



**UNIVERSITÉ
DE LORRAINE**

**BIBLIOTHÈQUES
UNIVERSITAIRES**

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact bibliothèque : ddoc-theses-contact@univ-lorraine.fr
(Cette adresse ne permet pas de contacter les auteurs)

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4

Code de la Propriété Intellectuelle. articles L 335.2- L 335.10

http://www.cfcopies.com/V2/leg/leg_droi.php

<http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm>

Synthesis of Boolean Networks from the Structure and Dynamics of Reaction Networks

THÈSE

présentée et soutenue publiquement le 7 juillet 2023

pour l'obtention du

Doctorat de l'Université de Lorraine
(mention informatique)

par

Athénaïs Vaginay
 0000-0001-5062-7993

<i>Président :</i>	Thierry Bastogne	Professeur, Université de Lorraine
<i>Rapporteurs :</i>	François Fages Loïc Paulevé	Directeur de Recherche, Inria Saclay Chargé de Recherche CNRS, Labri
<i>Examinatrice :</i>	Anna Niarakis	Maître de Conférences HDR, Université d'Evry Val d'Essonne
<i>Encadrants :</i>	Taha Boukhobza Malika Smaïl-Tabbone	Professeur, Université de Lorraine Maître de Conférences HDR, Université de Lorraine



Remerciements

C'est en fait ça, la partie la plus difficile à écrire de la thèse. On veut faire honneur à tous les gens qui ont compté, n'oublier personne, tout en étant un peu original ; c'est un sacré jeu d'équilibriste !

Dans l'introduction du *Traité élémentaire de chimie* (1789), Lavoisier décrit que, lorsqu'on débute dans une discipline, on est comme un petit enfant dont les choix ne sont au début guidés que par les besoins, et non par une réflexion profonde. Un petit enfant... c'est un peu comme ça que je me suis sentie tout au long de mon doctorat, et il y a pas mal de moments où tout me semblait vertigineusement nouveau. Heureusement, le destin a mis sur ma route tout un tas de gens pour me guider. Dans les lignes qui suivent, ce sont toutes ces personnes que je souhaite remercier. Merci de m'avoir aidée, à votre manière, à grandir.

D'abord, un grand merci à mes encadrants Taha Boukhobza et Malika Smaïl-Tabbone, sans qui l'idée même de ce doctorat n'aurait pas été possible. J'ai quitté mes briques rouges lilloises adorées pour l'art nouveau de Nancy, et vous avez su me rassurer pour mes premiers pas dans ce nouveau monde. Merci aussi pour votre confiance (même lorsque mes choix m'éloignaient petit à petit de vos cœurs d'expertise), et pour avoir veillé à ce que je ne m'égare pas trop et n'ouvre pas de parenthèse dans les parenthèses.

Un gros merci aux membres de mon jury de soutenance. À mes deux rapporteurs, Loïc Paulevé et François Fages, à Thierry Bastogne qui a présidé le jury, et à Anna Niarakis qui en a été l'examinatrice. Vous m'avez fait un grand honneur en évaluant mes travaux et merci pour la discussion qui a suivi. Vos avis (parfois contradictoires !) m'ont aidée à mieux appréhender qu'il n'y a en effet pas qu'une seule et unique manière valide de présenter son travail, et qu'il est toujours possible (et nécessaire) de discuter de ces choix en argumentant au mieux.

Un merci aussi à mes co-auteurices de papiers. J'ai énormément apprécié progresser à vos côtés, et je ne serai pas mécontente de continuer à explorer les perspectives de nos travaux. Un autre merci à mes collègues d'enseignement, j'ai hâte de reprendre du service. De manière plus pragmatique, j'adresse un remerciement aux sources de financement de mon doctorat : la fédération Charles Hermite, le CRAN, le Loria et l'Université de Lorraine.

Enfin, merci à celles et ceux qui ont ajouté un peu de Tendresse à ce monde de brutes (vous vous reconnaitrez, mais big up à la famille, aux amiès d'ici et d'ailleurs, à 'tit biscuit ♡, aux grenouilles, aux ours, aux p'tits loups et aux canards qui peuplent cette planète), un peu d'Hérésie (genre les gens avec qui il est possible de se poser tout un tas de questions sur le sens de la vie), une dose d'Énergie (youhou les aficionados des pauses cappuccinos), des notes de muSique (vive la trompette, la guitare et la chorale !), ou plus globalement, pris soin de mon Environnement de travail et de vie (le Loria est vraiment l'endroit rêvé pour bosser, glander, et vivre, tout simplement). Merci à vous, sans qui ces années en THESE n'auraient pas été aussi agréables.

Contents

List of Figures	vii
List of Tables	ix
List of Algorithms	ix

General Introduction	1
----------------------	---

Chapter 1	
Preliminaries	9

1.1	General Concepts and Notations	10
1.1.1	Sets	10
1.1.2	Relations	11
1.1.3	Functions	11
1.1.4	Derivatives	12
1.1.5	Multisets	12
1.1.6	Graphs	12
1.1.7	Computational Complexity	13
1.2	Let There be Logic	14
1.2.1	Propositional Logic	15
1.2.2	First-Order Logic	17
1.3	Abstraction	21
1.4	Arithmetic in First-Order Logic	23
1.4.1	Syntax	23
1.4.2	Semantics	24
1.4.3	Abstraction from $S_{\mathbb{R}}$ to $S_{\mathbb{S}}$	25
1.5	First-Order Ordinary Differential Equations in FOL	27
1.5.1	Syntax	27
1.5.2	Usual Semantics	28

1.6	Boolean functions	29
1.6.1	Representation with Hypercube	31
1.6.2	Representation with DNF Propositional Formulas	31
1.6.3	Minimal Expressions	32
1.6.4	Specific Classes of Boolean Functions	35
1.7	Models for Dynamical Systems	37
1.8	Boolean Networks	39
1.8.1	Syntax	40
1.8.2	Semantics	40
1.8.3	Dynamics: Transition Graphs	42
1.8.4	Structure: Influence Graph	42
1.9	Reaction Networks	44
1.9.1	Syntax	44
1.9.2	Differential Semantics	45

Chapter 2 Formal Modeling of Biological Systems	47
--	-----------

2.1	Introduction	47
2.2	Success Stories in Systems Biology	48
2.3	A Dichotomous Zoo of Model Blueprints	50
2.3.1	Blueprints of Mechanism-Based Models	50
2.3.2	Blueprints of Influence-Based Models	56
2.4	Model Synthesis as a Parameter Fitting Task	58
2.4.1	Input: Available Knowledge and Data	58
2.4.2	Scoring Functions	64
2.4.3	Search Algorithms	65
2.5	Model Analyses and Transformations	67
2.5.1	Structure Analyses	67
2.5.2	Dynamic Analyses	67
2.5.3	Model Simplification	68
2.5.4	Aggregation of Models	71
2.5.5	Conversion Between Models	71
2.6	Wrap-up	73

Chapter 3 Extracting the Structure and Dynamics of a Reaction Network	75
--	-----------

3.1	Introduction	75
-----	--------------	----

3.2	The Structure of a Reaction Network	76
3.3	The Dynamics of a Reaction Network	77
3.3.1	From ODEs to Boolean Configuration Transitions	78
3.3.2	Concrete Numerical Simulation and Binarisation	89
3.3.3	Abstract Simulation	91
3.4	Related Works on the Extraction of a Transition Graph from a Reaction Network	96
3.5	Summary	97
3.5.1	Wrap-up	97
3.5.2	Discussion and Perspectives	98

Chapter 4 Boolean Network Synthesis from Given Structure and Dynamics	103
--	------------

4.1	Introduction	103
4.2	Presentation of ASK&D-BN	105
4.2.1	Step 1: Binarisation of the Given Dynamics	105
4.2.2	Step 2: Synthesis of Transition Functions	106
4.2.3	Step 3: Global Boolean Network Assembly	118
4.2.4	Evaluation of the Synthesised Boolean Networks	118
4.3	Related Works	119
4.3.1	Non-redundant Boolean Networks	120
4.3.2	Fitting the Structure	121
4.3.3	Fitting the Dynamics	121
4.3.4	Partial Good-Enough versus Exhaustive Optimal Synthesis	124
4.3.5	Simplicity	125
4.4	Evaluation of ASK&D-BN	126
4.4.1	Boolean Network Synthesis Methods for the Comparison	126
4.4.2	Datasets	126
4.4.3	Evaluation Procedure	128
4.4.4	Results on the Two Real Datasets	129
4.4.5	Results on the Generated Datasets	129
4.5	Wrap-Up	132

Chapter 5 The SBML2BNET Pipeline and its Evaluation	135
--	------------

5.1	Introduction	135
5.2	Input and Output File Formats	137

5.2.1	Reaction Networks in SBML	137
5.2.2	Boolean Networks in BNET	138
5.3	Implementation of the SBML2BNET Pipeline	138
5.4	Evaluation	141
5.4.1	Results on \mathcal{R}_{enz}	141
5.4.2	Results on SBML Models from BioModels	144
5.5	Wrap-up, Discussion, and Perspectives	149

General Conclusion	153
Bibliography	157
Appendix A SBML file for \mathcal{R}_{enz}	177
Appendix B Analytical Solution of Equation (3.1)	179
Appendix C Extended abstracts (en/fr)	181
Résumé, Abstract	

List of Figures

1	Current consensus of the systems biology cycle.	3
1.1	Truth tables of the logical connectives.	14
1.2	Interpretation of propositional formulas.	17
1.3	Interpretation of first-order terms and formulas.	20
1.4	Homomorphism preservation of the interpretation of a functional relation.	22
1.5	Interpretation of arithmetic terms and formulas.	24
1.6	Rule of signs.	26
1.7	Unit hypercubes from size zero to size three.	31
1.8	Examples of Boolean networks.	41
2.1	Reaction networks of the enzymatic process.	52
2.2	Power law distribution.	63
2.3	How the mutual information relate to the join entropy, individual entropies, and conditional entropies.	64
2.4	Formal relationships between reaction networks and their stochastic, discrete, Boolean, and differential semantics.	72
3.1	Successive configurations of equation (3.1) according to the analytical solution and to the Euler algorithm.	84
3.2	Procedure of the Euler algorithm on one iteration.	85
3.3	Error accumulation after several iterations of the Euler algorithm.	85
3.4	More accurate approximation with the Euler algorithm using smaller Δ .	86
3.5	Illustration of the abstract temporally next relation.	88
3.6	Illustration of the abstract temporally next relation after restriction on the base variables.	88
3.7	Deterministic numerical simulation of \mathcal{R}_{enz} .	91
3.8	Binarised timeseries for \mathcal{R}_{enz} .	91
3.9	An FO-BNN for the reaction network \mathcal{R}_{enz} ($bnn(\mathcal{R}_{\text{enz}})$).	94
3.10	Abstract simulation of \mathcal{R}_{enz} via $bnn(\mathcal{R}_{\text{enz}})$.	94
3.11	Partial abstract simulation of \mathcal{R}_{enz} via $bnn(\mathcal{R}_{\text{enz}})$.	95
3.12	Abstract simulation of \mathcal{R}_{enz} with one threshold (on S).	100
3.13	Abstract simulation of \mathcal{R}_{enz} with two thresholds (on P and S).	101
4.1	Workflow of ASK&D-BN.	106

- 4.2 Coverage evaluation of the Boolean networks synthesised by ASK&D-BN, REVEAL, Best-Fit and Caspo-TS on two real datasets. 130
- 4.3 Coverage evaluation of the Boolean networks synthesised by ASK&D-BN, REVEAL, Best-Fit and Caspo-TS on 42 generated datasets in the ARN setting. 132

- 5.1 Workflow of the SBML2BNET pipeline. 136
- 5.2 Abstract simulation of B448 starting from the abstraction of the initial configuration provided in the SBML model. 140
- 5.3 A Boolean network to model \mathcal{R}_{enz} . 143
- 5.4 Runtime of the Boolean network synthesis step. 147
- 5.5 Coverage evaluation for the Boolean networks synthesised by SBML2BNET for 155 SBML models. 148
- 5.6 Evaluation of the impact on the coverage of constructing the PIG with rules and/or events. 148

- C.1 Current consensus of the systems biology cycle. 181
- C.2 A reaction network and its graphical representation. 182
- C.3 A Boolean network and its influence graph. 182
- C.4 Workflow of the SBML2BNET pipeline. 184
- C.5 Consensus actuel sur le cycle de la recherche en biologie des systèmes. 186
- C.6 Un réseau de réactions et sa représentation graphique. 186
- C.7 Un réseau booléen et son graphe d'influences. 187
- C.8 Déroulé du pipeline SBML2BNET. 188

List of Tables

- 1.1 Summary of the sets used in the thesis. 10
- 1.2 Common complexity classes. 13
- 1.3 Interpretation of arithmetic expressions in $S_{\mathbb{R}}$, $S_{\mathbb{R} \rightarrow \mathbb{R}}$, and $S_{\mathbb{S}}$. 25
- 1.4 Some ODEs and their solutions. 30
- 1.5 Number of Boolean functions of size n (for $n \leq 8$) in various classes. 38

- 4.1 Comparison of ASK&D-BN with REVEAL, Best-Fit and Caspo-TS. 126
- 4.2 Summary of the two real datasets used for the evaluation of ASK&D-BN. 127
- 4.3 Number of experiments for which each method failed to synthesise any Boolean network, number of Boolean networks synthesised over all 336 experiments with generated timeseries and number of Boolean networks returned over the 42 experiments with the ARN setting. 131

- 5.1 Settings for the evaluation of the SBML2BNET pipeline on \mathcal{R}_{enz} . 141
- 5.2 Summary of the Boolean networks synthesised for \mathcal{R}_{enz} , using a concrete numerical simulation and different binarisation procedures to retrieve Boolean transitions. 142
- 5.3 Settings for the evaluation of the SBML2BNET pipeline on SBML models from BioModels. 145
- 5.4 Number of SBML models processed and CPU time of the Boolean network synthesis step. 147
- 5.5 Number of Boolean networks synthesised by the SBML2BNET pipeline. 147
- 5.6 Impact of the setting on the search space and synthesised functions in terms of monotony and constancy. 151

List of Algorithms

- 1 Local synthesis of ASK&D-BN. 107
- 2 Abstract simulation of an FO-BNN from a given abstract initial configuration. 140

General Introduction

Almost never can a complex system of any kind be understood as a simple extrapolation from the properties of its elementary components.

Marr in [Mar82, p. 19]

Life and other processes at its periphery are made possible by intertwined (bio)-chemical transformations that form so-called biological *systems* [Far21; Lav89]. Hillmer gives the following definition of a system [Hil15]:

A system is a collection of parts and factors that work together to complete a task. Conversely, for a given task, the system is defined by the set of all parts and factors which influence, accomplish, or impede that task.

Biological systems are often described as *complex*, because their dynamics are not easily derived from the static knowledge of their constituents in isolation [Spi04]. Moreover, the dynamics is often found to be “robust yet fragile” [Kit04; Alo19]. This means that most of the functions of biological systems are typically maintained despite external and internal perturbations, but it can happen that some small local changes have a strong impact on the complete system [Whi12]. The yeast cell cycle is an example of such a robust yet fragile system [Li+04; LA21].

Depending on the situation, the impact of the perturbations may be considered to be for the better (e.g., more production of a desired compound) or for the worse (e.g., a cell that stops answering to external stimuli and becomes cancerous). In particular, health and disease can be seen respectively as normal and perturbed dynamics of a biological system.

To gain a comprehensive and systematic understanding of life in general (and of disease in particular), it is thus necessary to study the *dynamics* of biological systems. Moreover, the mechanisms driving those dynamics are important, in order to predict useful interventions that make the systems act according to our wishes. These two aspects are the subjects of systems biology [ZBH20; Bre10].

To date, the approaches of systems biology have been applied to study biological systems at different levels and to solve a variety of problems [Kha21]. These levels include those of genes, RNA, proteins, metabolite, cells, tissues, organs, whole individuals, and communities. As for the kind of problems systems biology solves, examples include: deciphering disease mechanisms, classifying patients, finding therapy options and diagnostic, prognostic or theranostic features of biomarkers, and developing efficient drugs [Che+14; Bas+21; Cla+20; CRF20; Tur+21].

As for other fields, the core tool of systems biology are *models* [Laz02; WL05]. In the broadest sense, a model is an abstract representation (abbreviated and convenient) of the

reality (more complex and detailed) [Koh+10]. A model relies on some assumptions. It is *formal* when it is expressed in a given formalism, i.e., a set of rules that specifies the model blueprint rigorously [Gun14a]. It can be used to make predictions one can trust if the model is correct [Gun14b]. As a matter of fact, the correctness of a model is hard to evaluate, but it is estimated through the following points:

1. the validity of its assumptions,
2. how well its predictions fit the existing knowledge and experimental data.

