by an orange box, is shown in Figure 5.1.

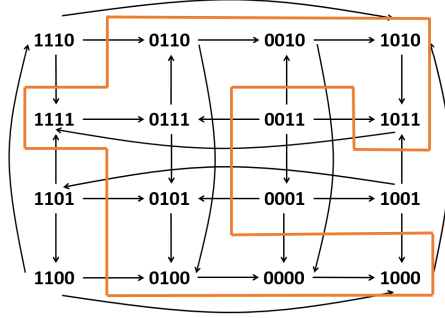


Figure 5.1: BN model of the circadian rhythm in a cyanobacteria.

With this, we can find a bisimulation such that all complete bisimulations contained on it are minimal and compute the quotient directed graph. Hence, if we consider the following bisimulation $\mathcal{B} = \{(a, b) : a, b \in \{0110, 0010, 1010, 1111, 0111, 1011, 0101, 0100, 0000, 1000\}\} \cup \{(a, a) : a \in \{1110, 1101, 1100, 0011, 0001, 1001\}\}$, the resultant quotient directed graph is the one which is presented in Figure 5.2.

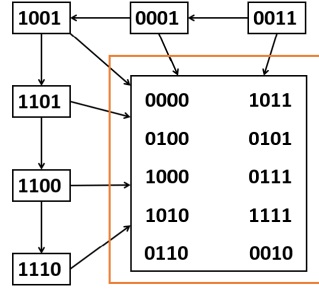


Figure 5.2: Reduced model for the circadian rhythm in a cyanobacteria.

A related approach.

A method to minimize Boolean networks which is widely used when searching for attractors is clustering the SCCs. This allows us to minimize a model and still preserve the attractors (which are SCCs). This is a well-known idea and several other methods to find attractors were developed after it. However, the vertices contained in an SCC are not necessarily bisimilar. Furthermore, when we reduce a BN model using bisimulations by obtaining a quotient graph, we can cluster states which would not be clustered by the SCCs clustering. On the other hand, when constructing the quotient graph, it may happen that we do not cluster all SCCs. We can see this because, for any SCC which is not a terminal, there exists some state in the SCC admitting a transition for a state out of that SCC. Therefore, it may be impossible to find a complete bisimulation to cluster all their states.

Finally, we point out an important feature of bisimulations. Since we are dealing with discrete state transition models (automata), it can be useful to use modal logic to reason about such models. Hence, it could be useful to obtain minimization processes which guarantee that all states in a cluster verify the same modal formulas. Indeed, due to their definition, bisimulations are suitable to be used for this purpose.

5.2 Extended asymptotic graphs.

The method of asymptotic graphs, as presented in Section 2.2.3, reduces a model and signals all attractors of the original model. The problem with this method is that spurious attractors can occur in the reduced model and to verify if they are, in fact, attractors of the original model is, in general, computationally costly. Thus, our goal would be to obtain a method which could reduce a model without creating spurious attractor and preserving all existing ones.

This was, in fact, already achieved in [68] where the concept of *cross graph* is introduced. Indeed, it is proven that all attractors of a BN model are preserved by its cross graph and no spurious attractors are created. However, the spatial and time complexity of building and analyzing a cross graph is usually even higher than the complexity of analyzing a usual BN model. This is because the method for obtaining a cross graph is similar to the one for obtain a asymptotic graph but, instead of only considering the attractors of the submodels, every SCCs must be considered. This is an issue because sets with a single state are themselves SCCs.

Thus, in this section we obtain an intermediary method – Extended asymptotic graph. This method considers all attractors – such as in AG – and some more vertices – as in cross graphs – in order to reduce the number of spurious attractors obtained but still maintains a small number of states.

5.2.1 Analyzing the AG method.

When we build an asymptotic graph, we consider the attractors of each submodel – which we call A and B – and operate a conceptual Cartesian product between the set of semi-attractor of each submodel $SAttr(A)$ and $SAttr(B)$. Then we obtain a new graph $(SAttr(A) \times SAttr(B), E)$ where the set of edges E is obtaining according to the rules described above. The proof of Theorem 5.1.2 and the connection between bisimulation and attractors provides some insights about the reason why some spurious attractors are created in asymptotic graphs. Fundamentally, this occurs because some paths between states of the original BN are lost when we consider the corresponding states of AGs. In this way, some SCC of a AG become terminals. We present an example to make this clearer.

Note that, even so, we can guarantee some attractors of an AG to signal real attractors of the original BN. In fact, some sufficient conditions are presented in some papers [11, 68, 15]. For instance, if an attractor of a AG is composed of a single element, then it signals a real attractor of the original BN model.

Example 5.2.1 (from [11]). Consider the toy model given by the following Boolean equations:

$$\begin{cases} a_1^+ = b_2 \wedge a_1 \wedge \neg a_2 \\ a_2^+ = (a_1 \wedge a_2) \vee (a_1 \wedge b_2) \vee (\neg a_1 \wedge \neg b_2) \\ b_1^+ = (b_1 \wedge \neg b_2) \vee (\neg a_2 \wedge \neg b_2) \vee (a_2 \wedge \neg b_1 \wedge b_2) \\ b_2^+ = (b_1 \wedge b_2) \vee (a_2 \wedge b_1) \vee (a_2 \wedge b_2) \end{cases}$$

The method for obtaining the AG consists of considering two subsystems A and B composed of the sets of variables $\{a_1, a_2\}$ and $\{b_1, b_2\}$, respectively. Then, we note that, in this



Figure 5.3: BN models for the subsystems A and B .

context, $h_A(B) = \{b_2\}$ and $h_B(A) = \{a_2\}$. The resulting graphs for these subsystems are represented in Figure 5.3

Thus, the sets of semi-attractors are $SAttr(A) = \{A_{1,1}^0, A_{1,0}^1\}$ and $SAttr(B) = \{B_{1,0}^0, B_{1,1}^1, B_{2,0}^1\}$. With these sets we can obtain the asymptotic graph which is shown in Figure 5.4

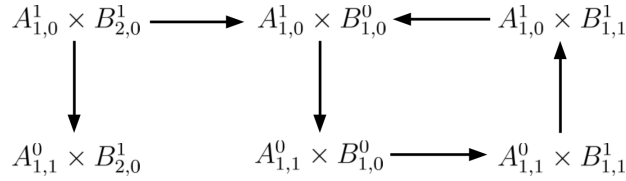
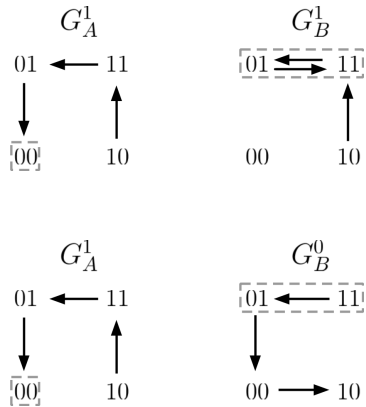


Figure 5.4: Asymptotic graph with a spurious attractor.

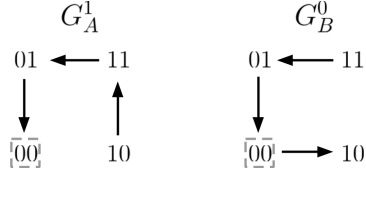
Analyzing this AG we find two attractors: $\mathcal{A}_1 = \{A_{1,1}^0 \times B_{2,0}^1\}$ and $\mathcal{A}_2 = \{A_{1,0}^1 \times B_{1,0}^0, A_{1,0}^1 \times B_{1,1}^1, A_{1,1}^0 \times B_{1,0}^0, A_{1,1}^0 \times B_{1,1}^1\}$. At this point, we can already be sure that, even not knowing the original BN, \mathcal{A}_1 is an attractor because it only contains one vertex, which is a sufficient condition for signaling a real attractor, as mentioned before. However, while \mathcal{A}_1 corresponds to the attractor $\{0100\}$ which is also present in the original BN models, \mathcal{A}_2 finds no correspondence in the original BN. Thus, \mathcal{A}_2 is a spurious attractor of the AG represented in Figure 5.4.

Let us explore the reason for this spurious attractor occur in this model. Due to Theorem 5.1.2, one already knows that this is due to some paths from the original model being ignored. Indeed, consider the vertex $A_{1,0}^1 \times B_{1,1}^1$ of the asymptotic graph. Bellow we illustrate how it is possible to find a “hidden” path between this vertex and $A_{1,1}^0 \times B_{2,0}^1$.