A classic example of a model in biology is the use of ATCG strings to represent the sequence of nucleotide bases that make up DNA: adenine (A), thymine (T), cytosine (C), and guanine (G). This model ignores some aspects of the system under study, such as the cellular context in which the DNA molecule exists, its interaction with the molecules around, and its 3D structure (DNA is a double-stranded helix composed of atoms, themselves composed of smaller entities). However, these omitted elements are irrelevant to answer many questions. Indeed, the sole study of motifs in these DNA string models was successful to identify genes, to predict the function of their products, to find the binding sites of some molecules, and to understand the evolutionary history of different organisms [Dha06].

Various blueprints (modelling formalisms) have been proposed to study biological systems [Nov15; Mac+11]. To the best of my knowledge, they all rely on the traditional *discretisation* philosophy of chemistry [Goo12]. Indeed, while the (chemical) processes that govern the biological systems at a molecular level consist of *continuous* transformations of some (chemical) structures into some others, classic chemistry (and systems biology), traditionally breaks down these continuous processes into a sequence of steps (*reactions*) applied to specific components (*species*), considered as important [Goo12]. As for the study of the exact continuous processes, it is the realm of molecular dynamics.

Several dichotomous classifications of the formal models used in systems biology have been discussed in the literature:

1. static versus dynamical [Dai+10],
2. qualitative versus quantitative [Nov15],
3. mechanism-based versus influence-based [FH07; Hun+08a; Hun+08b; Fag+16; Bak+18; Vod23].

The third dichotomy is the one that matters in this thesis. In the mechanism-based category, a model mimics the underlying mechanisms by using *reactions*. Each reaction is formally described by giving its inputs (*reactants*) and outputs (*products*). The simulation of a mechanism-based model consists in picking a subset of its reactions and then applying the corresponding rewriting to the current configuration of the system. In contrast, a model from the influence-based category usually consists of a set of components, each associated with a transition rule that encodes the relationship of that component to the others. In this category, we are particularly interested in Boolean networks, which associate a Boolean function to each component. A simulation consists in picking which components to update. The next configuration of the system is then computed by updating the value of the components which have been chosen.

The assumptions and predictions of all these models typically concern the structure of the system under study as well as its dynamics [RS02]. The former is about the components involved and how they influence each other, while the latter is about patterns in the dynamics of the system, such as the successive and stable configurations, or how the system reacts to some perturbations. To achieve sufficient correctness, models and wet experiments are typically refined over repeated iterations of hypothesis formulation, modelling, prediction, and experimentation. This is the main idea of the so-called cycle of systems biology. In 2002, Kitano detailed a relatively simple cycle, consisting of hypotheses from which we design and run wet experiments to generate data [Kit02]. A model is then formulated in accordance with these data. In turn, the simulation and analysis of the model produce new data and predictions that must be tested experimentally. Other authors have subsequently refined the cycle, in particular to acknowledge that the “dry” *in silico* experiments can also produce useful data, *in parallel* of “wet” experiments and that the data integration part of the cycle has become more and more central [PK06; AKM09]. The current consensus about this cycle is summarised in figure 1.

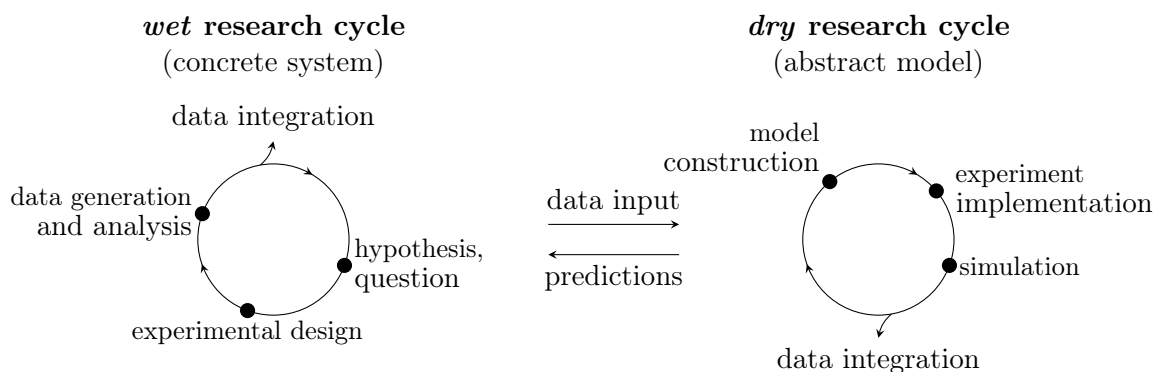


Figure 1: Current consensus of the systems biology cycle adapted from [BS20, Figure 1]. The “wet” and “dry” research cycles consist basically of the same steps, and can provide insights in parallel to one another.

The data integration step concerns the data useful for and produced by the cycle, as well as the models and other artefacts produced from them. This data tends to be large and heterogeneous but paradoxically, they are often insufficient to produce detailed models [SLG06]. As for the models, they are often scattered. To conciliate, organise and manage all this lore is difficult but of great importance. In the last years, lots of efforts have been done to create “ressource-ome”¹ [CMA05; Lis11]. One of the most successful initiatives to store biological models is the repository BioModels, which contains a curated collection of over a thousand published models [Mal+20]. The repository uses different languages to define the content, syntax, and semantics of the models [Nov06]. In particular, the Systems Biology Markup Language (SBML) is the *de facto* standard for encoding biological models [Kea+20]. The success of SBML is demonstrated by both its prevalence in the literature and its use in many programs (for model creation, simulation, analysis, annotation, etc.). In SBML, the encoding of a model consists of a static representation that strives to be formalism-agnostic. Then, tools importing an SBML model can cast it in (virtually) any

¹ The suffix “ome” is principally used in biology, and forms nouns referring to a totality of things. For example, genome, proteome, etc.

formalism.

Each formalism has its own strengths and weaknesses [Nov15]. The choice of which to use is guided by the question at hand: the best one is the simplest one which is sufficient to answer the question [Bak+18; Bor05]. In the literature, most of the models correspond to detailed reaction networks usually studied using the formalisms of ordinary differential equations. However, and while counter-intuitive, it is sometimes interesting to downgrade (abstract) to a “simpler” (coarser, less granular) formalism [DB08a]. In some cases, the granularity level of two different models are nested in such a way that it is possible to draw conclusions about the more granular model by running computations on the less granular model [HGD08; FS08a].

The most important kind of abstraction considered in this thesis pertains to the values of the variables in the models: we will abstract the precise amount of the species and only keep track of whether they are above a given threshold or not. Formally, value abstraction is a process which approximates a computation on values from a set C (called the *concrete domain*) with values from a smaller set A (called the *abstract domain*). This is done by mapping the elements of C to the elements of A . The mapping is designed to preserve some relevant properties. For example, if C is ordered, then the abstraction should preserve the ordering as well. This allows us to reason directly with operations on A while drawing conclusions on C .

A classic example of such an abstraction is the mapping from the reals (concrete domain) to the signs (abstract domain). The “rules of signs” (figure 1.6 on page 26), first described by Brahmagupta in his book *Brāhmasphuṭasiddhānta* (628 c.e.), state that there are cases where we do not need to know the exact value of the arguments to know the sign of the result of an arithmetic operation. We intuitively all know that the sign abstraction is *correct*, in that every solution of the computation done with real values is consistent with the solutions of the computation done with the sign of the real values. However, there are also some cases in which the sign of the result is unknown. For example, we cannot know the exact sign of the sum of a positive and a negative. It is symbolised by using \top in the tables from figure 1.6 on page 26. The sign abstraction is thus *incomplete*.

Computer science abounds with formal methods to study abstraction [Mel13]. Fages et al. used some of these methods to establish formally the relationship of core reaction network models along with their differential (system of first-order differential equations), stochastic (continuous-time Markov chain), discrete (Petri net) and Boolean (Boolean asynchronous transition system) semantics [FS08a]. Such formal connection does not exist for all the kinds of models that are used in systems biology, in particular between reaction networks and Boolean networks. Still, Boolean automata networks are often used in place of reaction network models because they are simpler to reason with. Therefore, this thesis studies the conversion of reaction network models to Boolean network models.

Problem Statement and Contributions

In this thesis, we challenged ourselves to improve the model development cycle. The novelty is in that we do so by up-cycling knowledge and data retrieved from existing models, instead of producing new ones through wet experiments. Indeed, the literature is rich of “ready-to-use” models, especially reaction network models, but we are convinced that converting these models

into the formalism of Boolean networks can help to discover new insights and ease some analyses that are not easy to run on the original models (such as control). Moreover, by avoiding a hand-made conversion process and by using peer-reviewed models (which have undergone rigorous testing and validation), we avoid an error-prone conversion, and we can validate the resulting models against high-quality data, providing confidence in the overall process.

With these ideas in mind, we develop an approach to *automatically* synthesise Boolean networks from *existing* reaction networks. The proposed approach is inspired by what is classically done by the existing model synthesis methods. In particular, it is formulated as a parameter-fitting task, and the information used to fit the Boolean networks pertains the *structure* (list of components and their direct influences on one another) and the *dynamics* (list of behavioural patterns of the components) of the underlying system.

For a given reaction network, we define its structure as a graph which captures the *direct* influences of the components. We can retrieve such a graph by simply parsing the input model. As for the dynamics, we explore two ways of retrieving it. Both ways rely on the simulation of ordinary differential equations (ODEs) retrieved from the reaction network. First, the qualitative dynamics of a reaction network is obtained by *abstract simulation*. For this, we construct a first-order logic formula from the ODEs of the reaction network. We call this formula a *first-order Boolean networks with non-deterministic updates* because its abstract interpretation (where the precise values of the variables are replaced by their sign) defines a transition relation on the Boolean configurations of the system. It can thus be used to build a Boolean transition graph. We show that this graph is a correct overapproximation relatively to the Euler simulation of the ODEs. Second, we use a concrete simulation of the ODEs, followed by a binarisation step. This second approach may seem naive at first glance, but the problem is not that simple as ODEs of biological system exhibit so-called stiff behaviours (mix of slow and rapid evolutions). In particular, simple simulation algorithms, such as the Euler algorithm, do not perform very well, and *clever* simulation algorithms have to be used to balance the trade-off between accuracy and efficiency of the simulation. Moreover, as it is more direct, it allows us to process reaction networks for which the abstract simulation is not yet applicable. In particular, if the reaction network is coupled with discontinuous events, which is the case for real-life models available in databases.

Then, the Boolean network synthesis step consists in finding *all* the Boolean networks *compatible* with a given structure and dynamics. As mentioned earlier, this step is formulated as a parameter-fitting task. More precisely, we see it as a constraint satisfaction problem mixed with optimisation. We encode the problem in the answer-set programming framework (ASP) because it is well suited for this kind of problems. In our framework, a Boolean network is considered to be compatible with a given structure if the influences that play a role in its dynamics are all allowed by the provided structure. As for the dynamics, we investigate cases where it is given as either a complete truth table, or a partial truth table, or a quantitative timeseries which undergoes a binarisation step. A Boolean network is considered to be compatible with a given dynamics if it can somehow reproduce it. The structure constraints are hard, but the dynamics constraints are soft. Indeed, it is not always the case that a Boolean network with a perfect fit exists, in which case we want to synthesise Boolean networks that respect the provided structure and reproduce the dynamics *as well as possible*.

The SBML2BNET pipeline is an implementation of all the methods mentioned above, where the input reaction network is encoded in the Systems Biology Markup Language (SBML) and the synthesised Boolean networks are encoded in the BNET format. This pipeline is intended to be modular, so that several variants can be explored. We apply SBML2BNET on simple reaction networks and more complex ones, retrieved from the BioModels repository, and show that it is always able to retrieve Boolean networks with the best compatibility, according to our criteria. We also discuss the limitations and difficulties faced, in particular when the input reaction network makes use of additional discontinuous events and rules.

In summary, along the document, we attempt to answer the following questions:

- Q1** What are the advantages of a Boolean network, compared to a reaction network? In which cases is a Boolean network preferable to a reaction network?
- Q2** What information can we use to synthesise Boolean networks?
- Q3** How can this information be retrieved from a reaction network?
- Q4** How can this information be used to define the notion of *compatibility* of a Boolean network with a reaction network?
- Q5** How to synthesise automatically such a compatible Boolean network?
- Q6** Does a compatible Boolean network always exist?

Outline

This thesis is organised as follows. **Chapters 3 to 5** are the core contribution chapters of the thesis.

Chapter 1: Preliminaries

This chapter presents general concepts and notations, as well as propositional and first-order logic, which are major formal tools used in the thesis. The former is used to formalise Boolean functions and ultimately Boolean networks. The latter is used to formalise algebra and the sign abstraction on equations (that we prove to be correct but incomplete), as well as systems of ordinary differential equations which are in use in the so-called differential semantics of reaction networks.

Chapter 2: Formal Modeling of Biological Systems

In this chapter, we do a literature review to address **Q1** and **Q2**. We present the field in which this thesis is situated, namely the field of formal modelisation of biological systems. We highlight several blueprints of models proposed to study biological systems, as well as their strength and weaknesses. We also analyse a plethora of existing model construction methods, and highlight their commonalities. In particular, that this task is usually formulated as a parameter-fitting task, and that the usual information used to fit the models pertains the *structure* (list of components and their interaction) and the *dynamics* (list of behavioural patterns of the components) of the underlying system.

Chapter 3: Extracting the Structure and Dynamics of a Reaction Network

This chapter addresses **Q3**. We detail the proposed procedure to retrieve the structure and the dynamics of a given reaction network.

Chapter 4: Boolean Network Synthesis from Given Structure and Dynamics

In this chapter, we address **Q4**, **Q5** and **Q6**: we describe our approach to synthesise Boolean networks compatible with a given structure and dynamics. We compare our approach with three state-of-the-art Boolean network synthesis methods, namely **REVEAL**, **Best-Fit**, and **Caspo-TS**, and we discuss the limitations from which they suffer from in the context of this thesis.

Chapter 5: The SBML2BNET Pipeline and its Evaluation

The implementations of the methods presented in **chapters 3** and **4** are put together in this chapter. Several variants of the resulting pipeline are applied on simple reaction networks and more complex ones, retrieved from the BioModels repository. We show that the translation from a reaction network to a set of Boolean networks is overall quite straightforward, and we discuss the limitations and difficulties faced, in particular when the reaction networks make use of additional features.


General Conclusion

This chapter summarises the results obtained and discusses several perspectives, which mainly concern the application of the SBML2BNET pipeline on bigger and more complex reaction networks.


Associated Papers

I co-authored five papers during my PhD.


The abstract interpretation of the ODEs presented in **chapter 3** is the direct application of a work done in collaboration with Joachim Niehren and Cristian Versari and published in

 J. Niehren et al. “Abstract Simulation of Reaction Networks via Boolean Networks”. In: *Computational Methods in Systems Biology*. Ed. by I. Petre et al. Vol. 13447. Lecture Notes in Computer Science. Springer, 2022, pp. 21–40. DOI: [10.1007/978-3-031-15034-0_2](https://doi.org/10.1007/978-3-031-15034-0_2)


The retrieval of the structure of a reaction network as well as the concrete simulation (**chapter 3**) and the extensive application of the pipeline performed in **chapter 5** is based on the work published in

 A. Vaginay et al. “From Quantitative SBML Models to Boolean Networks”. In: *Complex Networks & Their Applications*. Ed. by R. M. Benito et al. Vol. 1016. Studies in Computational Intelligence. Springer, 2021, pp. 676–687. ISBN: 978-3-030-93412-5 978-3-030-93413-2. DOI: [10.1007/978-3-030-93413-2_56](https://doi.org/10.1007/978-3-030-93413-2_56)


and extended in

 A. Vaginay et al. “From quantitative SBML models to Boolean networks”. In: *Applied Network Science* 7.1 (2022), p. 73. DOI: [10.1007/s41109-022-00505-8](https://doi.org/10.1007/s41109-022-00505-8)

Chapter 4, which focuses on the automatic synthesis of Boolean networks from a given structure and dynamics is mainly based on research published in

 A. Vaginay et al. “Automatic Synthesis of Boolean Networks from Biological Knowledge and Data”. In: *Optimization and Learning*. Ed. by B. Dorronsoro et al. Communications in Computer and Information Science. Cham: Springer International Publishing, 2021, pp. 156–170. ISBN: 978-3-030-85672-4. DOI: [10.1007/978-3-030-85672-4_12](https://doi.org/10.1007/978-3-030-85672-4_12)

Finally, part of the general reflexions about formal modelling of biological systems (especially in [chapter 2](#)) was shaped by a project done in collaboration with biologist colleagues from CRAN. We were given expression data and used machine learning to understand the impact of the expression of the membrane G-protein-coupled estrogen receptor (GPER) on the survival of female glioblastoma patients. This work has led to the following publication:

 A. Hirtz et al. “GPER Agonist G-1 Disrupts Tubulin Dynamics and Potentiates Temozolomide to Impair Glioblastoma Cell Proliferation”. In: *Cells* 10.12 (2021), p. 3438. ISSN: 2073-4409. DOI: [10.3390/cells10123438](https://doi.org/10.3390/cells10123438)

Chapter 1

Preliminaries

Contents

1.1	General Concepts and Notations	10
1.1.1	Sets	10
1.1.2	Relations	11
1.1.3	Functions	11
1.1.4	Derivatives	12
1.1.5	Multisets	12
1.1.6	Graphs	12
1.1.7	Computational Complexity	13
1.2	Let There be Logic	14
1.2.1	Propositional Logic	15
1.2.2	First-Order Logic	17
1.3	Abstraction	21
1.4	Arithmetic in First-Order Logic	23
1.4.1	Syntax	23
1.4.2	Semantics	24
1.4.3	Abstraction from $S_{\mathbb{R}}$ to $S_{\mathbb{S}}$	25
1.5	First-Order Ordinary Differential Equations in FOL	27
1.5.1	Syntax	27
1.5.2	Usual Semantics	28
1.6	Boolean functions	29
1.6.1	Representation with Hypercube	31
1.6.2	Representation with DNF Propositional Formulas	31
1.6.3	Minimal Expressions	32
1.6.4	Specific Classes of Boolean Functions	35
1.7	Models for Dynamical Systems	37
1.8	Boolean Networks	39
1.8.1	Syntax	40
1.8.2	Semantics	40

1.8.3	Dynamics: Transition Graphs	42
1.8.4	Structure: Influence Graph	42
1.9	Reaction Networks	44
1.9.1	Syntax	44
1.9.2	Differential Semantics	45

This chapter is mostly intended for readers with little experience with concepts and terminology classically used in mathematics.