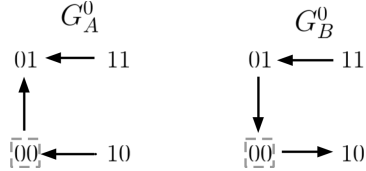


Consider the corresponding states on the graphs of the respective subsystems, given the considered input. Here we note that output of $A_{1,0}^1$ does not agree with the input of G_B^1 . Therefore, we consider the same states of $B_{1,1}^1$ but in the graph G_B^0 .

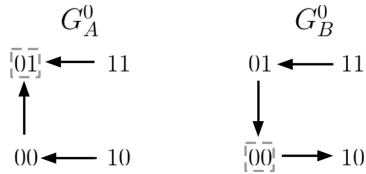
In G_B^0 , $\{01, 11\}$ is no more a (semi-)attractor, then we can move along the graph till the vertex 00.



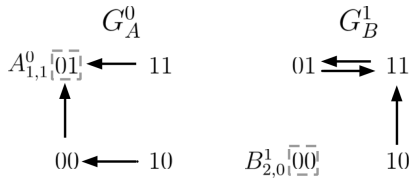
Following the usual method to obtain a AG method, we would directly move to the reachable semi-attractor(s) of G_B^0 and not consider this scenario. However, we note that, at this point, the output of 00 in the graph G_B^0 does not agree with the input of G_A^1 . Thus we consider 00 in G_A^0 .



Now, both 00 in G_A^0 and 00 at G_B^0 are not steady states, thus we could move in either one or another. For the purpose of this example, we move 00 to 01 in G_A^1 .



Moving 00 to 01 in G_A^1 , we reach a state whose output does not agree with the input of G_B^0 . Thus, we consider 00 in the graph G_B^1 .



Here, both 01 and 00 are semi-attractors of the corresponding graphs. Thus, we found a lost edge on the graph of the asymptotic graph. Moreover, the inclusion of this edge removes the spurious attractor, since \mathcal{A}_2 is no more an attractor of the asymptotic graph.

Thus, as hinted by the proof of Theorem 5.1.2, in order to preserve all attractors and avoid the creation of new ones, all paths between two related vertices should be somehow preserved on the resulting graphs of the different approaches. In Example 5.2.1, this was not the case for the AG since a path between the states 0001 and 0100, present in the original BN model, is not described in the corresponding asymptotic graph.

In order to obtain an extended method as an alternative to AG that does not create spurious attractors, one must ensure that all possible paths are somehow represented in the graph obtained by an extended method. One hypothesis would be to compute all paths which were possible to be obtained, using a process similar to the one described before. However, this would be computationally unreasonable, since one would obtain a combinatorial problem, thus with exponential complexity. Therefore, our extended approach considers additional vertices to the AG instead. In this way, we are in an intermediary step between asymptotic graphs and cross graphs.

Before presenting the extended algorithm, we discuss some preliminary considerations. Note that spurious attractors may appear due to output changes along the pathways followed by the states of a semi-attractor $A_{i,v}^u$ in a state transition graph $G_A^{\bar{u}}$ with $\bar{u} \neq u$. These pathways and the corresponding sequence of outputs are “forgotten” in the construction of the AG (which keeps only the final semi-attractor of the pathway, by definition). This is exactly what happened in Example 5.3, as illustrated before. Thus, our extended method considers some additional vertices for or AG, thus obtaining a extended asymptotic graph (EAG), and these added vertices correspond to those whose value of the output changes.

We now formally describe the improved algorithm. The idea is to analyze the pathways of each state of each semi-attractor $X_{i,v}^u$ along the same subsystem graphs $G_A^{\bar{u}}$ with $\bar{u} \neq u$, until a change in the output is detected. This is described in the following way:

1. Divide the set of components of a BN into two disjoint subsets in order to obtain two subsystems. For each subsystem A and B consider all possible values for the inputs and build the graphs G_A^u and G_B^u .
2. Compute the SCCs of each graph G_A^u and their attractors. Collect its semi-attractors in $SAttr(A)$.
3. Pick an attractor $A_{i,v}^u$, suppose it contains the states $\{x_1, \dots, x_k\}$, and look for all the possible forward pathways for each x_ℓ in the other graphs $G_A^{\bar{u}}$ with $\bar{u} \neq u$. For instance, consider a path $(x_\ell, y_1, y_2, \dots, y_n)$ where y_n belongs to an attractor in $G_A^{\bar{u}}$. Along each path, each state has its own output $h_A(y_i)$.
4. For each pathway of each state x_ℓ , pick the first state $y_{\bar{i}}$ that has a different output from x_ℓ , *i.e.* $h_A(y_{\bar{i}}) = \bar{u} \neq u = h_A(x_\ell)$.
5. State $y_{\bar{i}}$ belongs to some SCC $S_i^{\bar{u}}$ of the graph $G_A^{\bar{u}}$. Divide this SCC according to the outputs of their elements and add these sets $S_{i,v}^{\bar{u}}$ to the set of semi-attractors: $SAttr(A) = SAttr(A) \cup \{S_{i,v}^{\bar{u}}\}$.
6. Repeat steps 2-5 for the subsystem B . The set of vertices of the EAG is the Cartesian product between the set of semi-attractors of each subsystem.
7. To obtain the set of edges, apply the method for usual AGs, as described in Section 2.2.3, to every state of the EAG.
8. Consider a state $\bar{A} \times \bar{B}$ with $\bar{A} \in SAttr(A)$ and $\bar{B} \in SAttr(B)$. If $\bar{A} = S_{i,v}^u$ is an SCC but not a semi-attractor then look on graph G_A^u for every other $X \in SAttr(A)$ reachable from $S_{i,v}^u$ and consider an edge between $\bar{A} \times \bar{B}$ and $X \times \bar{B}$. Thereafter, we proceed analogously for \bar{B} .

To test this method, it was applied to 750 randomly generated pairs of modules. The modules comprise 2, 3, 4, 5 or 6 and up to 3 inputs/outputs. Each module was obtained by randomly generating the Boolean truth tables for G_X^u , $u \in \{1, \dots, 2^{p_X}\}$, where p_X is the number of inputs of the subsystem X . The results obtained with this extended method can be seen in Figure 5.5) and indicate that the spurious attractor problem is solved in about 99.8% of the cases, resulting in an extremely low occurrence rate. Nevertheless, the extended asymptotic graphs are still not exact and they are harder to calculate when comparing to usual asymptotic graphs. The test propose that extended asymptotic graphs become efficient only for systems whose subsystems admit, at least 6 components. In Figure 5.5, we compare the AG method with the EAG one. On the left, the percentage of failure (*i.e.* the generation of spurious attractors) is shown. On the right, we can see the sizes of both asymptotic graphs relative to the size of the full BN. To simplify the notation we denote by N_A and N_B the number of components of each subsystem, G^{as} denotes the graph for the AG method, G^{ext} denotes the graph of the EAG method and N_i denotes the number of inputs.

These results also suggest that the addition of other SCCs to the sets $SAttr(A)$ and $SAttr(B)$ will further improve the asymptotic graph. For instance, by successively adding the SCCs corresponding to a second or third change of output along the pathways within each graph G_A^u . This would make our algorithm more precise but computational complexity would increase, only becoming efficient at even higher dimensions. Indeed, this agrees with

the result shown for cross-graphs, which recovers exactly the same attractors of the original BN model. The cross graph is at least as costly to compute as the BN model itself, not being an efficient method for large networks. Nevertheless, the successive constructions AG, EAG, ..., cross graph provide a clear illustration of the organization of asynchronous dynamical behavior.

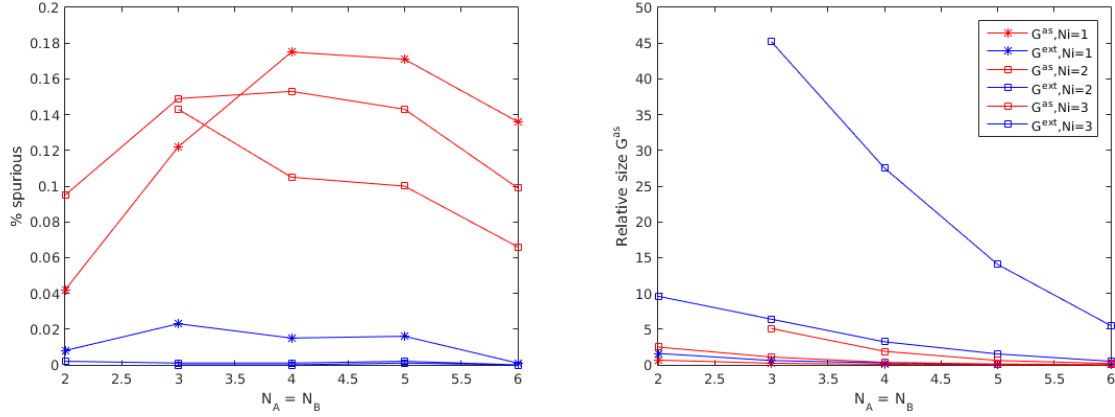


Figure 5.5: Performance of the extended asymptotic graph method for asynchronous networks and comparison with the AG original method.