Outline In [section 1.1](#), we introduce general concepts and notations. Then in [section 1.2](#), we focus on logic, as it is the foundation of our definitions for Boolean networks and reaction networks. Ordinary differential equations and Boolean functions are presented in [sections 1.5](#) and [1.6](#) respectively. Dynamical systems are presented in [section 1.7](#), along with Boolean networks and reaction networks in [sections 1.8](#) and [1.9](#) respectively.

1.1 General Concepts and Notations

1.1.1 Sets

Definition 1.1.1 (Set). *A set is a collection of distinct elements whose order does not matter. Sets are denoted using the delimiters $\{\dots\}$. Given a finite set S , $|S|$ denotes its cardinality (i.e., the number of elements in the set). If an element e belongs to a set S , we write $e \in S$. If all elements of a set S are also elements of a set S' , we say that S is a subset of S' and write $S \subseteq S'$.*

In this thesis, some sets are of special interest. They are summarised in [table 1.1](#). For the sets \mathbb{B} and \mathbb{S} , the values used might depend on the context.

Table 1.1: Summary of the sets used in the thesis.

Name	Symbol	Values
Booleans	\mathbb{B}	$\{\text{False}, \text{True}\}$ or $\{0, 1\}$ or $\{-1, 1\}$
Signs	\mathbb{S}	$\{-1, 0, 1\}$ or $\{\searrow, \rightarrow, \nearrow\}$
Naturals	\mathbb{N}	$\{0, 1, 2, \dots\}$
Integers	\mathbb{Z}	$\{\dots, -1, 0, 1, \dots\}$
Positive Reals	\mathbb{R}_+	$\{0, \dots, 0.4567\dots, 13.12, \dots\}$
Reals	\mathbb{R}	$\{\dots, -5.67383, \dots, 0, \dots, 123.45678, \dots\}$

Definition 1.1.2 (Powerset). *The powerset of a set S is the set of all $2^{|S|}$ possible subsets of S , i.e.,*

$$2^S = \{R \mid R \subseteq S\}.$$

Example 1.1.3. *The powerset of $\{a, b\}$ is $\{\emptyset, \{a\}, \{b\}, \{a, b\}\}$.*

Definition 1.1.4 (Cartesian Product). *The Cartesian product of two sets S and T is the set of all tuples where the first element is from S and the second element is from T :*

$$S \times T = \{(s, t) \mid s \in S, t \in T\}.$$

The Cartesian product of three or more sets is defined by the obvious extension.

Example 1.1.5. The Cartesian product of $\{a, b\}$ and $\{1, 2, 3\}$ computes the set $\{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3)\}$.

1.1.2 Relations

Definition 1.1.6 (Relation). An n -ary relation r is a subset of the Cartesian product of n sets. Its domain, denoted $\text{dom}(r)$, is the corresponding Cartesian product. The number n of arguments of a relation r is called the arity. A relation r of arity n is denoted $r^{(n)}$.

A relation r can be defined by giving a list of ordered tuples of values, or by a generative expression (formula).

Example 1.1.7. Let $S = \{0, 2, 4\}$, $T = \{1, 3, 5\}$ and l and e two relations meaning respectively “less than” and “equals”. We have that $r = \{(0, 1), (0, 3), (0, 5), (2, 3), (2, 5), (4, 5)\}$ and $e = \emptyset$.

A binary relation is a relation of arity two. When dealing with a binary relation r , we will use three interchangeable notations to say that $s \in S$ is in relation with $t \in T$ in r : either $r(s, t)$ or $(s, t) \in r$ or $s \, r \, t$. The first notation will be particularly useful later when dealing with predicates in logic, the second will be particularly used when dealing with abstraction and the third one will serve as a shortcut when the relations will be classic operators.

1.1.3 Functions

Definition 1.1.8 (Function). A function $f \subseteq I_1 \times \dots \times I_n \times O$, also denoted $f : I_1 \times \dots \times I_n \rightarrow O$, is a special kind of relation which assigns at most one element $f(x) \in O$ to every $x \in I_1 \times \dots \times I_n$. Adopting the terminology of programming, we refer to x and $f(x)$ as input and output, respectively. For a given function f taking n inputs, the notations $f(x) = y$ and $(x, y) \in f$ are equivalent if we consider the corresponding relation of arity $n + 1$. The domain of a function $f : A \rightarrow B$ is the set A and it is denoted $\text{dom}(f)$. A function f is said to be partial if not all the elements of $\text{dom}(f)$ have an output, and total otherwise. The restriction of f to a subset $A' \subseteq \text{dom}(f)$ is written as $f|_{A'}$. The composition $f \circ g$ of two functions f and g is obtained by first applying g and then applying f .

The mapping that defines a function can be given in different ways. For example, we use the notation $\{(a_1, b_1), \dots, (a_n, b_n)\}$ for the finite function f with finite domain $\text{dom}(f) = \{a_1, \dots, a_n\}$ and $f(a_i) = b_i$ for all $1 \leq i \leq n$.

The mapping can also be represented using a *function table*, which is a complete list of all the points in the domain of f along the value of f at each point. When the function takes two inputs, the function table can be given as a 2-entries table, such that what is done in [figure 1.6](#) on page 26. One can also use *function graphs* which are particularly suitable representations for functions with one input. The input values are put in abscissa, and the output values in ordinate. Finally, a more compact way to define a function is to use a formula.

In some sections of this thesis, it will be of prime importance to distinguish the function itself (the mapping) from the generative expression (formula) that defines it.

Example 1.1.9. $f = \{(0, 1), (1, 2), (2, 3), (3, 0)\}$ defines a total function from A to B , with $A = B = \{0, 1, 2, 3\}$. Its table and graph are given below:

input	output
0	1
1	2
2	3
3	0

By contrast, $g : \mathbb{R} \rightarrow \mathbb{R}, g(x) = 2x^2 + 13$ is an example of a function defined by a formula.

1.1.4 Derivatives

Under some conditions (of continuity, smoothness, \dots), we can define the rate of change of a function f with respect to changes in its inputs. It is called the derivative.

Example 1.1.10. The derivative of $f(x) = 3x$ is $\frac{df}{dx} = 3$ (later denoted $\dot{f} = 3$ for compactness), intuitively because $f(x)$ grows three times faster than x .

If f has several arguments, it can be partially derivated with respect to each of its arguments. The derivative of $f(a, b)$ relatively to a is denoted $\frac{\partial f}{\partial a}$.

1.1.5 Multisets

A multiset is similar to a set, except in that it can contain the same element several times. We model a multiset as a function that counts the number of occurrences of each element. The following definitions are adapted from [Bli88].

Definition 1.1.11 (Multiset). A multiset m is a function from a domain A to $\mathbb{N} \setminus \{0\}$. $m(a)$ is called the multiplicity of a in m . By convention, we note $a \notin m$ if $a \notin A$, and we consider that $m(a) = 0$.

Note that in this thesis, we will only consider finite multiset, that is, with finite domains.

Definition 1.1.12 (Multiset union). The union of two multisets m_1 and m_2 , denoted $m_1 \cup m_2$ is the set resulting from the union of their domain

$$S = \{x \mid x \in m_1 \text{ or } x \in m_2\}.$$

Definition 1.1.13 (Multiset sum). The sum of two multiset m_1 and m_2 , denoted $m_1 \uplus m_2$ is the multiset

$$m(x) = m_1(x) + m_2(x) \quad \forall x \in m_1, x \in m_2.$$

1.1.6 Graphs

Complete introduction to graph theory can be found for example in [Wil09]. In this thesis, we will use the following concepts.

Definition 1.1.14 (Graph). A graph is a tuple $\mathcal{G} = (V, E)$ where V is an arbitrary set and $E \subseteq V \times V$ is a binary relation. V is called the set of vertices and E the set of edges. If E is symmetric, then \mathcal{G} is undirected; otherwise, it is directed. A function $E \rightarrow \{+, -, \pm\}$ is a signature over \mathcal{G} . By abuse of notation, we will denote $X \in \mathcal{G}$ and $(X, Y) \in \mathcal{G}$ if the node X in V and if the edge $(X, Y) \in E$.

Definition 1.1.15 (Spanning subgraph). Given two graphs $\mathcal{G} = (V, E)$ and $\mathcal{G}' = (V', E')$, \mathcal{G}' is a spanning subgraph of \mathcal{G} if $V' = V$ and $E' \subseteq E$.

Definition 1.1.16 (Bipartite graph). A bipartite graph is a tuple $\mathcal{G} = (V_1 \cup V_2, E)$ where V_1 and V_2 are arbitrary sets such as $V_1 \cap V_2 = \emptyset$ and $E \subseteq V_1 \times V_2 \cup V_2 \times V_1$ is a binary relation.

1.1.7 Computational Complexity

We recall some basics about computational complexity [Pap94; AB09]. A computational problem is a question on a given input, that can be answered by an algorithm. Examples of such problems are decision problems (yes-no questions, such as “is the number n prime?”), search problems (“is there a prime numbers in \mathbb{N} ?”), counting problems (“how many prime numbers are there in \mathbb{N} ”), optimisation problems (“what is the best prime number, given a set of constraints”). The complexity of a problem can be measured by looking at how much time and space is needed to solve the problem computationally. In this context, time and space are abstract, and do not represent physical time and space; they thus have no unit. Several classes of complexity have been defined in the literature. Common classes are P , NP , $PSPACE$. They are summarised in [table 1.2](#).

Table 1.2: Common complexity classes.

Class	Contains the problems solvable in. . .	Example
P	polynomial time.	Find the shortest paths in a graph
NP	polynomial time with non-deterministic choices.	The Boolean Satisfiability Problem (SAT)
$PSPACE$	polynomial space.	Determining properties of regular expressions and context-sensitive grammars

Although unproven, it is commonly assumed that these classes are hierarchised and indicate the “difficulty” of solving a problem. For example, it is generally assumed that problems from the class P are “simpler to solve” compared to problems from the class NP , for example.

For a given complexity class X , a problem P is *co- X* if checking a given counter-example for the P is in X , *X -complete* if P is among the hardest problem of the class, *X -hard* if P is at least as hard as the hardest problems in X (yet not necessarily in X), and A^B if P can be solved with complexity A assuming an oracle (black-box) can solve problems of class B in one instruction. For example, the SAT problem is *NP-complete*. An *NP-hard* problem might not be decidable, for example, the halting problem.

1.2 Let There be Logic

Logic is of preponderant importance in this thesis. Indeed, we will use propositional logic and first-order logic to define formally what a Boolean network and a system of differential equations are, respectively. Logic is also at the basics of the programming paradigms used to solve some central problems in the thesis core. In particular, the Boolean network synthesis problem is solved using a logic programming approach, namely Answer-Set Programming.

In the following, we recall some key concepts about logics. Exhaustive introductions to propositional and first-order logics can be found in the book edited by the Open Logic Project and in [CL03a; CL03b] (in French).

In any logic, there are two key attributes: the *syntax* and the *semantics*. The syntax of a logic consists of a set of rules that specifies which finite sequences of symbols are well-formed expressions, while the semantics determines the actual meanings behind these expressions.

Once syntax and semantics have been defined, it makes sense to *evaluate* a given formula, i.e., to ask whether it evaluates to **True** or **False** for a given assignment of its constituents. Another key problem is to determine what are the *solutions* of the formula, i.e., what are the possible assignments that make the formula evaluate to **True**. Recall from [table 1.1](#) that we can encode **False** and **True** using 0 and 1 respectively (from the set \mathbb{B}).

Logics rely on a set of *logical connectives*. In the thesis, the set of connectives consists of the unary connective \neg (which stands for “not”), and the binary connectives $\wedge, \vee, \Rightarrow, \Leftrightarrow$ (which respectively stand for “and”, “or”, “implication” and “equivalence”). Each of these symbols denotes a Boolean function which takes some inputs and returns a Boolean value, according to the truth tables given in [figure 1.1](#). We can see that the conjunction of two elements of \mathbb{B} is equal to their product as integers. We can push the analogy and omit the operator \wedge , when it does not interfere with readability. For example, with $A, B \in \mathbb{B}$, we can write AB instead of $A \wedge B$.

input P	output $\neg_{\mathbb{B}} P$	input P Q	output $P \wedge_{\mathbb{B}} Q$	output $P \vee_{\mathbb{B}} Q$	output $P \Leftrightarrow_{\mathbb{B}} Q$	output $P \Rightarrow_{\mathbb{B}} Q$
0	1	0 0	0	0	1	1
1	0	0 1	0	1	0	1
		1 0	0	1	0	0
		1 1	1	1	1	1

Figure 1.1: Truth tables of the logical connectives.

In this thesis, we will use a simplified Backus-Naur Form (BNF) notation² to describe the syntaxes. This notation was first introduced in the ALGOL 60 report [Bac+60]. It consists of a set of rules that describe how a symbol can be expanded in terms of other symbols. A rule is typically written in the form:

$$\text{symbol} ::= \text{expansion1} \mid \text{expansion2}$$

where $::=$ means “is defined as”, \mid indicates a choice between several possible expansions, which

² In the literature, BNF is sometimes expanded to Backus Normal Form instead, but that is not recommended as it is not a proper *normal* form [Knu64].

are sequences of symbols that might themselves be recursively defined with a BNF.

1.2.1 Propositional Logic

1.2.1.1 Syntax

The set Φ of well-formed formulas in propositional logic is defined by induction, given a set \mathcal{V} of propositional variables. A propositional variable $p \in \mathcal{V}$ is a well-formed formula. Now, if φ is a well-formed formula, then so is its negation $\neg\varphi$. Let φ and φ' be two well-formed formulas and \odot one of the binary logical connective defined earlier (such as \wedge for example), then $\varphi \odot \varphi'$ is also a well-formed formula. Using BNF notation:

$$\begin{aligned} \varphi, \varphi' \in \Phi(\mathcal{V}) ::= & 0 \mid 1 \mid p \mid \neg(\varphi) \\ & \mid (\varphi \wedge \varphi') \mid (\varphi \vee \varphi') \mid (\varphi \Rightarrow \varphi') \mid (\varphi \Leftrightarrow \varphi') \end{aligned}$$

When parenthesis are omitted, we apply the following priorities: negations over conjunctions over disjunctions over equivalence over implication.

Example 1.2.1. If $\mathcal{V} = \{A, B\}$, the following are formulas:

$$\begin{aligned} & A \\ & \neg A \\ & A \wedge \neg B \end{aligned}$$

However, “ C ” and “ $\sim A$ ” are not formulas because they use symbols that are not allowed by the syntax.

Some expressions respecting particular rules have specific names. In particular, below, we define what is a literal, an elementary conjunction, a non-redundant elementary conjunction, a DNF and a non-redundant DNF.

Definition 1.2.2 (Literal). A literal is an expression of the form x or $\neg x$, where x is a propositional variable or a constant.

$$\text{literal} ::= 0 \mid 1 \mid p \mid \neg p \quad \text{where } p \in \mathcal{V}$$

Definition 1.2.3 (Elementary conjunction). An elementary conjunction —also called cube in reference to the hypercube representation (see [section 1.6.1](#)) or product in reference to the analogy with polynomials— is a conjunction of literals.

$$\text{elemconj} ::= \text{literal} \mid \text{literal} \wedge \text{elemconj}$$

Definition 1.2.4 (Non-redundant conjunction). An elementary conjunction is non-redundant if the variables appear at most once.

Example 1.2.5. The conjunction A is non-redundant, while the conjunction $A \wedge A$ is redundant.

Remark 1.2.6. The set of non-redundant conjunction one can construct from a finite set of variables is finite.