Example 5.2.2. Consider the system in Example 5.2.1 where a spurious attractor occurred. In this example we obtain the extended asymptotic graph in order to show that the spurious attractor obtained in the AG of Example 5.2.1 does not occur in the respective EAG.

In order to obtain the EAG we consider the graphs for the subsystems presented in Figure 5.6. There, a dashed line divides the states of each graph according to their output. Then, following the method described before for obtaining an EAG, we look on each graph of the same subsystem but with different output for SCCs which occur in the pathways of the states of each semi-attractor, until a change in the output is detected. For these graphs, the only SCC which fulfills these requisites is 00 in G_B^0 , which we denote by $S_{2,0}^0$. This SCC is obtained by considering $B_{1,1}^1$ in the graph G_B^0 . For both elements – 01 and 11 – of $B_{1,1}^1$, 00 is the first state in their pathways with a different output.

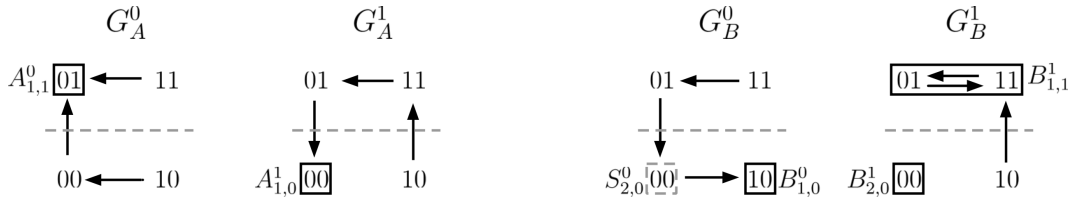


Figure 5.6: Auxiliary graphs for the construction of an EAG.

We thus obtain the extended sets $SAttr(A) = \{A_{1,1}^0, A_{1,0}^1\}$ and $SAttr(B) = \{B_{1,0}^0, B_{2,0}^1, B_{1,1}^1, S_{2,0}^0\}$. The set of edges is obtained as before. Thus, the resulting EAG is shown in Figure 5.7. While most of the construction follows the rules presented before, the novelty is the presence of vertices $A_{1,0}^1 \times S_{2,0}^0$ and $A_{1,0}^1 \times S_{2,0}^0$. Some edges like $(A_{1,0}^1 \times S_{2,0}^0, A_{1,0}^1 \times B_{1,0}^0)$

are also obtained using a specific rule for EAGs. Note that the input for the graph G_B^0 , where $S_{2,0}^0$ is found is 0, just like the output of $A_{1,0}^1$. However, since $S_{2,0}^0$ is an SCC but not an attractor, we must also consider the pathways from this vertex in G_B^0 .

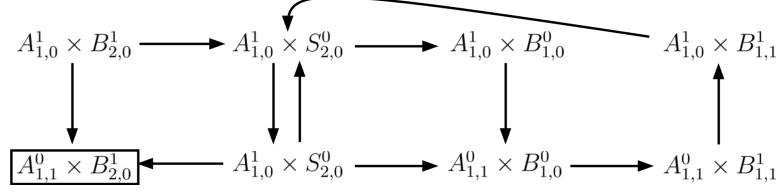


Figure 5.7: An extended asymptotic graphs with no spurious attractors.

For the obtained EAG, we can find a single attractor $\{A_{1,1}^0 \times B_{2,0}^1\}$, marked in Figure 5.7 by a rectangle. This is, indeed, a real attractor of the original system, as explained in [11]. This also means that this extended method was able to remove the spurious attractor obtained by the AG in Example 5.2.1. Moreover, in this example we are sure that, even not knowing the original BN, $\{A_{1,1}^0 \times B_{2,0}^1\}$ signals a real attractor of the full BN, because we still can use the sufficient conditions mentioned before for asymptotic graphs and $\{A_{1,1}^0 \times B_{2,0}^1\}$ contains a single vertex of the EAG.

5.3 Final considerations.

In this chapter we presented an analysis of BN models and a method to find their attractors. With this we highlighted some connections between bisimulation and attractors, and proposed an extension to the AG method.

The concept of complete bisimulation provides a theoretical notion for this concept. Moreover, it shows that methods for reducing BN models which “preserve” complete bisimulations make possible to identify all attractors of the original one. One example of a reduced BN model is the one obtained by quotient graphs.

A method which is also used to find attractors of BN models is to construct the corresponding asymptotic graph, which generally generates a graph with a reduced size preserving all attractors of the original BN individually. However, some spurious attractors can occur on these structures. Although some specific properties can be used to check that some attractors of an asymptotic graph are indeed attractors of the original system, many others are not easy to verify. Thus, we proposed an extended version of this method which we called extended asymptotic graph. This method generates a larger graph but the generation of spurious attractors is clearly reduced. Moreover, just like in AGs, we still can use the same properties to check that attractors of an EAG are, in fact attractors in the real model.

Unfortunately, this method is still not perfect because the occurrence of spurious attractors, despite being highly reduced, is still possible. This could be fixed by gradually augmenting the size of the EAG, through the inclusion of new vertices (*i.e.* SCC of the original model) until we obtain the full cross graph, as introduced in [68]. This cross graph preserves all attractors of the original model and does not generated spurious attractors. However, it is not useful since it is, in general, larger than the original BN. Because of this, we believe that our EAG method is a good option for large BN, since it becomes efficient for systems accommodating more than 13 components.

Chapter 6

A reactive approach to stochastic methods

We end the thesis with a chapter about a stochastic approach. Stochasticity is not a topic approached by M. Chaves in [10] but with the development of reactive formalisms, some new possibilities become attractive to be considered in a stochastic way. Hence, in this chapter we consider weighted switch graphs, as introduced in Chapter 4, as structures which make possible to introduce reactivity in stochastic models. Summarizing, in this chapter we start by illustrating how these models can be used in diverse biological contexts (apart from solely biological regulatory networks), we recall the notion of RBNs with probabilities and, finally, introduce rPrism, which is an extension to the tool *PRISM model checker* (check [46] for more details), and allows one to perform simulations for these reactive and stochastic models. The work presented in this Chapter was published in [22].

6.1 Stochasticity and reactivity in biological context.

As explained in previous chapter, all cellular processes occurring in biological organisms follow stochastic rules, since they obey to chemical and physical laws. Nevertheless, deterministic models like those presented in Chapter 2 which are, in general, simpler and easier to apply, can be used because of the law of large numbers, since cells admit a large number of elements for each component. Nevertheless, one must note that this is not always the case, *i.e.* there exist some biological processes which are guided by a fewer number of elements. These processes must be studied carefully using stochastic methods.

On other hand, reactivity is itself very common in biochemical examples. Apart from the cases presented before, in biological regulatory networks, we can find many other examples of reactivity at a larger physical scale. A simple example is the vaccination process. Upon vaccination, which can be considered a discrete event, a susceptible individual acquires a lower probability of becoming infected if he eventually contacts with a virus. A slightly more complex example is the Rhesus incompatibility between the fetus and the mother (see [39]). This incompatibility may occur when the mother is Rhesus negative and the fetus is Rhesus positive. If the mother's blood already contacted with Rhesus positive blood, the immune system of the mother has already developed antibodies against it and her blood will not be compatible with the fetus, and the immune system of the mother will attack the baby. If no contact happened before, the pregnancy may expect no complications since no antibodies were

created and the blood of the mother is not expected to contact with the blood of the fetus until his birth. In other words, the contact with Rhesus positive blood alters (or “reconfigures”) the mother’s immune system, creating the reactive behavior mentioned above.