Definition 1.2.7 (Local adjacency). *Two conjunctions are locally-adjacent if they contain the same variables and differ in more than one literal.*

Example 1.2.8. *The conjunctions $A \wedge B$ and $A \wedge \neg B$ are locally-adjacent.*

Definition 1.2.9 (Subsumption). *Given two conjunctions $c = \ell_1 \wedge \dots \wedge \ell_m$ and $c' = \ell'_1 \wedge \dots \wedge \ell'_n$, we say that c subsume c' if for all i such that $1 \leq i \leq m$, there exists j such that $1 \leq j \leq n$ such that $\ell_i = \ell'_j$.*

Example 1.2.10. *The conjunction $A \wedge \neg B \wedge C$ is subsumed by the conjunction $A \wedge \neg B$.*

Definition 1.2.11 (Expression in Disjunctive Normal Form (DNF)). *A logical formula is said to be in Disjunctive Normal Form (DNF) if it is a disjunction of one or more elementary conjunctions. Later in this thesis, such formulas will be of particular interest. The BNF of a DNF expression is:*

$$DNF ::= \text{elemconj} \mid (\text{elemconj}) \vee DNF$$

Definition 1.2.12 (Non-redundant DNF expression). *A DNF is non-redundant if all its conjunctions are non-redundant and appear at most once.*

Remark 1.2.13. *The set of non-redundant DNF expression one can construct from a finite set of variables is finite.*

In the remaining of the thesis, the DNF are by default assumed to be non-redundant, except if stated otherwise.

Example 1.2.14. *The following propositional expressions are in disjunctive normal form: “ $(A \wedge \neg B) \vee (\neg A \wedge B)$ ” and “ $A \wedge \neg B \vee \neg A \wedge B$ ”, but not “ $(A \vee B) \wedge \neg A$ ”. The expression “ $(A \wedge A) \vee (A \wedge A)$ ” is also in DNF, but it is redundant.*

1.2.1.2 Semantics

In propositional logic, an *interpretation* simply consists in an arbitrary assignment of a truth value $\{1, 0\}$ to each propositional variable $p \in \mathcal{V}$, i.e., a function $\iota : \mathcal{V} \rightarrow \mathbb{B}$. Hence, for a formula which consists of n propositional variables there are 2^n possible interpretations.

1.2.1.3 Formula Evaluation

Let φ be a formula and ι an interpretation, both over the same set of variables \mathcal{V} . Since all the variables in φ have an assigned truth value, we can compute the truth value of φ , denoted $\llbracket \varphi \rrbracket_\iota$. To do so, we substitute each variable in φ by its value given by ι , and recursively apply the truth tables of the logical connectives (figure 1.1 on page 14). When evaluating a formula without parentheses, we apply the following priorities: negation over conjunction over disjunction over equivalence over implication. The complete process is summarised in figure 1.2. We write $\iota \models \varphi$ if φ evaluates to 1 when using the truth assignment of the interpretation ι . In this case, we say that ι is a *model* of φ , and that φ is *satisfied* by ι .

$$\begin{aligned}
\llbracket 0 \rrbracket_\iota &= 0 \\
\llbracket 1 \rrbracket_\iota &= 1 \\
\llbracket p \rrbracket_\iota &= \iota(p) & p \in \mathcal{V} \\
\llbracket \neg \varphi \rrbracket_\iota &= \neg_{\mathbb{B}}(\llbracket \varphi \rrbracket_\iota) \\
\llbracket \varphi \wedge \varphi' \rrbracket_\iota &= \llbracket \varphi \rrbracket_\iota \wedge_{\mathbb{B}} \llbracket \varphi' \rrbracket_\iota \\
\llbracket \varphi \vee \varphi' \rrbracket_\iota &= \llbracket \varphi \rrbracket_\iota \vee_{\mathbb{B}} \llbracket \varphi' \rrbracket_\iota \\
\llbracket \varphi \Rightarrow \varphi' \rrbracket_\iota &= \llbracket \varphi \rrbracket_\iota \Rightarrow_{\mathbb{B}} \llbracket \varphi' \rrbracket_\iota \\
\llbracket \varphi \Leftrightarrow \varphi' \rrbracket_\iota &= \llbracket \varphi \rrbracket_\iota \Leftrightarrow_{\mathbb{B}} \llbracket \varphi' \rrbracket_\iota
\end{aligned}$$

Figure 1.2: Interpretation of a propositional formula φ with respect to the variable assignment $\iota : \mathcal{V} \rightarrow \mathbb{B}$.

1.2.1.4 Solution of a formula

The set of solutions $\text{sol}(\varphi)$ is the set of assignments of the propositional variables of φ such that φ evaluates to 1. Formally: $\text{sol}(\varphi) = \{\iota : \mathcal{V} \rightarrow \mathbb{B} \mid \llbracket \varphi \rrbracket_\iota = 1\}$.

1.2.2 First-Order Logic

The expressiveness of propositional logic introduced above is quite limited. It can express statements such as “*this* rat is cute” or “*this* bike is blue” but not “*all* rats are cute” or “*there is* a blue bike”. First-order logic is in a sense an extension of propositional logic: it provides a richer language that allows the use of variables whose domains can be other than \mathbb{B} , of relations (called predicates), and of quantification, thanks to which we can say things about the valuation of the variables for which a formula is evaluates to 1.

1.2.2.1 Syntax

First-order logic adds, on top of the set of variables \mathcal{V} and the of logical connectives $\{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$, the universal \forall (“for all”) and existential \exists (“exists”) quantifiers, as well as disjoint sets of constant symbols \mathcal{C} , functions symbols $\mathcal{F}^{(k)}$ of k inputs, for all $k \geq 0$, and relations (called predicates) symbols $\mathcal{P}^{(k)}$ of arity k for all $k \geq 0$. Formally, a first-order signature is a tuple

$$\Sigma = \left(\mathcal{C}, \bigcup_{k \geq 0} \mathcal{F}^{(k)}, \bigcup_{k \geq 0} \mathcal{P}^{(k)} \right).$$

From a given first-order signature, we can build two types of expressions: *terms* and *formulas*. A term denotes an object in the domain of interest, while a formula makes a statement (which can be true or false) about the relationship between some terms. Terms are thus the basic building blocks of first-order logic formulas.

The set $\mathcal{T}_\Sigma(\mathcal{V})$ of terms is defined recursively from variable, constants, and function symbols (but not from predicates!): every constant symbol $c \in \mathcal{C}$ and variable symbol $v \in \mathcal{V}$ is a term. Moreover, from k terms t_1, \dots, t_k , and a function symbol $f \in \mathcal{F}^{(k)}$, a larger term $f(t_1, \dots, t_k)$ can be built. In BNF:

$$t \in \mathcal{T}_\Sigma(\mathcal{V}) ::= c \quad | \quad v \quad | \quad f(t_1, \dots, t_k) \quad \text{with } c \in \mathcal{C}, v \in \mathcal{V}, f \in \mathcal{F}^{(k)}, t_1 \dots t_k \in \mathcal{T}_\Sigma(\mathcal{V})$$

The set $\Phi_\Sigma(\mathcal{V})$ of first-order formulas with signature Σ and over the variables \mathcal{V} is defined by induction. An atomic formula $p(t_1, \dots, t_k)$ can be built from k terms t_1, \dots, t_k and a k -ary predicate symbol p . Then, if φ, φ' are formulas, \odot a binary logical connective, and x a variable, $\neg\varphi$ and $\varphi \odot \varphi'$ are also formulas, as well as $\forall x \varphi$ (“for all valuations of x , φ evaluates to 1”) and $\exists x \varphi$ (“there exists valuations of x such that φ evaluates to 1”). In BNF:

$$\begin{aligned} \varphi, \varphi' \in \Phi(\mathcal{V}) ::= & p(t_1 \dots t_k) \quad \text{where } p \in \mathcal{P}^{(k)}, t_1 \dots t_k \in \mathcal{T}_\Sigma(\mathcal{V}) \\ & | \quad \neg\varphi \quad | \quad \varphi \odot \varphi' \\ & | \quad \forall x \varphi \quad \text{where } x \notin \mathcal{V}, \varphi \in \Phi(\mathcal{V} \cup \{x\}) \\ & | \quad \exists x \varphi \quad \text{where } x \notin \mathcal{V}, \varphi \in \Phi(\mathcal{V} \cup \{x\}) \end{aligned}$$

Definition 1.2.15 (Free and bound variables). *In a formula, a variable that is not bound to a quantifier is called free, and bound in the other case. In our setting, a variable cannot appear both free and bound, as a new identifier is declared every time we use the connectives \exists and \forall . The set of free variables in a formula φ is denoted $fv(\varphi)$.*

Example 1.2.16. *In the formula $\forall X \exists Y p(X, Y, Z)$, the variables X and Y are bounded while Z is free.*

Definition 1.2.17 (Ground formula). *A formula is ground if it is variable-free.*

1.2.2.2 Semantics

In this thesis, the interpretation of a first-order formula $\varphi \in \Phi_\Sigma(\mathcal{V})$ requires two objects: a *relational structure* S over signature Σ , and an *assignment function*, giving values to the variables in \mathcal{V} .

A relational structure S is a tuple consisting of:

- a set D , called the interpretation domain, also denoted $dom(S)$,
- the interpretation of each of the constant symbol (i.e., for each constant symbol $c_\Sigma \in \mathcal{C}_\Sigma$, a value in D),
- the interpretation of each of the function symbols (i.e., for each function symbol of k inputs $f_\Sigma \in \mathcal{F}_\Sigma$, a $(k+1)$ -ary relation $f_S : \underbrace{D \times \dots \times D}_{k+1}$),
- the interpretation of each of the predicate symbols (i.e., for each k -ary predicate symbol $p_\Sigma \in \mathcal{P}_\Sigma$, so a k -ary relation $p_S \subseteq \underbrace{2^D \times \dots \times 2^D}_k$). Remark that the interpretation of a predicate can be viewed as the set of ordered tuples that make the predicate evaluate to 1.

To indicate that a structure is for a particular signature Σ , one can refer to it as a Σ -structure.

Remark 1.2.18. *Relational structures are typically used in database theory. They differ from the classic structures used to interpret first-order formulas in that they associate relations to the function symbols, instead of functions. In this thesis, the use of relational structures is justified in that they are well attuned for an operation we will need later, namely the abstraction from arithmetics on reals to arithmetics on signs (section 1.4.3). Moreover, any relational structure with domain D corresponds to a classic structure whose domain is the powerset 2^D of D .*

Note that this definition of a structure does not specify which values are assigned to the free variables in \mathcal{V} . Indeed, the value of the variables are given by a dedicated assignment function $\alpha : \mathcal{V} \rightarrow \text{dom}(S)$. Doing so is a common practice in logics, so one can define a structure and use it with different variable assignments. Without this convention, the variable assignment has to be directly given by the structure, thus a new structure has to be defined each time we change the variable assignment.

1.2.2.3 Formula Evaluation

Given a Σ -structure S and a variable assignment $\alpha : \mathcal{V} \rightarrow \text{dom}(S)$, a first-order formula φ over signature Σ can then be interpreted i.e., evaluated to either 0 or 1. Give a Σ -structure S , a first-order formula φ over signature Σ is satisfiable if it exists a variable assignment function α such that $\llbracket \varphi \rrbracket_{\alpha, S} = 1$. We denote this with $\alpha \models_S \varphi$. If φ is not satisfiable, it is called unsatisfiable. **Figure 1.3** summarises how to compute the truth value $\llbracket \varphi \rrbracket_{\alpha, S} \in \mathbb{B}$ of φ . We first define the value $\llbracket t \rrbracket_{\alpha, S}$ of each term t as subset of elements of the interpretation domain $D = \text{dom}(S)$ as follows:

- for a constant symbol $c_\Sigma \in \mathcal{C}_\Sigma$, $\llbracket c_\Sigma \rrbracket_{\alpha, S} = \{c_S\}$;
- for a variable symbol $v \in \mathcal{V}$, $\llbracket v \rrbracket_{\alpha, S} = \{\alpha(v)\}$;
- for a term $f_\Sigma(t_1, \dots, t_k)$, where f_Σ is a function symbol with k inputs and t_1, \dots, t_k are terms, $\llbracket f_\Sigma(t_1, \dots, t_k) \rrbracket_{\alpha, S} = \{s \mid s_1 \in \llbracket t_1 \rrbracket_{\alpha, S}, \dots, s_k \in \llbracket t_k \rrbracket_{\alpha, S} \text{ and } (s_1, \dots, s_k, s) \in f_S\}$. Intuitively, this set consists of all the possible outputs that each combinations of the values in the set of interpretation of each terms would return.

Now, for formulas, it works as follows. The evaluation of a formula consisting of a predicate $p_\Sigma(t_1, \dots, t_n)$ consists in checking if the given arguments (some terms, which evaluate as subsets of D) are defined by the relation p_S . If it is the case, the formula evaluates to 1. A universally quantified formula evaluates to 1 iff the unquantified formula evaluates to 1 with the quantified variable instantiated to each of the domain elements. An existentially quantified formula evaluates to 1 iff the unquantified formula evaluates to 1 with the quantified variable instantiated to at least one of the domain elements. Based on this, we make two remarks:

1. **Only the assignments to the free variables matter when evaluating a formula φ .** Indeed, for a bound variable, the value given by an assignment function is overwritten by the quantifier anyway. For example, the formula $\neg P(y) \wedge \forall x P(x)$ can never be satisfiable, whatever the assignment function. Indeed, any value assigned to x will be overwritten by the $\forall x$, and ultimately, x will take the same value than the one assigned to the variable y , so the formula will evaluate to 0.

Interpretation of first-order terms:

$$\llbracket c_\Sigma \rrbracket_{\alpha, S} = \{c_S\} \subseteq \text{dom}(S)$$

$$\llbracket v \rrbracket_{\alpha, S} = \{\alpha(v)\} \subseteq \text{dom}(S)$$

$$\llbracket f_\Sigma(t_1, \dots, t_k) \rrbracket_{\alpha, S} = \{s \mid s_1 \in \llbracket t_1 \rrbracket_{\alpha, S}, \dots, s_k \in \llbracket t_k \rrbracket_{\alpha, S}, (s_1, \dots, s_k, s) \in f_S\} \subseteq \text{dom}(S)$$

Interpretation of first-order formulas:

$$\llbracket p_\Sigma(t_1, \dots, t_n) \rrbracket_{\alpha, S} = \begin{cases} 1 & \text{if } (\llbracket t_1 \rrbracket_{\alpha, S}, \dots, \llbracket t_n \rrbracket_{\alpha, S}) \in p_S \\ 0 & \text{otherwise} \end{cases} \in \mathbb{B}$$

$$\llbracket \neg \varphi \rrbracket_{\alpha, S} = \neg_{\mathbb{B}} \llbracket \varphi \rrbracket_{\alpha, S} \in \mathbb{B}$$

$$\llbracket \varphi \wedge \varphi' \rrbracket_{\alpha, S} = \llbracket \varphi \rrbracket_{\alpha, S} \wedge_{\mathbb{B}} \llbracket \varphi' \rrbracket_{\alpha, S} \in \mathbb{B}$$

$$\llbracket \varphi \vee \varphi' \rrbracket_{\alpha, S} = \llbracket \varphi \rrbracket_{\alpha, S} \vee_{\mathbb{B}} \llbracket \varphi' \rrbracket_{\alpha, S} \in \mathbb{B}$$

$$\llbracket \varphi \Rightarrow \varphi' \rrbracket_{\alpha, S} = \llbracket \varphi \rrbracket_{\alpha, S} \Rightarrow_{\mathbb{B}} \llbracket \varphi' \rrbracket_{\alpha, S} \in \mathbb{B}$$

$$\llbracket \varphi \Leftrightarrow \varphi' \rrbracket_{\alpha, S} = \llbracket \varphi \rrbracket_{\alpha, S} \Leftrightarrow_{\mathbb{B}} \llbracket \varphi' \rrbracket_{\alpha, S} \in \mathbb{B}$$

$$\llbracket \forall v. \varphi \rrbracket_{\alpha, S} = \begin{cases} 1 & \text{if } \llbracket \varphi \rrbracket_{\alpha[v/s], S} = 1 \text{ for all } s \in \text{dom}(S) \\ 0 & \text{otherwise} \end{cases} \in \mathbb{B}$$

$$\llbracket \exists v. \varphi \rrbracket_{\alpha, S} = \begin{cases} 1 & \text{if } \llbracket \varphi \rrbracket_{\alpha[v/s], S} = 1 \text{ for some } s \in \text{dom}(S) \\ 0 & \text{otherwise} \end{cases} \in \mathbb{B}$$

Figure 1.3: Interpretation of first-order terms and formulas over a Σ -structure S and with respect to a variable assignment $\alpha : \mathcal{V} \rightarrow \text{dom}(S)$. $c_\Sigma, f_\Sigma, p_\Sigma$ denote respectively a constant, a function and a predicate symbols from Σ , v denotes a variable from \mathcal{V} , t and t' denote terms, and $\alpha[v/s]$ denotes the substitution of v by the value s .