In this chapter we consider a tool to be applied when studying systems where both stochasticity and reactivity are presented. We model this kind of system with a weighted switch graph. However, in this context, the weights assigned to edges of the considered reactive model describe rates. This will become clearer with the presentation of some examples. It is common to relate this approach with Markov chains but we note that reactivity introduces, in some sense, a notion of “memory” and reconfigurability in the system, which is not the case for Markov chains. Nevertheless, a connection is established further when the tool – rPrism – is introduced.

Example 6.1.1. A biological system which could be described by reactive and stochastic formalisms is the *cooperativity of a hemoglobin protein* and it is described in [16]. A single hemoglobin protein can bind up to 4 oxygen molecules. Thus, although this seems a simple system, it needs to be described by a model which accommodates features such as counters. Weighted switch graphs are perfect for the case. Indeed, the cooperativity of hemoglobin is characterized by the fact that, when this protein binds one oxygen molecule, the likelihood of the protein to bind to another one increases (up to a maximum of four). These changes in the rate of the oxygen binding can be described using the reactive properties of weighted switch graphs. Also, we note that this system can not be described by a deterministic method since we are considering a single hemoglobin protein.

The example of a weighted switch graph describing a hemoglobin protein is shown in Figure 6.1. There, each loop represents the binding of one oxygen molecule and the weight of the loop represents the corresponding rate: note that the weight increases for the binding of successive oxygen molecules, but a fifth molecule can no longer bind.

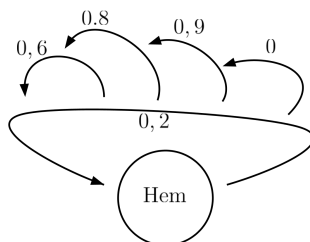


Figure 6.1: Model representing the cooperativity of hemoglobin.

6.2 The tool: rPrism.

The tool presented in this section – rPrism – is an extension of the PRISM model checker to include reactivity. This is done by translating a weighted switch graph into a usual weighted automaton and, then, use the tool PRISM. Before introducing it with more detail, we provide some insights about the *PRISM model checker* itself.

6.2.1 PRISM model checker.

PRISM is a free computational tool which is able to formally model and analyze systems which exhibit random or probabilistic behavior. It can be used to perform simulation but also admits temporal logics that allows to check several quantitative properties of modeled systems. It can deal with either discrete or continuous time Markov chains as well other probabilistic approaches. In order to model a probabilistic system, PRISM offers a proper language which is state oriented. This language considers several variables which can take values within a range of natural numbers, specified for each variable. Each state of the model is obtained as a specific admissible combination of values for the variables. The transition between states is directed by some rules. These transitions are described in a different way according to the type of probabilistic model considered. In this chapter we consider only continuous-time Markov chains. Each transition can have a label and is composed of:

- A “guard”, *i.e.* a condition over the values of the variables involving Boolean expressions and comparisons between integers. If a guard is not satisfied, then the transition cannot be triggered;
- A “rate” (in continuous scenarios), which is a positive real number determining the average time between two occurrences of the transition. If no value is specified, the default value for this field is 1;
- An “assignment” describing how the values of the variables change whenever the transition triggers.

Finally, we must also define the set of variables and the initial state/configuration of the system, *i.e.* the initial values of the variables.

An example of how to define a variable in PRISM language is:

```
VariableName : [0..10] init 5;
```

where 0 and 10 are the lowest and greatest values that the variable can take, respectively, and 5 is its initial value.

An example of how to define a transition in PRISM language is:

```
[TransitionLabel1] (VariableName>0) & (VariableName<7) -> 5 : true;
```

where $(\text{VariableName} > 0) \ \& \ (\text{VariableName} < 7)$ is a guard, 5 is the rate (which must always be a value in \mathbb{Q}^+) and `true` is an empty assignment. Another example is:

```
[TransitionLabel1] true -> VariableName'=3;
```

Here, `true` is a guard which is always satisfied, the rate is 1 by default, and `VariableName'=3` is an assignment which assigns 3 to the value of variable `VariableName`.

We can only define variables and transition inside modules. A variable can only occur within the module where it is defined. Thus, every transition (unambiguously identified by its label) must specify on each module how it depends and affects the variables defined on the module. This gives rise to many parallel guards and assignments. If that is the case, all guards must be satisfied in order to trigger the transition and all assignments occur simultaneously

for every module when the respective transition is triggered. The rate of a transition does not need to be presented on every module but may not exist any incoherence, *i.e.* it should be presented exactly once or none (being considered 1 by default). Finally, the PRISM language allows us to define constants (variables with fixed values) and rewards. Rewards are real values associated to certain states or transitions of the model.

This tool has already been applied to biochemical contexts and problems (see [37] for some examples). In the PRISM model checker website¹ an example of a stochastic model for a circadian rhythm model can be found, which is based in a ODE model which documented in [69]. More information about modeling biochemical processes using continuous-time Markov chains can be found in [47].

6.2.2 Weighted graphs and PRISM models.

When we compare weighted switch graphs to PRISM models, we can find some connections. On the one hand, each state of a PRISM model represents a combination of values for some variables. On their turn, a notion of transition between these states is considered, which provides a graph-like structure that considers vertices as states and edges as transitions. Moreover, the PRISM language also considers other features for transitions: guards, rates and assignments. Indeed, guards and assignments are implicitly represented in the graph structure, since they are responsible for deciding whether an edge is present in the graph or not. However, the inclusion of rates is only attained when we consider weighted edges, where weights represent rates. In this way, we note that a PRISM model can be translated into a weighted graph.

Example 6.2.1. Recall the hemoglobin protein example from Example 6.1.1. The resulting plain representation is illustrated in Figure 6.2. This model is, in this representation, a simple weighted graph which fully describes the dynamics of a single hemoglobin protein.

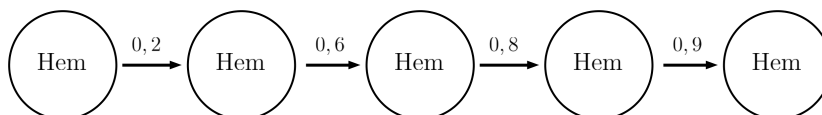


Figure 6.2: Plain representation for a hemoglobin cooperativity reactive model.

In order to use PRISM to study a weighted switch graph, we can translate this reactive structure into a weighted graph. Indeed, as mentioned before, a weighted switch graph admits a plain representation, thus becoming a weighted graph. Moreover, the method for obtaining this plain representation considers each vertex of the graph as a pair (x, I) where x is a vertex of the weighted switch graph and I is an instantiation. Thus, in order to accommodate this plain representation in the PRISM language, we must add a new variable I to the system in order to determine the relevant instantiation. Thus, two different instantiation always generate different states independently of the value of all other components and we can assign different rates to the transition from states which only differ in the instantiation. Hence, both higher-level edges and an initial instantiation can be encoded in the PRISM language using an additional variable I for instantiation and an initial state, respectively.

¹<https://www.prismmodelchecker.org/tutorial/circadian.php> (lastly accessed in August 29, 2019)

Note that it becomes possible to translate and use PRISM to study RBNs with probabilities, as introduced in Chapter 4. Nevertheless, this tool is not adequate for dealing with this kind of model. Indeed, we must take two things into consideration: Firstly, we note that RBN variables are Boolean and, therefore, only consider two values (unless more qualitative levels are considered). Moreover, transitions between vertices of a RBN do not represent activation and inhibition between components. The problem of this is that it seems not easy to understand, for each particular system, how edges from a RBN with probabilities can be encoded in the PRISM language because one must define transitions in the PRISM language which describe the same set edges of the corresponding RBN plain representation.

6.2.3 Biological regulatory networks as weighted switch graphs.

Consider the graph diagram formalism presented in the beginning of Chapter 2. In this section, we generalize it to a weighted graph and, thereafter, to a stochastic model. In order to do this, we start by analyzing these graph diagrams. Remember that a graph diagram is a graph (V, E) where the set of vertices coincide with the set of components and the set of edges $E \subseteq V \times V \times \{+, -\}$ describes the regulation relations. Although we can assign a rate to a positive regulation, illustrated as an usual edge, the same cannot be done when working with edges describing negative regulations, since no rate can be assigned. However, we can think about what a negative regulation implies: the presence of a component inhibit the production of another one. *I.e.* if some specific component is being produced and it inhibits other components, the second one is produced at a lower rate. This naïve approach is the basic idea for obtaining a reactive weighted graph from a graph diagram, representing a biological regulatory network. In this case it makes no sense to consider the plain representation for these models because we may consider several elements for each component, which means that the PRISM model would admit many values for each component variable and its graph representing state transitions would not coincide with plain representation.

Example 6.2.2. Consider the graph diagram in Figure 2.1 representing the regulatory network of circadian rhythm. We can assign weights to edges representing positive regulations, as illustrated in Figure 6.3 (left). The edges with weights represent phosphorylation reactions and weights denote rates. We note that the edges $(T, U, -)$, $(TS, U, -)$ and $(S, U, -)$ represent natural dynamics. Note that T , TS and S are phosphorylated for of unphosphorylated KaiC (U) because, if the total amount of KaiC is constant, then the presence of T , TS and S negatively regulates U . This is because the value of U decreases when values of T , S and TS increase. Nevertheless, the edge $(S, A, -)$, which is also present in the model does not describes any of this phenomena but a real inhibition in the transcription of the protein KaiA. Thus, since KaiA, in its turn, induces the production of T and TS without being consumed during the process, we replace both A and the edges representing inhibitions by higher-level edges. In this way, we obtain a weighted switch graph and the weights in the edges represent rates, as shown in Figure 6.3 (right).

We must note that, in this example, these rates where not estimated and were only introduced to provide an example. Nevertheless, more accurate values could be obtained with estimation.

This kind of representation, where each state represents a components and admits a number of elements, along with edges representing biochemical reactions and processes resembles to Petri nets [66] which can also be studied as stochastic models for biological processes [32].



Figure 6.3: A graph diagram of circadian rhythm with some weights representing rates (left); and a complete weighted switch graph obtaining by removing inhibitor edges and component A (right).

Indeed, we note that, apart from what concerns to specification, there not much difference between Petri nets and PRISM models. However, the main difference between these structures and weighted switch graphs is the inclusion of higher-level edges, allowing rates to change.

Finally, we consider two examples of a biological models and show how a weighted switch graph model can be analyzed in PRISM.

Example 6.2.3. Consider the weighted switch graph model introduced in Example 6.2.2. In order to obtain a model which can be specified by PRISM language we start by noting that it admits two instantiations: I_1 , the initial one which is represented in Figure 6.3; and I_2 , which is equal to I_1 except that $I_2((U, T)) = 0.2$, $I_2((T, TS)) = 0.2$ and $I_2((TS, S)) = 0.6$. Thus, in order to specify this model using the PRISM language, we must consider an additional variable which indicates us the instantiation that should be considered.

Since we are studying a stochastic model, we only consider 100 KaiC proteins in this model (both phosphorylated or unphosphorylated). Given this, we present the PRISM code for this model and explain it afterward:

```
ctmc

module components

u : [0..100] init 50;
t : [0..100] init 24;
s : [0..100] init 26;
ts : [0..100] init 0;
inst : [0..1] init 0;

[ts_s0] (ts>0) & (inst=0) -> 0.4 : (inst'=1) & (ts'=ts-1) & (s'=s+1);
[ts_s1] (ts>0) & (inst=1) -> 0.6 : (ts'=ts-1) & (s'=s+1);
[s_u] (s>0) -> 0.3 : (inst'=0) & (s'=s-1) & (u'=u+1);
[u_t0] (u>0) & (inst=0) -> 0.4 : (u'=u-1) & (t'=t+1);
[u_t1] (u>0) & (inst=1) -> 0.2 : (u'=u-1) & (t'=t+1);
[t_ts0] (t>0) & (inst=0) -> 0.4 : (t'=t-1) & (ts'=ts+1);
[t_ts1] (t>0) & (inst=1) -> 0.2 : (t'=t-1) & (ts'=ts+1);

endmodule
```

First of all, we begin by specifying which kind of approach we want. By writing `cmtc`, we choose “continuous-time Markov chain”. Thereafter, we must specify a module in order to define variables. This is done using the text `module components`, where “components” is the name of the module. Within this module we define all variable and transitions, as explained before. We note that the initial valued for variables u , t , ts and s sums to 100 and that the amount of KaiC is constant, as mentioned before. Additionally, a variable called *inst* is included to distinguish both instantiation which may be considered. Afterward we define transitions which describe the general phosphorylation/unphosphorylation reactions and assign them a rate according to the instantiation being considered. Finally, we close the module with the command `endmodule`.

With this, we were able to perform a stochastic simulation. The output of this simulation is shown in Figure 6.4. Even without a precise estimation of the parameters, it is interesting to notice that a cyclic behavior can already be observed since we have several peaks for each component value. This is interesting because it fits what would be expected for a circadian rhythm model.

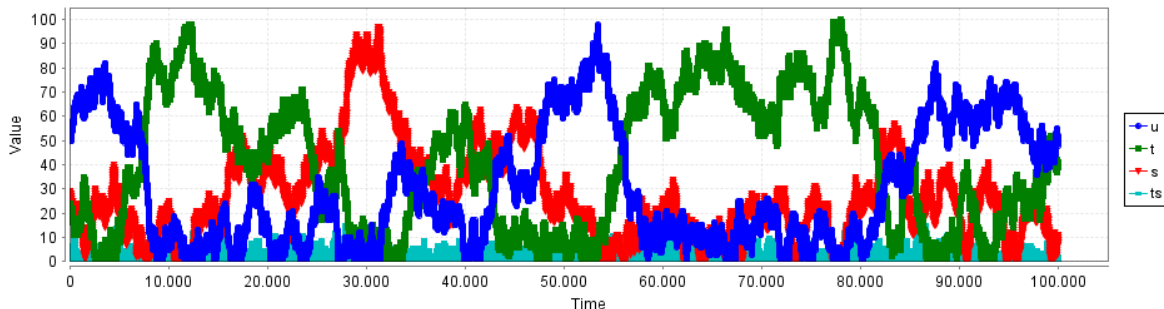


Figure 6.4: Results from a stochastic simulation.

Example 6.2.4. In this example we consider a Lac operon biological system for *E. coli* (see [33] for a more detailed description). This system considers a bacteria of *E. coli* in an environment with two types of sugar – glucose and lactose – and analyses how this bacteria metabolizes them in order to grow.

For short, the bacteria shows preference for glucose and it grows at a great rate while consuming it, consequently increasing the mass of the cell. Also, while the cell is consuming glucose, the lactose permease is shot down, meaning that no lactose is transported into the cell. Moreover, when glucose is fully consumed, the bacteria is able to consume lactose. However, there is some delay in this process, *i.e.* there is a time interval between the growth caused by the consumption of glucose until it runs out, and the cell to grow again due to the consumption of lactose. A cause for this is the lac repressor, *lacI*, which halts the production of enzymes encoded by the lac operon while lactose is not present. However, when lactose enters in the cell, *lacI* starts being inhibited and the cell grows again, at a lower rate than with glucose.

We built a PRISM model for this system using reactive formalisms. Our interest was to replicate the behavior of the bacteria and analyze how would the mass of the cell increase along time. Thus, a stochastic simulation was performed and it was checked if the obtained output matches the results predicted. For this, we consider an *E. coli* bacteria on an environment

with both glucose and lactose and specify its metabolism using a weighted switch graph, which was translated into PRISM language. The output for our model is shown in Figure 6.5.

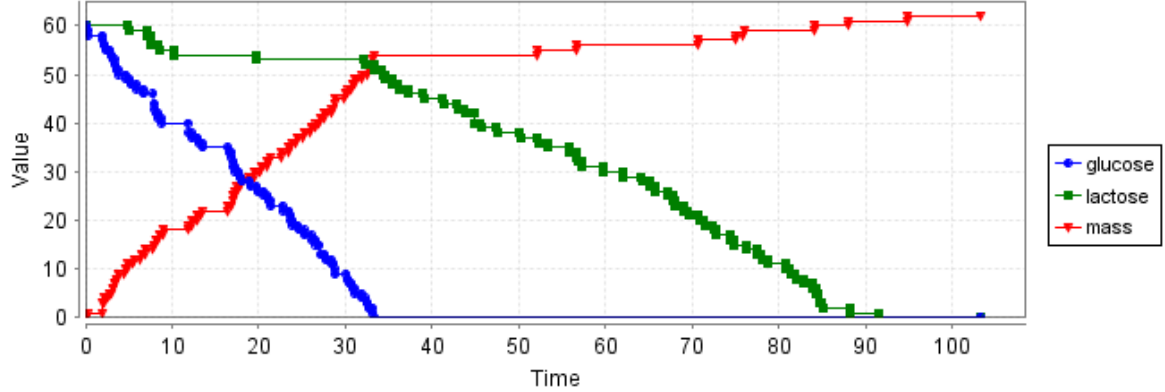


Figure 6.5: PRISM output for the growth of *E. coli* with glucose and lactose.