2. **First-order logic is decidable in finite domains.** Indeed, when the interpretation domain is finite, the interpretation process is finite and we will ultimately derive a truth value for a formula. However, in the case where the domain of interpretation is infinite, it is not always possible to determine that a universally quantified formula evaluates to 1, nor that an existentially quantified formula evaluates to 0. In this thesis, we will only compute the truth values of formulas using finite interpretation domain.

1.2.2.4 Solution of a formula

Given a structure S , the solution set $sol^S(\varphi)$ is the set of assignments to the *free* variables of φ such that φ evaluates to 1. As mentioned before, the truth value of a formula φ only depends on the values of its free variable. Formally:

$$sol^S(\varphi) = \{\alpha|_{fv(\varphi)} \mid \alpha : \mathcal{V} \rightarrow dom(S), \llbracket \varphi \rrbracket_{\alpha, S} = 1\}.$$

1.3 Abstraction

Several Σ -structures can be used to interpret a given Σ -formula. Sometimes, two Σ -structures correspond to different granularities, and the solutions they lead to are nested in such a way that it is possible to draw conclusions about the solutions obtained with the more granular structure, by only reasoning on the solutions obtained with the less granular structure. This kind of reasoning is the matter of *formal abstraction*.

To define formally what an abstraction between two relational structures is, we need specific functions called *homomorphisms*. Such a function relates two Σ -structures to each other, while preserving the interpretation of the constants, the relations and the functions, as well as the order of the domains, if any.

Definition 1.3.1 (Homomorphism of relational structures). *A homomorphism between two Σ -structures S and T is a function $h : dom(S) \rightarrow dom(T)$ such that³:*

- *It respects the interpretation of constant symbols. If c_Σ is a constant symbol from Σ and ρ an arbitrary value from $dom(S)$ then :*

$$\rho \in c_S \implies h(\rho) \in c_T.$$

This is in fact equivalent to $\{h(\rho)\} = \llbracket c \rrbracket_T$ since there is only one element in c_S .

- *It respects the interpretation of function symbols. If f_Σ is an n -ary function symbol from Σ and $\rho_1 \dots \rho_{n+1}$ are arbitrary values from $dom(S)$, then:*

$$(\rho_1, \dots, \rho_{n+1}) \in f_S \implies (h(\rho_1), \dots, h(\rho_{n+1})) \in f_T.$$

- *It respects the interpretation of predicate symbols. If p_Σ is a predicate symbol from Σ and*

³ Technically, it is the definition of a *weak* homomorphism otherwise it would have been \iff instead of \implies .

P_1, \dots, P_n arbitrary sets from $2^{\text{dom}(S)}$:

$$(P_1, \dots, P_n) \in p_S \implies (\{h(\{\rho \mid \rho \in P_1\}), \dots, \{h(\rho) \mid \rho \in P_n\}\} \in p_T.$$

Figure 1.4 illustrates what a homomorphism h from structure S to structure T looks like, given a function f and two arbitrary values ρ_1 and $\rho_2 \in \text{dom}(S)$. Note that we assume here that the interpretation of f is a functional relation (i.e., its result is a set with one element).

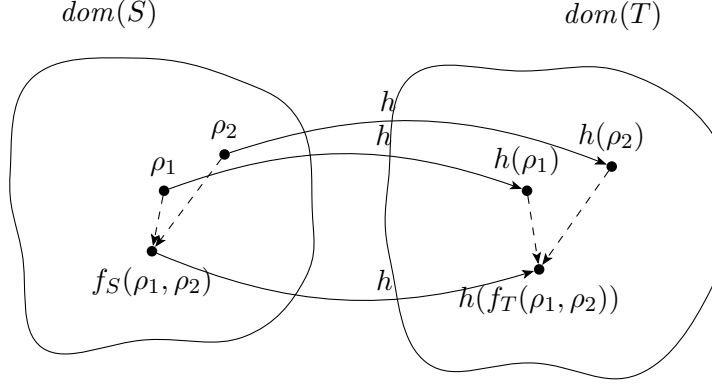


Figure 1.4: Homomorphism h from structure S to structure T , given a function f and two arbitrary values $\rho_1, \rho_2 \in \text{dom}(S)$. The homomorphism preserves the interpretation of f .

Depending on the two structures that are related, a homomorphism can in fact be an abstraction.

Definition 1.3.2 (Structure abstraction). *Given two relational structures \mathcal{C} and \mathcal{A} , a homomorphism $h : \text{dom}(\mathcal{C}) \rightarrow \text{dom}(\mathcal{A})$ is an abstraction from \mathcal{C} to \mathcal{A} if $|\text{dom}(\mathcal{C})| > |\text{dom}(\mathcal{A})|$. \mathcal{C} is called the concrete structure (and its domain $\text{dom}(\mathcal{C})$ is the concrete domain) while \mathcal{A} is called the abstract structure (and its domain $\text{dom}(\mathcal{A})$ is the abstract domain).*

As already mentioned in the general introduction of the thesis, an abstraction preserves some information but might throw away some other. This loss of information occurs because a homomorphism is not necessarily reversible. Formally, we have that h is a function mapping elements of $\text{dom}(\mathcal{C})$ to elements of $\text{dom}(\mathcal{A})$. Let h' be the function that does the reverse: mapping elements of $\text{dom}(\mathcal{A})$ to elements of $\text{dom}(\mathcal{C})$. A term t_Σ interpreted with the concrete relational structure \mathcal{C} returns a subset of $\text{dom}(\mathcal{C})$:

$$C = \llbracket t_\Sigma \rrbracket_{\mathcal{C}} \subseteq \text{dom}(\mathcal{C})$$

applying h to C returns a set $A \subseteq \text{dom}(\mathcal{A})$:

$$A = \{h(x) \mid x \in C\}$$

and applying h' to A returns a set C' of concrete values in $\text{dom}(\mathcal{C})$

$$C' = h'(A).$$

From there, we can distinguish three cases:

Case 1: sound and precise, if $C = C'$.

Case 2: sound but not precise, if $C \subset C'$.

Case 3: not sound, if $C \cap C' = \emptyset$.

The same principles apply when reasoning on the solutions of first-order formulas. The following theorem characterises the solutions of sound yet imprecise abstractions, such as the sign abstraction that is formalised in the next section and that we will use intensively in [chapter 3](#) to characterise the dynamics of a reaction network.

Theorem 1.3.3 (Generalised John’s theorem [All21]). *For any homomorphism h from a relational structure S to a relational structure T (both on signature Σ) and for all existential and positive first-order formula φ (on signature Σ as well) (no \forall nor \neg in φ), the set of solutions of φ in the abstract structure contains at least all the abstraction of the concrete solutions:*

$$h \circ \text{sol}^S(\varphi) \subseteq \text{sol}^T(\varphi).$$

John’s theorem was primarily introduced in [JNN13; Nie+16] to approximate the solutions of first-order formulas with specific structures and homomorphism. Its generalisation to any structure and any homomorphism (including abstractions) can be found in the thesis of Emilie Allart [All21]. Similar theorems with slightly different formulations can be found in the field of model theory, such as the homomorphism preservation theorem [EF95; Ros08].

1.4 Arithmetic in First-Order Logic

In this section, we use first-order formulas to encode systems of arithmetic equations. We also present three different structures with which we can interpret the formulas. These structures are fundamental as we will build upon them to do abstract interpretation of ordinary differential equations system in [chapter 3](#).

1.4.1 Syntax

We consider the following signature:

$$\Sigma_{arith} = \left(\mathbb{R}, \Sigma_{arith}^{(2)}, \overset{\circ}{=} \right)$$

with the set \mathbb{R} of constant symbols (from the real numbers), the set \mathcal{V} of variables symbols, and the set of binary operators symbols $\Sigma_{arith}^{(2)} = \{+, \times, -, /, \text{pow}\}$.

The set $\mathcal{E}_{arith}(\mathcal{V})$ of terms is built from constant $\rho \in \mathbb{R}$, variables $x \in \mathcal{V}$, and binary operators $\odot \in \Sigma_{arith}^{(2)}$:

$$e, e' \in \mathcal{E}_{arith}(\mathcal{V}) ::= \rho \in \mathbb{R} \quad | \quad x \in \mathcal{V} \quad | \quad \odot ee'.$$

Recall that the notations $\odot ee'$ and $e \odot e'$ are equivalent. For the operators in $\Sigma_{arith}^{(2)}$, we will use the infix notation.

Example 1.4.1. *The following expressions are arithmetic terms:*

$$\begin{aligned} 2 \times x \\ 4y + \text{pow}(4, 2) \end{aligned}$$

but not $2\% \setminus 8$ as it does not respect the given syntax.

The set $\Phi_{\Sigma_{arith}}(\mathcal{V})$ of first-order arithmetic formulas is constructed from equations between arithmetic terms $e, e' \in \mathcal{E}_{\Sigma}(\mathcal{V})$, and a subset of the usual first-order connectives:

$$\varphi \in \Phi_{\Sigma_{arith}}(\mathcal{V}) ::= e \stackrel{\circ}{=} e' \mid \exists x. \varphi \mid \varphi \wedge \varphi \mid \neg \varphi \quad \text{where } x \in \mathcal{V}.$$

We sometimes use shorthands $e \geq 0$ for the formula $\exists x. e \stackrel{\circ}{=} x \times x$ and $e \leq e'$ for $e' - e \geq 0$.

1.4.2 Semantics

A formula $e \stackrel{\circ}{=} e'$ corresponds to an *arithmetic equation* between two arithmetic terms $e, e' \in \mathcal{E}_{arith}(\mathcal{V})$. A conjunction of such formulas corresponds to a *system of arithmetic equations*. For any Σ_{arith} -structure S with domain $\text{dom}(S)$, variable assignment $\alpha : \mathcal{V} \rightarrow \text{dom}(S)$, [figure 1.5](#) summarises the interpretation of arithmetics terms $\llbracket t \rrbracket_{S, \alpha} \subseteq \text{dom}(S)$ and formulas $\llbracket \varphi \rrbracket_{S, \alpha} \in \mathbb{B}$. In particular, a formula $t \stackrel{\circ}{=} t'$ is true if the intersection of the possible values for t and the possible values for t' is non-empty, that is, if $\llbracket t \rrbracket_{\alpha, S} \cap \llbracket t' \rrbracket_{\alpha, S} \neq \emptyset$.

Interpretation of arithmetic terms:

$$\begin{aligned} \llbracket c_S \rrbracket_{\alpha, S} &= \{c_S\} && \subseteq \text{dom}(S) \\ \llbracket v \rrbracket_{\alpha, S} &= \{\alpha(v)\} && \subseteq \text{dom}(S) \\ \llbracket t + t' \rrbracket_{\alpha, S} &= \{r \mid s \in \llbracket t \rrbracket_{\alpha, S}, s' \in \llbracket t' \rrbracket_{\alpha, S}, (s, s', r) \in +_S\} && \subseteq \text{dom}(S) \\ \llbracket t \times t' \rrbracket_{\alpha, S} &= \{r \mid s \in \llbracket t \rrbracket_{\alpha, S}, s' \in \llbracket t' \rrbracket_{\alpha, S}, (s, s', r) \in \times_S\} && \subseteq \text{dom}(S) \\ \llbracket t - t' \rrbracket_{\alpha, S} &= \{r \mid s \in \llbracket t \rrbracket_{\alpha, S}, s' \in \llbracket t' \rrbracket_{\alpha, S}, (s, s', r) \in -_S\} && \subseteq \text{dom}(S) \\ \llbracket t/t' \rrbracket_{\alpha, S} &= \{r \mid s \in \llbracket t \rrbracket_{\alpha, S}, s' \in \llbracket t' \rrbracket_{\alpha, S}, (s, s', r) \in /_S\} && \subseteq \text{dom}(S) \\ \llbracket \text{pow}(t, t') \rrbracket_{\alpha, S} &= \{r \mid s \in \llbracket t \rrbracket_{\alpha, S}, s' \in \llbracket t' \rrbracket_{\alpha, S}, (s, s', r) \in \text{pow}_S\} && \subseteq \text{dom}(S) \end{aligned}$$

Interpretation of arithmetic formulas:

$$\begin{aligned} \llbracket t \stackrel{\circ}{=} t' \rrbracket_{\alpha, S} &= \begin{cases} 1 & \text{if } \llbracket t \rrbracket_{\alpha, S} \cap \llbracket t' \rrbracket_{\alpha, S} \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \in \mathbb{B} \\ \llbracket \varphi \wedge \varphi' \rrbracket_{\alpha, S} &= \llbracket \varphi \rrbracket_{\alpha, S} \wedge_{\mathbb{B}} \llbracket \varphi' \rrbracket_{\alpha, S} \in \mathbb{B} \end{aligned}$$

Figure 1.5: Interpretation of arithmetic terms and formulas with respect to the structure S and the variable assignment $\alpha : \mathcal{V} \rightarrow \text{dom}(S)$. Moreover, $v \in \mathcal{V}$.

We now present three different structures that can be used to interpret the expressions. Later on, we will build on them to interpret not only arithmetic expressions but differential equations as well. These structures are called $S_{\mathbb{R}}$, $S_{\mathbb{R} \rightarrow \mathbb{R}}$, and $S_{\mathbb{S}}$, according to their domain. These structures

correspond to algebra on reals, real-valued functions and signs, respectively. They are summarised in [table 1.3](#).

Table 1.3: Interpretation of arithmetic expressions in $S_{\mathbb{R}}$, $S_{\mathbb{R} \rightarrow \mathbb{R}}$, and $S_{\mathbb{S}}$.

Structure	Domain	Variable Assignments
$S_{\mathbb{R}}$	\mathbb{R} (reals)	$\alpha : \mathcal{V} \rightarrow \mathbb{R}$
$S_{\mathbb{R} \rightarrow \mathbb{R}}$	$\mathbb{R} \rightarrow \mathbb{R}$ (real-valued functions)	$\beta : \mathcal{V} \rightarrow (\mathbb{R} \rightarrow \mathbb{R})$
$S_{\mathbb{S}}$	\mathbb{S} (signs)	$\gamma : \mathcal{V} \rightarrow \mathbb{S}$

Structure $S_{\mathbb{R}}$, for an interpretation on reals Here, arithmetic expressions are interpreted in the domain of reals \mathbb{R} . A binary operator $\odot \in \Sigma_{arith}^{(2)} = \{+, \times, -, /, \text{pow}\}$ is interpreted as the classical binary (partial) functions $\odot_{S_{\mathbb{R}}}$ for the addition, multiplication, subtraction, division, and exponentiation of real numbers, respectively. Note that $/_{S_{\mathbb{R}}}$ is not a total function, since division by zero is not defined. Therefore, $\llbracket x/0 \rrbracket_{S_{\mathbb{R}}, \alpha} = \emptyset$ for any $\alpha : \mathcal{V} \rightarrow \mathbb{R}$. With this structure, the sets involved contain a unique element. The \doteq operator can thus be understood as the usual operator for equality on classic arithmetics, by considering the unique value we have on the left and on the right. In other words, we have $(P_1, P_2) \doteq_{S_{\mathbb{R}}}$ only if $P_1 = P_2 = \{x\}$ for some $x \in \mathbb{R}$.

Example 1.4.2. The term $\llbracket 6 + 4/x - (2 \times 0.5) \rrbracket = \{9\}$ if x is assigned 1, but \emptyset if x is assigned 0. The formula $\llbracket 6 \doteq 6 \rrbracket$ evaluates to 1, but $\llbracket 6 \doteq 10 \rrbracket$ evaluates to 0.

Structure $S_{\mathbb{R} \rightarrow \mathbb{R}}$, interpretation on real-valued functions Here, the arithmetic expressions are interpreted in the domain of real-valued functions $\mathbb{R} \rightarrow \mathbb{R}$. The binary operator $\odot_{S_{\mathbb{R} \rightarrow \mathbb{R}}}$ now denotes the partial functions for the addition, multiplication, subtraction, division, and exponentiation of real-valued functions, respectively. Note that a constant $\rho \in \mathbb{R}$ is now interpreted as a constant function: $\rho_{S_{\mathbb{R} \rightarrow \mathbb{R}}}(x) = \rho$ for all $x \in \mathbb{R}$. This time, the \doteq operator is understood as the point-wise equality of functions.

Structure $S_{\mathbb{S}}$, for an interpretation on the signs Here, the arithmetic expressions are interpreted in the domain of signs $\mathbb{S} = \{\nearrow, \rightarrow, \searrow\}$. To do so, we forget about the exact value of the terms and only focus on their signs. The evaluation is done non-deterministically to the set of all possible signs, according to the relational interpretation of the function symbols $+, \times, -, /, \text{pow}$ given in [figure 1.6](#).

Example 1.4.3. The term $\llbracket 3 + (-2) \rrbracket_{S_{\mathbb{S}}} = \{\nearrow\} +_{S_{\mathbb{S}}} \{\searrow\} = \{\nearrow, \searrow, \searrow\} = \mathbb{S}$ since we cannot determine the sign precisely. And indeed, $(\nearrow, \searrow, s) \in +_{S_{\mathbb{S}}}$ for all three signs $s \in \mathbb{S}$.

1.4.3 Abstraction from $S_{\mathbb{R}}$ to $S_{\mathbb{S}}$

Above, we defined $S_{\mathbb{R}}$ and $S_{\mathbb{S}}$ for the interpretation of arithmetic expressions with reals and signs, respectively. Intuitively, we know that arithmetics on signs corresponds to a sound but imprecise abstraction of arithmetics on reals (see Case 2 [section 1.3](#) on page 23).

Example 1.4.4. Consider the following formula: $3 + (-2) \doteq 1$. Its interpretation on the

		input 2				
		$+S_{\mathbb{S}}$	\nearrow	\rightarrow	\searrow	\top
input 1	\nearrow	\nearrow	\nearrow	\nearrow	\top	\top
	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\top	\top
	\searrow	\top	\searrow	\searrow	\top	\top
	\top	\top	\top	\top	\top	\top

$\times S_{\mathbb{S}}$	\nearrow	\rightarrow	\searrow	\top
\nearrow	\nearrow	\rightarrow	\searrow	\top
\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow
\searrow	\searrow	\rightarrow	\nearrow	\top
\top	\top	\rightarrow	\top	\top

$-S_{\mathbb{S}}$	\nearrow	\rightarrow	\searrow	\top
\nearrow	\top	\nearrow	\nearrow	\top
\rightarrow	\searrow	\rightarrow	\nearrow	\top
\searrow	\searrow	\searrow	\top	\top
\top	\top	\top	\top	\top

$/S_{\mathbb{S}}$	\nearrow	\rightarrow	\searrow	\top
\nearrow	\nearrow	\emptyset	\searrow	\top
\rightarrow	\rightarrow	\emptyset	\rightarrow	\rightarrow
\searrow	\searrow	\emptyset	\nearrow	\top
\top	\top	\emptyset	\top	\top

$\text{pow}_{S_{\mathbb{S}}}$	\nearrow	\rightarrow	\searrow	\top
\nearrow	\nearrow	\nearrow	\nearrow	\nearrow
\rightarrow	\rightarrow	\rightarrow	\nearrow	\rightarrow
\searrow	\searrow	\searrow	\nearrow	\searrow
\top	\top	\top	\nearrow	\top

Figure 1.6: Rule of signs. The symbols \nearrow , \searrow and \rightarrow denote respectively positive real numbers, negative real numbers, and 0. The operation returns the set $\top = \{\nearrow, \searrow, \rightarrow\}$ when nothing is known about the result.

structure $S_{\mathbb{R}}$ gives:

$$\begin{aligned}
 \llbracket 3 + (-2) \rrbracket_{S_{\mathbb{R}}} &\stackrel{\circ}{=} \llbracket 3 + (-2) \rrbracket_{S_{\mathbb{R}}} \stackrel{\circ}{=}_{S_{\mathbb{R}}} \llbracket 1 \rrbracket_{S_{\mathbb{R}}} \\
 &= \llbracket 3 \rrbracket_{S_{\mathbb{R}}} +_{S_{\mathbb{R}}} \llbracket -2 \rrbracket_{S_{\mathbb{R}}} \stackrel{\circ}{=}_{S_{\mathbb{R}}} \llbracket 1 \rrbracket_{S_{\mathbb{R}}} \\
 &= \{3\} +_{S_{\mathbb{R}}} \{-2\} \stackrel{\circ}{=}_{S_{\mathbb{R}}} \{1\} \\
 &= \{1\} \stackrel{\circ}{=}_{S_{\mathbb{R}}} \{1\} \\
 &= 1
 \end{aligned}$$

Its interpretation on the structure of signs $S_{\mathbb{S}}$ gives:

$$\begin{aligned}
 \llbracket 3 + (-2) \rrbracket_{S_{\mathbb{S}}} &\stackrel{\circ}{=} \llbracket 3 + (-2) \rrbracket_{S_{\mathbb{S}}} \stackrel{\circ}{=}_{S_{\mathbb{S}}} \llbracket 1 \rrbracket_{S_{\mathbb{S}}} \\
 &= \{\nearrow\} +_{S_{\mathbb{S}}} \{\searrow\} \stackrel{\circ}{=}_{S_{\mathbb{S}}} \{\nearrow\} \\
 &= \{\nearrow, \searrow, \rightarrow\} \stackrel{\circ}{=}_{S_{\mathbb{S}}} \{\nearrow\} \\
 &= 1
 \end{aligned}$$

This is equivalent of applying the function h that abstract values to their signs:

$$\{h(x_1) | x_1 \in \llbracket 3 \rrbracket_{S_{\mathbb{R}}}\} +_{S_{\mathbb{S}}} \{h(x_2) | x_2 \in \llbracket -2 \rrbracket_{S_{\mathbb{R}}}\} \stackrel{\circ}{=}_{S_{\mathbb{S}}} \{h(x_3) | x_3 \in \llbracket 1 \rrbracket_{S_{\mathbb{R}}}\}$$

But since $\llbracket c \rrbracket_{S_{\mathbb{R}}} = \{c\}$, we have that

$$\begin{aligned} \{h(3)\} +_{S_{\mathbb{S}}} \{h(-2)\} &\stackrel{\circ}{=}_{S_{\mathbb{S}}} \{h(1)\} = \{\nearrow\} +_{S_{\mathbb{S}}} \{\searrow\} \stackrel{\circ}{=}_{S_{\mathbb{S}}} \{\nearrow\} \\ &= \{\nearrow, \searrow, \rightarrow\} \stackrel{\circ}{=}_{S_{\mathbb{S}}} \{\nearrow\} \\ &= 1 \end{aligned}$$

More interestingly, given an arithmetic equation given as a first-order formula (which by definition does not contain \forall nor \neg), we can apply John's theorem ([theorem 1.3.3](#) on page 23), and we get that the solutions obtained with the $S_{\mathbb{S}}$ reflect the solutions we can obtain with $S_{\mathbb{R}}$.

1.5 First-Order Ordinary Differential Equations in FOL

Since Newton, mankind has come to realise that the laws of physics are always expressed in the language of differential equations.

Strogatz in [Str09]

Differential equations are a well-established mathematical tool extensively applied to model all kinds of phenomena in all kinds of fields. Their power comes from the fact they do not say explicitly what the next configuration of the system under study is, but instead toward where a configuration tends to evolve. Indeed, it is often easier to describe locally the evolution of some variables than it is to describe why they have some particular values at some point in time. For example, it is simpler to explain why population sizes grow or shrink than to give the exact values at a given time.

Mathematically, a differential equation is an equation that uses functions and function derivative(s):

$$\dot{y} = f(t, y)$$

Differential equations come in different flavours, but in this thesis, we will only deal with *autonomous first-order ordinary differential equations*. We abbreviate this as ODEs.

Here, *first-order* should not be mistaken with the first-order logic introduced earlier. Instead, it relates to the fact that higher-order derivatives (such as \ddot{x}) are not permitted. However, note that we will use first-order logic to encode the differential equations. The term *ordinary* means the involved functions and derivatives use a single input (thought of as time). Finally, *autonomous* means the equations do not explicitly depend on time. The relation between a function and its derivatives is thus fixed in time.

We now give a formal definition of the syntax and semantics of ODEs. It is based on notions from the first-order logic (see [section 1.2.2](#)). It is not the most common formulation, but this will pay off later when we will extract the dynamics of reaction networks ([chapter 3](#)).

1.5.1 Syntax

Let $\dot{\Sigma}_{arith} = (\mathbb{R}, \{+, \times, -, /, \text{pow}, \cdot\}, \stackrel{\circ}{=})$ denote the differential signature. Notice the addition of the dot operator \cdot compared to the signature Σ_{arith} introduced in [section 1.4](#). A term $t \in \mathcal{T}$ can be a variable (possibly dotted), a constant, or a sum, a difference, a multiplication, a division

or a pow of two terms. An ODE is a first-order formula $\phi \in \mathcal{F}_{\Sigma_{arith}}(\mathcal{V})$ we can build with the following BNF:

$$\begin{aligned}
 \text{ODE} &::= \text{dottedvar} \doteq \text{baseterm} \\
 \text{dottedvar} &::= \dot{v} \quad v \in \mathcal{V} \\
 \text{baseterm} &::= v \quad v \in \mathcal{V} \\
 &\quad | \quad \text{baseterm} + \text{baseterm} \\
 &\quad | \quad \text{baseterm} - \text{baseterm} \\
 &\quad | \quad \text{baseterm} \times \text{baseterm} \\
 &\quad | \quad \text{baseterm} / \text{baseterm} \\
 &\quad | \quad \text{pow}(\text{baseterm}, \text{baseterm}) \\
 \text{term} &::= \text{baseterm} \quad | \quad \text{dottedvar}
 \end{aligned}$$

With this definition, the ODEs can capture the kind of behaviour we are interested in. In particular, classic chemical kinetics with mass-action law, Michaelis-Menten law and Hill law.

Example 1.5.1 (The simplest possible ODE). *The following expression is an ODE:*

$$\dot{x} \doteq 0$$

Definition 1.5.2 (ODE system). *An ODE system is a first-order logic formula $\phi \in \mathcal{F}_{\Sigma_{arith}}(\mathcal{V})$ consisting of an ODE or the conjunction of ODEs. The free variables $fv(\phi)$ of the formula ϕ are undotted variables.*

Example 1.5.3. *The following first-order formula is an ODE system:*

$$\begin{aligned}
 \dot{A} &\doteq 0 \\
 \wedge \dot{B} &\doteq A \\
 \wedge \dot{C} &\doteq B + 1
 \end{aligned} \tag{1.1}$$

1.5.2 Usual Semantics

From a logic point of view, the variables represent real-valued functions (in the mathematical sense, not in the logic sense), the arithmetic operators $+$, \times , $-$, $/$ and pow are interpreted as the usual arithmetic operations in the structure of real-valued functions $\mathbb{R} \rightarrow \mathbb{R}$, such as in [section 1.4](#). Finally, the unary operator $\dot{}$ applied to some variable x returns the derivative of x , if x is derivable, and returns undefined otherwise.

When we are given an ODE (or an ODE system) represented by the formula ϕ , we only know how do the derivatives of the functions behave. Solving an ODE (system) consists in finding the functions which respect the ODE specifications. These functions are called *trajectories*. In order to find the functions themselves, we need to somehow integrate the derivative. However, an ODE (system) alone does not *uniquely* determine the solutions. One thus must provide some additional conditions, referred to as *initial conditions*.

Example 1.5.4. *If we try to solve [example 1.5.1](#), we know that x represents a constant function. But there is no way to specify the exact value of the constant x takes. If we set $x(0) = 42$ as initial condition, the solution is now uniquely defined.*

The process is also illustrated on a more complicated example in [section 3.3.1.1](#) on page 79.

From a logic point of view, these functions correspond to the solutions of ϕ over the structure of real-valued functions, i.e., $\text{sol}^{\mathbb{R} \rightarrow \mathbb{R}}(\phi)$. For each such solution β and variable $x \in \text{fv}(\phi)$, $\beta(x) : \mathbb{R}_+ \rightarrow \mathbb{R}$ is a trajectory of x .

In this thesis, we will assume that for any fixed initial values $(\alpha_0 : \text{fv}(\phi) \rightarrow \mathbb{R})$, there exists one solution $\beta \in \text{sol}^{\mathbb{R} \rightarrow \mathbb{R}}(\phi)$, such that $\beta(x)(0) = \alpha_0(x)$ for all $x \in \text{fv}(\phi)$. But in general, it is not necessarily the case. Formally, given a first-order differential equation $\dot{y} = f(t, y)$, the Picard-Lindelöf theorem (also known as the Cauchy-Lipschitz theorem) states that if f and the partial derivative $\frac{\partial f}{\partial y}$ is continuous in some neighbourhood of the initial condition point (t_0, y_0) , then there is a solution on an interval T about t_0 , and it is unique [NSS12]. A proof can be found in the literature [Wal98]. This theorem has stronger results when additional assumptions are made on f . Also, it can be extended to a system of n first-order differential equations [AAL93].

[Table 1.4](#) illustrates the descriptive power of differential equations on several examples. That is, the fact ODEs are easier (more concise) than their solutions. Indeed, constants ODEs have linear functions as solutions. Linear ODEs have exponential solutions, sine and cosine functions come from pretty simple linear differential equation systems.

The solutions are closed form if they consist of a finite sequence of elementary functions (no $\sqrt{\cdot}$, \cos and \log for example).

ODE solutions can be approximated numerically [DR15]. This is particularly useful for ODEs that cannot be solved explicitly. Several algorithms exist. Some will be mentioned in [chapter 3](#). But they all roughly consist in starting from a given configuration and following some direction based on what the equations are indicating for a duration based on a chosen timestep. Moreover, the accuracy of the approximated solution heavily depends on the chosen timestep, rounding errors, and equation stiffness, as will see in [chapter 3](#) as well.

Ultimately, the trajectory can be attracted to a specific value (fixed point) or be cyclic. Fixed-point and cycles are both referred to as attractors. One can also be interested into the stability of the attractors, because close to an unstable attractor, any small perturbations can lead the system to very different states. This is what is called a bifurcation.

There exist many mathematical and computational tools developed to simulate and analyse ODEs which have been used to study biological systems, in particular see [Stä+21; DR15] for numerical simulation, [Ter+19] for steady state identification and [Csi+06; BT04; TCN01] for bifurcation analysis.

1.6 Boolean functions

A Boolean function is a function from \mathbb{B}^n to \mathbb{B} . It thus maps points in \mathbb{B}^n to either 0 or 1.

Definition 1.6.1 (Truth table, True-point, False-point). *The function table of a Boolean function is commonly referred to as a truth table. Each input is identified with a decimal number corresponding to the input value in base 2. An input for which the function returns 1 (respectively 0) is called a True-point (respectively False-point).*

Table 1.4: Some ODEs and their solutions. α, β, C are constant.

	ODE	Encodable in Σ_{arith}	Solutions $sol_{\mathbb{R} \rightarrow \mathbb{R}}^{\mathbb{R}}(\phi)$	Encodable in Σ_{arith}
(constant)	$\dot{f}(t) = 0$	✓	$f(t) = C$	✓
	$\dot{f}(t) = \alpha$	✓	$f(t) = \alpha t + C$	✓
(linear)	$\dot{f}(t) = f$	✓	$f(t) = Ce^t$	✓
	$\dot{f}(t) = \alpha f$	✓	$f(t) = Ce^{\alpha t}$	✓
	$\dot{f}(t) = \alpha f + \beta$	✓	$f(t) = Ce^{\alpha t} - \frac{\beta}{\alpha}$	✓
(non-linear)	$\dot{f}(t) = f^2$	✓	$f(t) = -1/(t + C)$	✓
	$\dot{f}(t) = f^\alpha$	✓	$f(t) = -1/\sqrt[\alpha-1]{(1-\alpha)t + C}$	✓
	$\dot{f}(t) = f^2 + 1$	✓	$f(t) = \tan(t + C)$	
	$\dot{f}(t) = f^\alpha + \beta$	✓	$f(t) = \sqrt[\alpha]{\beta}$	✓
	$\dot{f}(t) = 1/f$	✓	$f(t) = \pm \sqrt{2t + C}$	(non-unique)
	$\dot{f}(t) = \sqrt{f}$	✓	$f(t) = \left(\frac{t}{2} + C\right)^2$	✓
(system)	$\begin{cases} \dot{f}(t) = g \\ \dot{g}(t) = -f \end{cases}$	✓	$\begin{cases} f(t) = C_1 \times \sin(t) + C_2 \times \cos(t) \\ g(t) = C_1 \times \cos(t) - C_2 \times \sin(t) \end{cases}$	

Example 1.6.2 (The “exclusive or” Boolean function). *The following truth table defines a Boolean function $f(A, B)$ which evaluates to 1 when either the value of A or of B (but not both!) is 1:*

	input	output
	AB	$X = f(A, B)$
0	00	0
1	01	1
2	10	1
3	11	0

1.6.1 Representation with Hypercube

The concepts related to Boolean functions have a simple (and useful) geometric interpretation based on hypercubes.

Let U^n be the unit hypercube of size n , i.e., the graph whose vertices consist of all the Boolean vectors of length n : $V(U^n) = \{(x_1, \dots, x_n) \mid x_i \in \{0, 1\}\}$. Two vertices are adjacent if they differ in exactly one coordinate. This is similar to [definition 1.2.7](#) on page 16. The unit hypercubes for n from 0 to 3 are depicted in [figure 1.7](#).

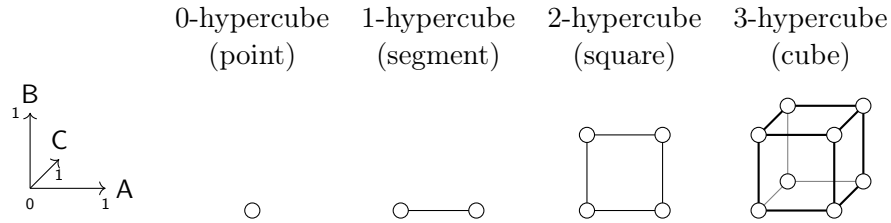
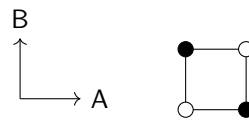


Figure 1.7: Unit hypercubes from size zero to size three.