As expected, the mass of *E. coli* grows quickly while glucose is being consumed and, during this period, lactose is, generally, not consumed. When the bacteria runs out of glucose, the cell starts to consume lactose and its growth stops for a small amount of time before to resume at a slower rate.

The aim of these two examples is not to obtain some new results (as these are well studied models) but to validate and evaluate the consistency of this model and process. Indeed, coherent results were obtained for both cases, using stochastic models based on reactive formalisms such as weighted switch graphs.

6.2.4 Language and examples.

Regarding the last section, we find a way to specify weighted switch graphs in the PRISM language: this was done in Example 6.2.3. However, this process can be long and costly. Note that the inclusion of higher-level edges in a weighted graph exponentially increase the size of the PRISM model. Its inclusion also exponentially increases the size of the code needed because one must, for each instantiation, define the right rates and updates to instantiations. Moreover, this process can be costly because we must treat each instantiation individually. The reason for this is that PRISM is not ready to directly work with weighted and reactive structures. Because of this, we developed an extension to PRISM – which we called rPrism – whose language is specially designed in order to make easier the use of PRISM when working with a weighted switch graph. This tool admits a specific language where one specifies the switch graph along with the initial state and instantiation, and allows the user to run PRISM for that model, after specifying the kind of desired output. It is important to mention that this software is still under development and new features and options are to be integrated. Nevertheless, a Demo version, implemented as a sDL package, is already available². The actual version only supports weighted switch graphs with edges whose level is not higher than 1. This is expected to be extended in future versions.

²http://sdl-vm2.mathdir.org/demos/sDL-pck-run?pck=rPrism/1.0&sdoc=Example_A (lastly accessed in August 29, 2019)

Given a weighted switch graph with edges whose level is not higher than 1, the user of rPrism must specify the model in a simple text format, with the following structure:

```
NS {
N "definition of node 1" {
"definition of edge 1";
"definition of edge 2";
...
}
N "definition of node 2" {
}
...
}

H1 {
"definition of one-level edge 1";
"definition of one-level edge 2";
...
}

options "command1";
output "command2";
sim cmtc;
```

The “definition of a node” has the following format:

“node label” “bound” “ubound” “initvalue”

where “node label” is a valid string with the name/identifier for a vertex of the weighted switch graph; “lbound” (respectively, “ubound”) is an integer determining the lower (respectively, upper) bound with respect to the number of elements of type “node” on the system; and “initvalue” is a integer with the initial value of elements of type “node” on the system.

The “definition of an edge” is done in the following way:

“target node” “initial weight”

where “target node” is a valid string with the name/identifier of the target node; and “initial weight” is a float with the initial weight of the edge.

Finally, to define a one-level edge, we must use the following code:

“source edge” “target edge” “weight”

where both “source edge” and “target edge” are strings and have the format “source node”:“target node”; and “weight” is a float with the weight of the one-level edge.

The entry “command1” determines the output of the program and must be filled according to the goal of the user, for example, can be “simpath 10” (meaning the simulation will make at least 10 steps) or “simtime 5.7” (the simulation will run at least until the time reaches the value 5.7). The entry “command2” can be replaced as “all” in order to obtain the entire set of outputs or restricted to any combination of the commands: “odel”, “simulation_results”, “simulation_plot”, “reachable_sets”, “transition_matrix”, “labels”, separated by a space.

Given a weighted switch graph with no 2-level edges, rPrism translates the model into the PRISM language in order to use it to study reactive models.

Example 6.2.5. Recall the weighted switch graph model presented in Example 6.2.2. The rPrism code to implement it is presented bellow.

```

NS {
N s 0 100 25 {
u 0.3;
}
N ts 0 100 25 {
s 0.4;
}
N t 0 100 25 {
ts 0.4;
}
N u 0 100 25 {
t 0.4;
}
}

H1 {
s:u u:t 0.4;
s:u ts:s 0.4;
s:u t:ts 0.4;
ts:s u:t 0.2;
ts:s t:ts 0.2;
ts:s ts:s 0.6;
}

options simtime 100000;
output all;
sim cmtc;

```

6.3 Final remarks.

When compared with previous chapters, the latter one studies biological regulatory networks on a complete different angle, since it considers a stochastic approach. Nevertheless, the common point is the application of reactive formalisms, which are a particular case of reconfigurable dynamics.

A new kind of stochastic representation, based on weighted switch graphs is proposed for biological regulatory networks and the software PRISM model checker is shown to be able to handle this kind of model. However, since its language is not optimized for these structures, a user-friendly extension to PRISM (called rPRISM) was developed to allow a user to directly specify the structure of a weighted switch graph. This software only accepts weighted switch graphs with no 2-level edges but it is expected to be extended to the entire class of weighted switch graphs soon.

This kind of approach was shown to be suitable to model biological regulatory networks and obtain satisfactory and coherent results. A performance analysis was not formally done, however, we expect this method to be computationally costly, since the number of states grows exponentially along with the number of higher-level edges considered. Nevertheless, note that, in these models, higher-level edges replace some other elements like states and transitions (see Example 6.2.2). Furthermore, in this thesis we considered systems whose dynamics and the expected simulation outputs were already known because we wanted to validate our models and methods. However, we believe that the main applicability of reactive formalisms is to describe phenomena whose cause is still not fully understood. Using weighted switch graphs we are able to describe some known phenomena without including the unknown component responsible for that behavior. This can be seen, for example, in Example 6.2.2 where the component A – representing the KaiA protein – was not considered.

Finally, the choice for PRISM as base software for the study of these models is justified by its capacity of study a probabilistic model in a symbolic way. However, one can note that this is not done in this work since only simulations were performed. Indeed, although PRISM is able to do simulations, it is not optimized for it. However, it can also be used to study several features of a probabilistic model such as steady state probabilities (which can be interesting in biological context). Moreover, PRISM admits a temporal logic to discuss properties of the system. Nevertheless, since these features of PRISM still were not implemented in rPrism, we decided not to present them yet, but expect to integrate them in future versions of rPrism. For now, simulations are used to validate our models and justify the use of weighted reactive switch graphs.

Chapter 7

Conclusion and future research

Which models, methods, tools can synthetic biology inherit from Computer Science? More precisely, what may core concepts in computing – e.g. automata, bisimulation, modalities, reconfigurability, hybrid systems, ... – bring to our understanding of biological or bio-synthetic systems and our ability to both design and analyze them? This thesis contributes to establish a roadmap to address these questions and goes a few steps in it. We have proceeded with a specific focus on the study biological regulatory networks, and took as a starting point the methodology proposed by M. Chaves [10] where a methodology to study a biological regulatory network, relating discrete and continuous models, was proposed.

Several contributions to this roadmap, suitably documented in the previous chapters, were developed along this doctoral research.

Starting with PWL models and ODEs with discrete controllers, we discussed the adoption of hybrid formalisms and tools like KeYmaera and dReach. Actually, biological networks are suitably abstracted as hybrid systems, which on their turn can be regarded as a particular case of the reconfigurable systems. We showed how the syntax of $d\mathcal{L}$ can be used to specify properties of their dynamics. This opened the possibility to use KeYmaera, which is a semi-automatic theorem prover developed in the context of cyber-physical programming, as a verifier in the bio-synthetic domain. Similarly, we have investigated the potentialities of dReach to complement KeYmaera since the former is able to deal with systems of differential equations numerically.

On a second stage, we studied the connection between PWL models and BN, and introduced a new intermediary model – reactive Boolean networks. Some steady states from PWL models are lost when one considers the corresponding simplified BN model. This problem can be partially solved considering generalized BN models based on the concept of switch graphs. We showed that, in general, RBN can preserve an increased number of attractors from a PWL than a BN model. Unfortunately these generalized models are computationally harder than the usual BN models. Therefore, their introduction in this thesis was accompanied by an investment on the corresponding theory, concerning, namely, bisimulation and a plain representation for switch graphs, as well as an adapted semantical interpretation for modal logic. We have also proposed another generalization dealing with the introduction of weights in the underlying switch graphs. This class of generalized models can be applied in diverse contexts, namely of a fuzzy or probabilistic character. In particular, we proposed a variant of RBN which includes probabilities, as a generalization of BN models with probabilities.