The mapping defined by a Boolean function of arity n can be represented on the unit hypercube U^n . As it is just another way to represent the truth table, the hypercube representation fully characterises the function without losing information. By convention in this thesis, **False**-points will be represented by empty circles and **True**-points by filled (coloured) circles.

Example 1.6.3 (Hypercube representation of a Boolean function). *The hypercube representation of the function defined in [example 1.6.2](#) is:*



1.6.2 Representation with DNF Propositional Formulas

The definition of a Boolean network given later in this thesis ([definition 1.8.1](#) on page 40) relies on Boolean functions that are represented by propositional formulas in non-redundant Disjunctive Normal Form ([definition 1.2.12](#)).

As stated before, a Boolean function f of arity n is a function $f : \mathbb{B}^n \rightarrow \mathbb{B}$. Propositional formulas, in particular those in DNF, can be used to represent Boolean functions. A given

propositional formula φ (in DNF or not) on n variables x_1, \dots, x_n encodes a unique Boolean function f_φ of arity n . To retrieve the function encoded by a formula φ , one computes the value $\llbracket \varphi \rrbracket_\iota$ for each of the 2^n possible interpretation ι of the n variables of φ . In turn, a Boolean function of arity n can be represented by propositional DNF formulas on n variables. However, it exists an infinite number of equivalent propositional DNF formulas on n variables: general DNF formulas are thus not canonical representation of Boolean functions.

Example 1.6.4. *The formulas A , $A \vee A$, $A \vee A \vee A$ encode for the same function whose truth table is*

	input A	output $f(A)$
0	0	0
1	1	1

Example 1.6.5. *The propositional expression $(A \wedge \neg B) \vee (\neg A \wedge B)$ is in DNF and encodes the Boolean function whose truth table is given in [example 1.6.2](#). The formula $(A \vee B) \wedge (\neg A \vee \neg B)$ on the other hand, encodes the same Boolean function but is not in DNF.*

1.6.3 Minimal Expressions

Later in this thesis, we will be interested in formulas which are as short as possible. Since we are only interested in expressions given in DNF, we assume that the expressions are all given in DNF from now on.

The size of a formula φ , denoted $|\varphi|$, corresponds to the number of literals used in φ .

Example 1.6.6 (Formula length). *The formula A has length 1 (one variable appearing once), and $A \wedge A$ has length 2 (one variable appearing twice).*

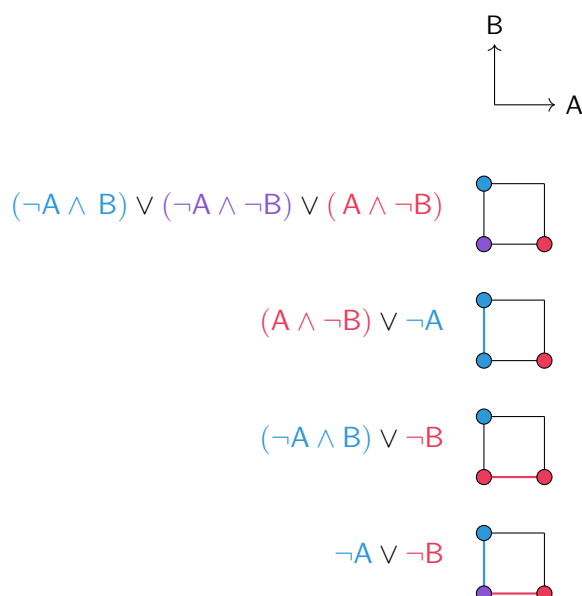
Formula minimisation is the process of shortening a formula. It is a hard problem in the general case. The computational complexity of the logic minimisation problem depends on the representation of the input. It is *NP*-hard when the Boolean function is given by the set of its True-points. The DNF minimisation process results in the generation of formulas in minimal Disjunctive Normal Form (minDNF).

Definition 1.6.7 (Subset minimal DNF). *Among the infinite number of DNF one can build to represent a given truth table, the smallest ones are called (subset) minimal DNF.*

Example 1.6.8. *The truth table*

	input AB	output X
0	00	1
1	01	1
2	10	1
3	11	0

can be represented by an infinite number of DNFs. In particular, the four DNFs below. The hypercube representations indeed prove that they encode for the same function. However, the shortest possible one, i.e., the *minDNF*, is $\neg A \vee \neg B$.

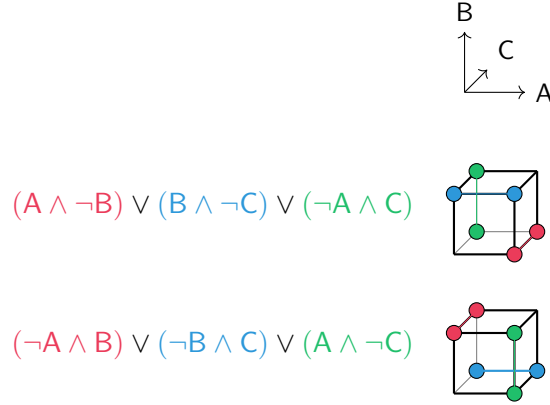


MinDNF is not a canonical representation since there are Boolean functions which can be represented by several equivalent minDNF.

Example 1.6.9. The function encoded by the following truth table

	input	output
	ABC	X
0	000	0
1	001	1
2	010	1
3	011	1
4	100	1
5	101	1
6	110	1
7	111	0

can be encoded with two minDNFs:



When given a partial truth table, each completion of the truth table corresponds to a set of minDNF. The smallest ones are called cardinal minimal DNF.

Definition 1.6.10 (Cardinal minimal DNF). *A DNF is cardinal-minimal if it is the smallest subset-minimal DNFs compatible with a partial truth table.*

Example 1.6.11. *We consider the following partial truth table.*

	input	output
	AB	X
0	00	1
1	01	1
2	10	1
3	11	*

As seen in [example 1.6.8](#), $\neg A \vee \neg B$ is a subset-minimal DNF if the completion for the last line is 0. However, if the completion is 1, the subset-minimal DNF is the constant 1, which is smaller than the expression mentioned above (it has a size of zero since it has no literal at all). It is thus a cardinal-minimal DNF.

We now describe minimal DNF more formally. The description relies on the notions of *implicants*, *prime implicants* and *redundant prime implicants*. These concepts were introduced by Quine [Qui52].

Definition 1.6.12 (Implicant). *Given two Boolean functions f and g , we say that f implies g (or f is an implicant of g) if $f(X) = 1 \Rightarrow g(X) = 1$ for all $X \in \mathbb{B}^n$. Since every Boolean expression can be regarded as a Boolean function, we will also say indifferently that expression f implies expression g .*

If φ is a DNF representation of the Boolean function f , then every conjunction of φ is an implicant of f .

Definition 1.6.13 (Prime implicant). *An implicant is prime if it ceases to be an implicant if any of its literal is removed.*

Example 1.6.14. *Let φ be $B \wedge (\neg A \vee C)$. The expression ABC is an implicant of ϕ (since $(A \wedge B \wedge C) \Rightarrow \phi$). However, it is not prime since we can obtain a shorter 1-implicant by removing the literal A : $(B \wedge C) \Rightarrow \phi$.*

Example 1.6.15. Similarly, ABC is an implicant of $ABC \vee AB$ but it is not prime because AB is a shorter implicant.

The conjunctions in a minDNF are necessary *prime implicants* [Qui52]. Moreover, they exhibit interesting patterns we will rely on in [chapter 4](#). In particular:

- The set of subset minDNF one can build with n variables is included in the set of non-redundant DNF.
- The conjunctions are *locally non-adjacent* to one another i.e., if they contain the same variables, they differ in more than one literal ([definition 1.2.7](#)). Indeed, if two conjunctions are locally adjacent, they can be replaced by a shorter conjunction.

Example 1.6.16. The two conjunctions in the following DNF are locally adjacent: $A \wedge B \vee A \wedge \neg B$. They can be replaced by the expression A .

- The conjunctions do not subsume one another i.e., for all conjunctions $c = \ell_1 \wedge \dots \wedge \ell_m$ and $c' = \ell'_1 \wedge \dots \wedge \ell'_n$, we don't have that, for all i such that $1 \leq i \leq m$, there exists j such that $1 \leq j \leq n$ such that $\ell_i = \ell'_j$ ([definition 1.2.9](#)). Indeed, subsumed conjunction can be removed.

Example 1.6.17. Let us consider the following DNF: $A \wedge \neg B \vee A \wedge \neg B \wedge C$. The second conjunction is subsumed by the first one. The DNF can be rewritten $A \wedge \neg B$.

Several methods have been proposed for DNF minimisation. Logic minimisation problems can be reduced to set covering problems [CH11]. Usually, the algorithm used is the one of Quine-McCluskey [McC56]. It is exact and can return all the solutions when several exist. Other algorithms such as Espresso [Bra+82] (available in the tool Logic Friday) only return an approximated result.

1.6.4 Specific Classes of Boolean Functions

In the thesis, we will refer to some specific classes of functions. The number of functions for some of these classes is given in [table 1.5](#).

Definition 1.6.18 (Variable essentiality, degenerated function). A variable i is essential for a Boolean function f of size n if there exist two inputs X and Y (in \mathbb{B}^n) that are identical except on their i -th coordinate and for which $f(X) \neq f(Y)$. The function f is degenerated if it has (at least) one of its variables that is non-essential. Otherwise, f is said to be non-degenerated.

Example 1.6.19 (Degenerated function). The expression $(A \wedge B) \vee A$ encodes for a degenerated function f that takes a priori two input variables A and B , but by checking the truth table of f , we can see that B is actually not essential:

	input AB	output $f(A, B)$
0	00	0
1	01	0
2	10	1
3	11	1

The truth table above can thus be represented by an alternative formula **A**, that uses only one input variable instead of two.

Definition 1.6.20 (Input-wise monotony). *A Boolean function f is 1-monotone in its variable i if for all input X and Y where X equals Y except on i where $X_i = 0$ and $Y_i = 1$, we have $f(X) \leq f(Y)$. Symmetrically, f is 0-monotone in i if $X < Y \implies f(X) \geq f(Y)$. The function f is monotone in i if it is either 1-monotone or 0-monotone in i .*

Definition 1.6.21 (1-monotone, 0-monotone and monotone function). *A Boolean function f of size n is 1-monotone if it is 1-monotone on all its inputs. Symmetrically, it is 0-monotone if it is 0-monotone on all its inputs. The function f is monotone if it is 1-monotone or 0-monotone.*

The output of a 1-monotone function cannot return 0 by simply setting one of its input to 1. Graphically, a Boolean function is 1-monotone when its hypercube representation has no “upward” edge from 1 to 0. A 1-monotone Boolean functions can be represented by monotone formulas, i.e., formulas that contain only conjunctions, disjunctions but no negations.

Conversely, a Boolean function is 0-monotone when its hypercube representation has no “upward” edge from 0 to 1.

The number of 1-monotone Boolean functions of arity n is counted by the Dedekind numbers D_n . They are known up to $n = 9^4$ (table 1.5), and for higher n , they are estimated by the following (tight) bounds:

$$\binom{n}{\lfloor n/2 \rfloor} \leq \log_2(D_n) \leq \binom{n}{\lfloor n/2 \rfloor} \times \left(1 + \mathcal{O}\left(\frac{\alpha \log n}{n}\right)\right)$$

Definition 1.6.22 (Bipolar function [BS17]). *A Boolean function f is bipolar if it is monotone in each of its input variable.*

Example 1.6.23. *The “exclusive or” function (whose truth table is shown in example 1.6.2 on page 31) is the classical example of non-bipolar function. Indeed it is non-monotone in all its inputs: $f(00) = 0$ and $f(10) = 1$, but $f(01) = 1$ and $f(11) = 0$. Hence f is non-monotone in **A**. Moreover, $f(00) = 0$ and $f(01) = 1$, but $f(10) = 1$ and $f(11) = 0$. Hence f is non-monotone in **B**.*

In the literature, some of the concepts introduced above take several inconsistent names. Below, we summarise some of these alternative names, and where they appear. Note that in this thesis, we will use the ones in bold.

1-monotone [ADG04] = monotone [ADG04]

0-monotone

bipolar [BS17] = unate [SMS22], monotone [CMC19]⁵

⁴ The result for $n = 9$ is very new [Jäk23; Van+23].

⁵ I am not so sure for this paper because it seems to me that the authors use the definition of bipolar Boolean functions while saying they are counted with the Dedekind number.

In summary, the set of 1-monotone functions (resp. 0-monotone functions), is the set of function where the variables have at most a positive influence, noted $+$ (resp. a negative influence, noted $-$), while in the set of bipolar function, we consider both signs for all the variables. This distinction is useful to understand the number of Boolean functions one can have given a set of variables: if we allow only simple sign influences, it is counted by the Dedekind numbers while if everything is with a double sign, it is bipolar. Otherwise, we are in between.

Enforcing monotone influences only is a useful constraint for search space reduction when synthesising Boolean functions, and it captures some ground truth about how influences between the species work in biological systems (see [section 2.4.1.3](#)).

1.7 Models for Dynamical Systems

In this thesis, we will focus on reaction network models and Boolean network models. They are both some kind of dynamical models used to describe *dynamical systems*, so they share a lot of concepts. Roughly speaking, a dynamical system consists of a set of components (species) that have a given status. At a given time, the configuration of the complete system is formed of the status of the components of the system. The system is also defined by a *transition function*, that returns the (set of) configuration(s) that are reachable from a given configuration. The exploration of the configuration landscape leads to the construction of a *transition graph*. The system is also associated with an *influence graph*, that summarises the influences between the components of the system (whether a component tends to favour or disfavour the presence of another component). Hereafter, we introduce the formal definitions for configuration, transition function, transition graph and influence graph.

Let \mathcal{S} be a finite set of *species* of interest, and V an arbitrary set of values. Depending on the context, elements in \mathcal{S} can for example, represent pools of genes, proteins, animals, etc.

Definition 1.7.1 (Dynamical model). *A dynamical model over a finite set of species \mathcal{S} is a tuple (\mathcal{C}, f) , with \mathcal{C} an arbitrary subset of \mathcal{S}^V , with V an arbitrary set of values and f a function $f : \mathcal{C} \rightarrow \mathcal{C}$. The set \mathcal{C} is called the set of configurations and the function f is called the transition function.*

Definition 1.7.2 (Configuration). *A configuration over species in \mathcal{S} is a function $C : \mathcal{S} \rightarrow V$, with V an arbitrary set of values. A configuration is also denoted $[X : C(X) \mid X \in \mathcal{S}]$, and the value $C(X)$ taken by a species X in a configuration C will sometimes be noted C_X .*

When $V = \mathbb{N}$, the configuration C is a multiset and can for example, be used to keep track of the precise amount of the chemical species in solutions. Following the classical chemical notation, we will write such a multiset as $\sum_{X \in \mathcal{C}} C(X) \times X$. When $V = \mathbb{R}$, the function can for example, be used to model the concentrations, or an average number of some molecules. When $V = \mathbb{B}$, a configuration can be represented with a bit vector once the ordering of the species has been fixed.

Example 1.7.3. *The configuration $[A:3, B:2]$, also denoted by $3 \times A + 2 \times B$, may represent a chemical solution in which there are 3 molecules of a species A , and 2 of a species B . It can also be a number of moles, or a number of moles per liter.*

Table 1.5: Number of Boolean functions of size n (for $n \leq 8$) in various classes.

n	Total (2^{2^n})	# non-degenerate A000371 ⁶	# 1-monotone (or 0-monotone) A000372 ⁷	# bipolar A245079 ⁸
0	2	2	2	2
1	4	2	3	4
2	16	10	6	14
3	256	218	20	104
4	65 536	64 594	168	2170
5	4 294 967 296	4 294 642 034	7581	230 540
6	18 446 744 073 710 000 000	18 446 744 047 940 725 978	7 828 354	499 596 550
7	$> 340 \times 10^{36}$	$> 340 \times 10^{36}$	2 414 682 040 998	30 907 579 915 064
8	$> 115 \times 10^{75}$	$> 115 \times 10^{75}$	56 130 437 228 687 557 907 788	⁹ 5 483 950 159 845 307 762

⁶ <https://oeis.org/A000371>

⁷ <https://oeis.org/A000372>

⁸ <https://oeis.org/A245079>

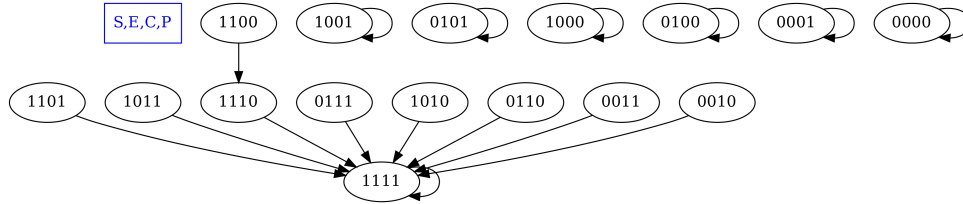
⁹ Maybe there is an error in the OEIS encyclopedia because it is weird that this number would be smaller than the number of 1-monotone functions.