We also developed a theoretical analysis to BN models, linking some of their features to

the concept of bisimulation, leading to possibly new application areas for these models. Actually, the relevant literature reports on several alternative representations of these models, e.g. hierarchical, SCC clustering, etc... (see [18]). Bisimulation provides a different characterization of systems and may be used along with a modal logic. In a different biological context, a similar idea was applied in [58]. Finally, the method of asymptotic graph was improved to capture more asymptotic dynamics of the original graph. When comparing to the usual AG method, EAG was shown to be very efficient in detecting and eliminating spurious attractors. Note that the EAG method is also computationally hard, only becoming efficient for biological models containing more than 13 components.

Finally, we developed a reactive and stochastic approach to biological models whose dynamics and components are not completely known. The proposed model is based in weighted switch graphs and considers weights as rates. This work was complemented by studying the application of the PRISM probabilistic model checker to the relevant models. Since PRISM is not able to deal directly with reactive models, a tool called rPrism, specifically designed for reactive models, was developed in this doctoral research. From an engineering point of view, PRISM functionality became available to work with weighted reactive models, through a user-friendly interface.

In a more fundamental perspective, the work presented in this thesis aims at clarifying the relationships between different kinds of models as well as highlighting their interconnections and common points. This was accomplished by presenting an exhaustive description of such models, a theoretical analysis, and by proposing new models and methods for the emerging problem. As mentioned above, for the success of this research contributed the revisit of basic software concepts – namely, bisimulation, reconfiguration, and reactivity. These concepts, which are already broadly applied in computational fields, often have shown appropriated to describe the dynamics of biological systems, and to develop new tools and methods.

Concluding this thesis, it is important to note that some of the results obtained not only solve a specific problems, but also pave the way for many other challenging developments. In particular we propose to pursue along the research lines described in the sequel.

- The idea of considering a PWL model or an ODE model with discrete controllers as an hybrid system leads to the adoption of many other tools and concepts. For instance, some other possible approaches for these models are found in [50, 54]. Nevertheless, we believe that further considerable improvements could be attained with KeYmaera in what concerns PWL models. Note that KeYmaera calls Mathematica to solve systems of differential equations. However, since the differential equations in PWL models are linear, a different (open-source) software could be used. In a distinct direction, the robustness of $d\mathcal{L}$ and KeYmaera can be questioned, since the logic admits no uncertainty in state variables values. Contrarily to what is done in dReach with δ -reachability, $d\mathcal{L}$ cannot deal with errors nor perturbations, which are quite common in real life. This problem could be addressed by considering the development of an intervalar or multi-valued version of $d\mathcal{L}$. In particular, the intervalar version of this logic would take $d\mathcal{L}$ as well as some basic results documented in [53, 62, 61] as starting point and targeting the developed of both a sound proof calculus and an automatic prover.
- Many directions can be followed to continue the work initiated in this thesis on RBN model. A concrete example would be the introduction of edges with level higher than 1 in order to check if more asymptotic behaviors could be recovered from a PWL model

when compared to usual RBN and BN models. In a theoretical perspective, we would like to study and develop more expressive logic languages to capture generalizations of the switch graphs introduced in this thesis. Some possibilities are hybrid [20] but also fuzzy and probabilistic versions. These logics should be able to capture the reactive properties of the system, offer a sound proof calculus, and studied for completeness, decidability and expressive power. Among the several options suggested by our work, we believe that the probabilistic version would be the most demanding and useful because it could be applied to the study of RBN models with probabilities.

- Finally, the idea of using reactivity in stochastic models builds, as far as we know, a completely new approach. The idea is quite promising and may lead to the development of new approaches and results in diverse fields. The further development of rPrism emerges as a main task since PRISM is not optimized for simulation and rPrism does not yet integrate several options available in the mother tool. Our goal is to extend it in future and use the multiple analytical tools of PRISM as well as its temporal logic, directly from rPrism, taking advantage of PRISM being open source.

Bibliography

- [1] Carlos Areces, Raul Fervari, and Guillaume Hoffmann. Swap logic. *Logic Journal of the IGPL*, 22(2):309–332, 2014.
- [2] Carlos Areces, Raul Fervari, and Guillaume Hoffmann. Relation-changing modal operators. *Logic Journal of IGPL*, 23(4):601–627, 2015.
- [3] Michal Baczyński and Balasubramaniam Jayaram. *Fuzzy Implications*, volume 231 of *Studies in Fuzziness and Soft Computing*. Springer, 2008.
- [4] Gleb Beliakov, Ana Pradera, and Tomasa Calvo. *Aggregation Functions: A Guide for Practitioners*, volume 221 of *Studies in Fuzziness and Soft Computing*. Springer, 2007.
- [5] Johan van Benthem. An essay on sabotage and obstruction. In Dieter Hutter and Werner Stephan, editors, *Mechanizing Mathematical Reasoning*, volume 2605 of *LNCS*, pages 268–276. Springer, 2005.
- [6] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic: Graph. Darst.*, volume 53. Cambridge University Press, 2002.
- [7] Patrick Blackburn and Johan van Benthem. 1 Modal logic: a semantic perspective. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, pages 1 – 84. Elsevier, 2007.
- [8] Claudio Fuentes Bravo and Patricio Fuentes Bravo. Molecular logic: Brief introduction and some philosophical considerations. In Madalena Chaves and Manuel A. Martins, editors, *Molecular Logic and Computational Synthetic Biology*, volume 11415 of *LNCS*, pages 1 – 17. Springer, 2019.
- [9] Christos G. Cassandras and Stephane Lafortune. *Introduction to discrete event systems*. Springer, 2009.
- [10] Madalena Chaves. *Predictive analysis of dynamical systems: combining discrete and continuous formalisms*. Habilitation thesis, University of Nice, 2013.
- [11] Madalena Chaves and Alfonso Carta. Attractor computation using interconnected boolean networks: testing growth rate models in E. coli. *Theoretical Computer Science*, 599:47–63, 2015.
- [12] Madalena Chaves, Etienne Farcot, and Jean-Luc Gouzé. Probabilistic approach for predicting periodic orbits in piecewise affine differential models. *Bulletin of mathematical biology*, 75(6):967–987, 2013.

- [13] Madalena Chaves, Daniel Figueiredo, and Manuel A. Martins. Boolean dynamics revisited through feedback interconnections. *Natural Computing*, 2018.
- [14] Madalena Chaves and Miguel Preto. Hierarchy of models: From qualitative to quantitative analysis of circadian rhythms in cyanobacteria. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 23(2):025113, 2013.
- [15] Madalena Chaves and Laurent Tournier. Predicting the asymptotic dynamics of large biological networks by interconnections of boolean modules. In *50th IEEE Conference on Decision and Control and European Control Conference*, pages 3026–3031. IEEE, 2011.
- [16] Kuo-Chen Chou. Low-frequency resonance and cooperativity of hemoglobin. *Trends in Biochemical Sciences*, 14(6):212, 1989.
- [17] Daniel Figueiredo. Differential dynamic logic and applications. Master’s thesis, University of Aveiro, 2015.
- [18] Daniel Figueiredo. Relating bisimulations with attractors in boolean network models. In María Botón-Fernández, Carlos Martín-Vide, Sergio Santander-Jiménez, and Miguel A. Vega-Rodríguez, editors, *Algorithms for Computational Biology*, volume 9702 of *LNBI*, pages 17–25. Springer, 2016.
- [19] Daniel Figueiredo and Luís S. Barbosa. Reactive models for biological regulatory networks. In Madalena Chaves and Manuel A. Martins, editors, *Molecular Logic and Computational Synthetic Biology*, volume 11415 of *LNCS*, pages 74–88. Springer, 2018.
- [20] Daniel Figueiredo, Manuel A Martins, and Luís S Barbosa. A note on reactive transitions and Reo connectors. In Frank de Boer, Marcello Bonsangue, and Jan Rutten, editors, *It’s All About Coordination*, volume 10865 of *LNCS*, pages 57–67. Springer, 2018.
- [21] Daniel Figueiredo, Manuel A. Martins, and Madalena Chaves. Applying differential dynamic logic to reconfigurable biological networks. *Mathematical Biosciences*, 291:10 – 20, 2017.
- [22] Daniel Figueiredo, Eugénio Rocha, Manuel A. Martins, and Madalena Chaves. rPrism – a software for reactive weighted state transition models. In Milan Češka and Nicola Paoletti, editors, *Hybrid Systems and Biology*, volume 11705 of *LNCS*, pages 165–174. Springer, 2019.
- [23] Aleksei F. Filippov. *Differential equations with discontinuous righthand sides: control systems*, volume 18 of *Mathematics and its Applications*. Springer, 1988.
- [24] János C. Fodor and Marc R. Roubens. *Fuzzy Preference Modelling and Multicriteria Decision Support*. Theory and Decision Library D:. Springer, 1994.
- [25] Dov M. Gabbay and Sérgio Marcelino. Modal logics of reactive frames. *Studia Logica*, 93(2):405–446, 2009.
- [26] Dov M. Gabbay and Sérgio Marcelino. Global view on reactivity: switch graphs and their logics. *Annals of Mathematics and Artificial Intelligence*, 66(1-4):131–162, 2012.

- [27] Julien Gagneur and Georg Casari. From molecular networks to qualitative cell behavior. *FEBS letters*, 579(8):1867–1871, 2005.
- [28] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In Maria Paola Bonacina, editor, *Automated Deduction CADE-24*, volume 7898 of *LNCS*, pages 208–214. Springer, 2013.
- [29] Leon Glass and Stuart A. Kauffman. Co-operative components, spatial localization and oscillatory cellular dynamics. *Journal of theoretical biology*, 34(2):219–237, 1972.
- [30] Leon Glass and Stuart A. Kauffman. The logical analysis of continuous, non-linear biochemical control networks. *Journal of theoretical Biology*, 39(1):103–129, 1973.
- [31] Leon Glass and Rafael Pérez. Limit cycle oscillations in compartmental chemical systems. *The Journal of Chemical Physics*, 61(12):5242–5249, 1974.
- [32] Peter J. E. Goss and Jean Peccoud. Quantitative modeling of stochastic systems in molecular biology by using stochastic petri nets. 95(12):6750–6755, 1998.
- [33] Anthony J. F. Griffiths, William M. Gelbart, Jeffrey H. Miller, and Richard C. Lewontin. Regulation of the lactose system. In *Modern Genetic Analysis*. WH Freeman, 1999.
- [34] Katrin P. Guillen, Carla Kurkjian, and Roger G. Harrison. Targeted enzyme prodrug therapy for metastatic prostate cancer—a comparative study of l-methioninase, purine nucleoside phosphorylase, and cytosine deaminase. *Journal of biomedical science*, 21(1):65, 2014.
- [35] Philippe van Ham. How to deal with variables with more than two levels. In Ren Thomas, editor, *Kinetic Logic a Boolean Approach to the Analysis of Complex Regulatory Systems*, volume 29 of *LNBM*, pages 326–343. Springer, 1979.
- [36] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic logic*. MIT press, 2000.
- [37] John Heath, Marta Kwiatkowska, Gethin Norman, David Parker, and Oksana Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 391(3):239–257, 2008.
- [38] Thomas A. Henzinger. The theory of hybrid automata. In M. Kemal Inan and Robert P. Kurshan, editors, *Verification of Digital and Hybrid Systems*, volume 170 of *NATO ASI Series*, pages 265–292. Springer, 2000.
- [39] J. Megginson Hollister, Peter Laing, and Sarnoff A. Mednick. Rhesus incompatibility as a risk factor for schizophrenia in male adults. *Archives of general psychiatry*, 53(1):19–24, 1996.
- [40] Hidde de Jong. Modeling and simulation of genetic regulatory systems: a literature review. *Journal of computational biology*, 9(1):67–103, 2002.
- [41] Maurice Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5):593–599, 1953.

- [42] Erich P. Klement, Radko Mesiar, and Endre Pap. *Triangular Norms*, volume 8 of *Trends in Logic*. Springer, 2000.
- [43] Koichi Kobayashi and Kunihiro Hiraishi. Design of probabilistic boolean networks based on network structure and steady-state probabilities. *IEEE transactions on neural networks and learning systems*, 28(8):1966–1971, 2016.
- [44] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dReach: δ -reachability analysis for hybrid systems. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 9035 of *LNCS*, pages 200–205. Springer, 2015.
- [45] Yanni Kouskoulas, David Renshaw, André Platzer, and Peter Kazanizides. Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 263–272. ACM, 2013.
- [46] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer aided verification*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
- [47] Marta Kwiatkowska, Gethin Norman, David Parker, Oksana Tymchyshyn, John Heath, and Eamonn Gaffney. Simulation and verification for computational modelling of signalling pathways. In Jason Liu Frederick P. Wieland L. Felipe Perrone, Barry Lawson, editor, *Proceedings of the Winter Simulation Conference*, pages 1666–1675. Omnipress, 2006.
- [48] Stéphane Leduc. *La biologie synthétique*, volume 2. A. Poinat, 1912.
- [49] Sarah M. Loos, André Platzer, and Ligia Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In Michael Butler and Wolfram Schulte, editors, *Formal Methods*, volume 6664 of *LNCS*, pages 42–56. Springer, 2011.
- [50] Alexandre Madeira. *Foundations and techniques for software reconfigurability*. PhD thesis, Universities of Aveiro, Minho and Porto, 2013.
- [51] Alexandre Madeira, Renato Neves, Luis S. Barbosa, and Manuel A. Martins. A method for rigorous design of reconfigurable systems. *Science of Computer Programming*, 132:50–76, 2016.
- [52] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.
- [53] Ramon E. Moore. *Interval Arithmetic and Automatic Error Analysis in Digital Computing*. PhD thesis, Stanford University, 1962.
- [54] Renato Neves and Luis S. Barbosa. Hybrid automata as coalgebras. In Augusto Sampaio and Farn Wang, editors, *Theoretical Aspects of Computing - ICTAC 2016*, volume 9965 of *LNCS*, pages 385–402. Springer, 2016.

- [55] André Platzer. *Logical analysis of hybrid systems: proving theorems for complex dynamics*. Springer, 2010.
- [56] André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In Ana Cavalcanti and Dennis Dams, editors, *Formal Methods*, volume 5850 of *LNCS*, pages 547–562. Springer, 2009.
- [57] André Platzer and Jan-David Quesel. European train control system: A case study in formal verification. In Karin Breitman and Ana Cavalcanti, editors, *Formal Methods and Software Engineering*, volume 5885 of *LNCS*, pages 246–265. Springer, 2009.
- [58] Giordano Pola, Maria D. Di Benedetto, and Elena de Santis. Arenas of finite state machines. *arXiv preprint arXiv:1106.0342*, 2011.
- [59] Manuel A. Martins Regivan H. N. Santiago and Daniel Figueredo. Introducing fuzzy reactive graphs. (submitted).
- [60] Steven L. Salzberg. Open questions: How many genes do we have? *BMC biology*, 16(1):94, 2018.
- [61] Regivan H. N. Santiago, Benjamín Bedregal, Alexandre Madeira, and Manuel A. Martins. On interval dynamic logic. In Leila Ribeiro and Thierry Lecomte, editors, *Formal Methods: Foundations and Applications*, volume 10090 of *LNCS*, pages 129–144. Springer, 2016.
- [62] Regivan H. N. Santiago, Benjamín R. C. Bedregal, Alexandre Madeira, and Manuel A. Martins. On interval dynamic logic: Introducing quasi-action lattices. *Science of Computer Programming*, 175:1–16, 2019.
- [63] Joseph R. Shoenfield. *Mathematical logic*. AK Peters/CRC Press, 2018.
- [64] Amilra P. de Silva. *Molecular logic-based computation*. 12. Royal Society of Chemistry, 2012.
- [65] Gautier Stoll, Eric Viara, Emmanuel Barillot, and Laurence Calzone. Continuous time boolean modeling for biological signaling: application of gillespie algorithm. *BMC systems biology*, 6(1):116, 2012.
- [66] Murata Tadao. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, 77(4), 1990.
- [67] Alvin Tamsir, Jeffrey J. Tabor, and Christopher A. Voigt. Robust multicellular computing using genetically encoded nor gates and chemical “wires”. *Nature*, 469(7329):212–215, 2011.
- [68] Laurent Tournier and Madalena Chaves. Interconnection of asynchronous boolean networks, asymptotic and transient dynamics. *Automatica*, 49(4):884–893, 2013.
- [69] José M. G. Vilar, Hao Yuan Kueh, Naama Barkai, and Stanislas Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proceedings of the National Academy of Sciences*, 99(9):5988–5992, 2002.

- [70] Gad Yagil. Quantitative aspects of protein induction. In *Current topics in cellular regulation*, volume 9, pages 183–236. Elsevier, 1975.