Example 1.7.4 (Boolean configuration). For instance, the configuration [S:1, E:1, C:0, P:1] can be identified with the bit vector 1101 when ordering the species as S, E, C, P.

Given two configurations C and C' , the set of components that have a different value in C and in C' is denoted by $\Delta(C, C') = \{X \in \mathcal{S} \mid C(X) \neq C'(X)\}$. Two configurations can be compared. For a fixed order of components, we say $C < C'$ if $\text{dom}(C) \subseteq \text{dom}(C')$ and $\forall X \in \text{dom}(C) : C(X) < C'(X)$.

Definition 1.7.5 (Transition graph). A transition graph over configurations in \mathcal{C} is a directed graph $G = (\mathcal{C}, E)$, where \mathcal{C} is a set of configurations and E a set of directed edges. In the figures, the configurations are drawn as bit vectors, respecting the provided order, and the transitions between configurations as arrows linking these nodes.

Example 1.7.6 (Transition graph on Boolean configurations). Below is a Boolean transition graph that represents the transitions between Boolean configurations of a system with four species (S, E, C and P).

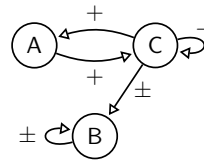


Definition 1.7.7 (Influence graph). An influence graph over species in \mathcal{S} is a directed signed graph $G = (\mathcal{S}, E)$ where E is a set of signed directed edges (Y, X) that represent the influence that the species $Y \in \mathcal{S}$ has on the species $X \in \mathcal{S}$. The sign of the edge encodes whether Y favours (+) or disfavours (-), or both/unknown (\pm in both cases) the presence of X .

Given an influence graph $G = (\mathcal{S}, E)$ and a species $X \in \mathcal{S}$, the set of parents of X is denoted $\text{parents}(X)$ and correspond to all the species Y in \mathcal{S} such that $(Y, X) \in E$.

Example 1.7.8. According to the influence graph given below, C and A are the parents of C.

- A activates C
- B activates itself
- B inhibits itself
- C activates A
- C activates B
- C inhibits B
- C inhibits itself



1.8 Boolean Networks

This section is dedicated to formally define what a Boolean network is, as well as some related key concepts. We also highlight some links between static (syntactic) and dynamical (semantic) properties of Boolean networks, as they will be exploited in the chapter on Boolean network synthesis (chapter 4).

1.8.1 Syntax

Let \mathcal{S} be a set of n species and $\mathcal{S}_{\text{next}}$ a set obtained from \mathcal{S} by the following bijection $\mathcal{S}_{\text{next}} = \{ \mathbf{X}_{\text{next}} \mid \mathbf{X} \in \mathcal{S} \}$. For example, if $\mathcal{S} = \{A, B, C\}$, $\mathcal{S}_{\text{next}} = \{ \mathbf{A}_{\text{next}}, \mathbf{B}_{\text{next}}, \mathbf{C}_{\text{next}} \}$.

Definition 1.8.1 (Boolean network). *A Boolean Network with species on \mathcal{S} is given as a propositional formula of the form:*

$$\bigwedge_{\mathbf{X} \in \mathcal{S}} \left(\mathbf{X}_{\text{next}} \Leftrightarrow \phi_{\mathbf{X}}(\mathcal{S}) \right),$$

and where $\phi_{\mathbf{X}}(\mathcal{S})$ is a propositional formula on \mathcal{S} .

Remark 1.8.2. *In this thesis, we will enforce each propositional formulas $\phi_{\mathbf{X}}(\mathcal{S})$ to be in minimal disjunctive normal form (minDNF). In contrast with using arbitrary expressions, we have a finite number of possible expressions for each species. Still, we can represent all the Boolean functions we may want to represent, since any Boolean functions correspond to a set of minDNF ([section 1.6](#)).*

Example 1.8.3. *[Figure 1.8](#) shows two Boolean networks with species on $\mathcal{S} = \{A, B, C\}$.*

1.8.2 Semantics

Given a variable assignment $\alpha : \mathcal{S} \rightarrow \mathbb{B}$, $\llbracket \mathbf{X} \rrbracket_{\alpha}$ can be understood as the *current* status of \mathbf{X} . The interpretation of $\llbracket \phi_{\mathbf{X}} \rrbracket_{\alpha}$ computes \mathbf{X}_{next} , which represents the *next* status of \mathbf{X} .

Example 1.8.4. *The transition function associated with \mathbf{B} in \mathcal{B}_2 ([figure 1.8f](#)) states that the value of \mathbf{B} will be 1 if either the value of \mathbf{B} or that of \mathbf{C} was 1 in the previous configuration.*

Altogether, a Boolean network with species on \mathcal{S} such as defined in [definition 1.8.1](#) represents a (global) transition function $f : \mathbb{B}^{|\mathcal{S}|} \rightarrow \mathbb{B}^{|\mathcal{S}|}$ between Boolean configurations. Alternatively, we can see f as a *system* of $|\mathcal{S}|$ Boolean functions $f_{\mathbf{X}} : \mathbb{B}^n \rightarrow \mathbb{B}$ (one per species $\mathbf{X} \in \mathcal{S}$). These functions correspond to the projection of f onto each of its axes, and they are referred to as (*local*) *transition functions*.

Example 1.8.5. *The following Boolean function:*

$$f : \mathbb{B}^2 \rightarrow \mathbb{B}^2, [(01, 00), (01, 01), (10, 01), (11, 11)]$$

can be projected on each of its two axes as:

$$f_1 : \mathbb{B}^2 \rightarrow \mathbb{B}, [(01, 0), (01, 0), (10, 0), (11, 1)]$$

and

$$f_2 : \mathbb{B}^2 \rightarrow \mathbb{B}, [(01, 0), (01, 1), (10, 1), (11, 1)]$$

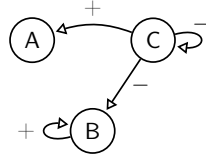
The function f is thus equivalent to the system formed of f_1 and f_2 .

$$\begin{aligned}\mathcal{B}_1 = & \left(A_{\text{next}} \Leftrightarrow C \right) \\ & \wedge \left(B_{\text{next}} \Leftrightarrow B \wedge \neg C \right) \\ & \wedge \left(C_{\text{next}} \Leftrightarrow \neg C \right)\end{aligned}$$

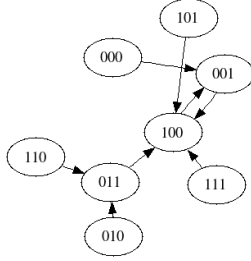
(a) Propositional formula representing the Boolean network \mathcal{B}_1 .

$$\mathcal{B}_1 = \begin{cases} f_A : A_{\text{next}} = C \\ f_B : B_{\text{next}} = B \wedge \neg C \\ f_C : C_{\text{next}} = \neg C \end{cases}$$

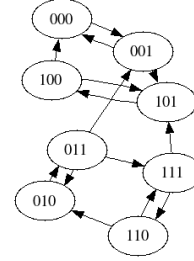
(b) System of Boolean functions represented by the formula [figure 1.8a](#).



(c) Influence graph of \mathcal{B}_1 .



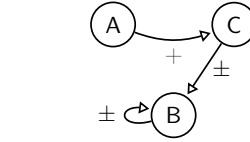
(d) Synchronous transition graph of \mathcal{B}_1 .



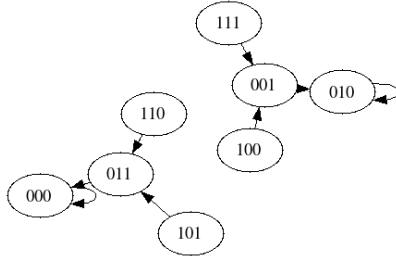
(e) Asynchronous transition graph of \mathcal{B}_1 .

$$\begin{aligned}\mathcal{B}_2 = & \left(A_{\text{next}} \Leftrightarrow 0 \right) \\ & \wedge \left(B_{\text{next}} \Leftrightarrow (B \wedge \neg C) \vee (\neg B \wedge C) \right) \\ & \wedge \left(C_{\text{next}} \Leftrightarrow A \right)\end{aligned}$$

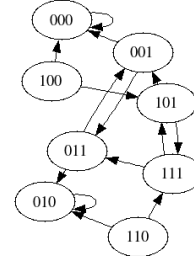
(f) Propositional formula representing the Boolean network \mathcal{B}_2 .



(g) Influence graph of \mathcal{B}_2 .



(h) Synchronous transition graph of \mathcal{B}_2 .



(i) Asynchronous transition graph of \mathcal{B}_2 .

Figure 1.8: The transition functions, derived influence graph, and transition graphs according to synchronous and asynchronous update schemes of two Boolean networks.

1.8.3 Dynamics: Transition Graphs

The global transition function $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ of a Boolean network over n species in \mathcal{S} in fact encodes the state *towards which* each species *tends*, given one of its 2^n possible configurations. A *trace* of a Boolean network corresponds to the precise evolution of a Boolean network. It is computed by applying iteratively the local transition functions, starting from a given configuration, in an order of application fixed by a chosen *update scheme*. In fact, the dynamics strongly depends on the chosen update scheme [PS22].

The common representation to summarize the possible traces of a Boolean network is a transition graph (definition 1.7.5) whose nodes are the 2^n possible configurations of the Boolean network. In this graph, there is a directed edge from C to C' if C' is the result of applying the transition function(s) to C , according to the chosen update scheme.

The chosen update scheme abstracts reactions time by defining the order of application of the transition functions. Formally, it consists in the application of a sequence (composition) of the elementary deterministic updates, encoded by the transition functions. If different elementary updates can be applied to the same configuration, we ultimately end up generating non-deterministic traces. There exists an infinite number of update schemes (corresponding to the infinite number of sequence of elementary updates), but the most commonly used ones are the synchronous, the asynchronous and the general-asynchronous update schemes.

In the *synchronous* update scheme, the transition functions are applied all at once [Kau69]. In the strict *asynchronous* update scheme, the transition functions are applied one by one, nondeterministically [Tho91]. In the *general-asynchronous* update scheme, any number of components can be updated at each step. It is the most general classic update scheme in the world of automata networks, as it captures all the transitions allowed from all the other existing update schemes (including the synchronous and asynchronous update schemes).

The three update schemes presented above are *incomplete* in the sense there are some cases in which they fail to capture a configuration sequence despite the logic to be correct. It is, for example, justified when the underlying processes actually use several thresholds for a species. The *most-permissive semantics* solves this. It consists in relaxing the status of the species thanks to the addition of an intermediate status $*$. If a species is given the $*$ status, the species it influences can non-deterministically consider it as being 0 or 1 [Pau+20].

Example 1.8.6. *Figure 1.8 shows examples of synchronous and asynchronous transition graphs.*

1.8.4 Structure: Influence Graph

The *structure* of a Boolean network is defined in terms of parent-child relationships between the components. These relationships are summarised as an *influence graph* (definition 1.7.7). In the following, this relationship is primarily defined from a dynamical point of view. However, if the logical functions are given in minDNF, they can also be retrieved syntactically from the polarity of the literals in the transition functions themselves.

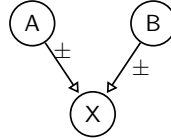
Given the transition function f_X of a species $X \in \mathcal{S}$, the *parents* of X are the components $P \in \mathcal{S}$ such that there exist two configurations C and C' which only differ on P such that $f_X(c) \neq f_X(c')$. The *polarity* of the influence of P on X is determined by looking at the order preservation:

Case positive influence If there are $C < C'$ such that $f(C) < f(C')$ (meaning $C(P) = 0$, $C'(P) = 1$, $f(C) = 0$ and $f(C') = 1$) we say that the polarity of the influence of P on X is *positive* (noted $+$).

Case negative influence Conversely, if there are $C < C'$ such that $f(C) > f(C')$ (meaning $C(P) = 0$, $C'(P) = 1$, but $f(C) = 1$ and $f(C') = 0$) we say that the polarity of the influence of P on X is *negative* (noted $-$).

A component can have both a positive and negative influence (noted \pm).

Example 1.8.7. The influence graph for the function $f_X = (\neg A \wedge B) \vee (A \wedge \neg B)$ is



Indeed, we can see in the truth table given in [example 1.6.2](#) on page 31 that there are configurations such that both cases presented above apply, for both species A and B :

- $A \xrightarrow{+} X$ because $f_X(00) = 0$ and $f_X(10) = 1$
- $A \xrightarrow{-} X$ because $f_X(01) = 1$ and $f_X(11) = 0$
- $B \xrightarrow{+} X$ because $f_X(00) = 0$ and $f_X(01) = 1$
- $B \xrightarrow{-} X$ because $f_X(10) = 1$ and $f_X(11) = 0$

If the transition functions are given as minDNF, it is possible to retrieve the influences syntactically. If a component P appears in the transition function of a component X , it is a parent of X . If P is negated, the polarity of its influence on X is negative. Conversely, if the parent is not negated, the polarity is positive. In case P has both a positive and a negative influence on X , the influence is non-monotone.

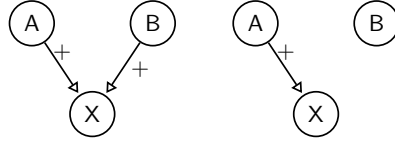
Example 1.8.8. The influence graph of \mathcal{B}_1 ([figure 1.8c](#)) contains $B \xrightarrow{+} B$ and $C \xrightarrow{-} B$ because B appears positively and C appears negatively in the transition function associated with B .

Remark 1.8.9. When we retrieve the influence graph syntactically, it is really necessary for the functions to be in minDNF. If it is not, we have no guarantee to obtain an influence graph that corresponds to the one we would have obtained by checking the dynamics.

Example 1.8.10. Both expressions $E = (A \wedge B) \vee A$ and $E' = A$ represent the same function f whose truth table is

	input AB	output $f(A, B)$
0	00	0
1	01	0
2	10	1
3	11	1

If we retrieve the influence graph syntactically from E and E' , we get two different influence graphs:



The one retrieved from E' is the same that what we get from one retrieved from the dynamics because E' is in *minDNF*, while E is not and introduces spurious influences.

1.9 Reaction Networks

1.9.1 Syntax

A *reaction* \mathcal{R} models a process that transforms a multiset R of species (the reactants) into another multiset P (the products), with a rate e . Formally, a reaction is thus a multiset of rewriting rules. The rate of \mathcal{R} is given by a mathematical expression e . In this thesis, we assume e is from the set $\mathcal{E}_{arith}(\mathcal{S})$ (see [section 1.4](#) on page 23 for details about it).

Definition 1.9.1 (Reaction). *A reaction with species in \mathcal{S} is an element of $\mathbb{N}^{\mathcal{S}} \times \mathcal{E}_{arith}(\mathcal{S}) \times \mathbb{N}^{\mathcal{S}}$. A reaction $\mathcal{R} = (R, e, P)$ is also denoted as*

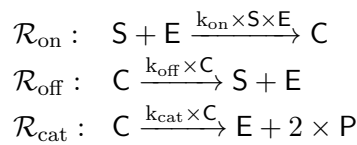
$$\mathcal{R} : \sum_{\mathbf{X} \in R} R(\mathbf{X}) \times \mathbf{X} \xrightarrow{e} \sum_{\mathbf{X} \in P} P(\mathbf{X}) \times \mathbf{X}$$

where $R(\mathbf{X})$ and $P(\mathbf{X})$ are the multiplicities of \mathbf{X} in R and P respectively. For a species $\mathbf{X} \in \mathcal{S}$, the net stoichiometry of \mathbf{X} in the reaction \mathcal{R} is defined as $R(\mathbf{X}) - P(\mathbf{X})$. If the result is negative (resp. positive), it means that \mathbf{X} is effectively consumed (resp. produced) by the reaction \mathcal{R} . A species \mathbf{X} that appears in the rate expression e of a reaction \mathcal{R} but for which $P(\mathbf{X}) - R(\mathbf{X}) = 0$ is called a *modifier*. It influences the speed of the reaction, without having its amount modified. A modifier which increases (resp. decreases) the speed of the reaction is called an *activator* (resp. *inhibitor*) of the reaction.

Example 1.9.2. Let \mathcal{S} be $\{A, B\}$ and $e = 5.1 \times \text{pow}(A, 2) \times B$. The following tuple is a reaction: $\mathcal{R} = (3 \times A + 1 \times B, e, 1 \times A + 2 \times C)$. It states that the 3 molecules of a species A and one of a species B are transformed, with a rate e , into one molecule of A and two molecules of C .

Definition 1.9.3 (Reaction network). *A reaction network with species in \mathcal{S} is a set \mathcal{R} of reactions with species in \mathcal{S} .*

Example 1.9.4 (The reaction network \mathcal{R}_{enz}). *The following reaction network models a simple enzymatic process.*



It has four species $\mathcal{S} = \{S, E, C, P\}$ and three reactions:

1. Reaction \mathcal{R}_{on} transforms a pair of a substrate S and an enzyme E to a complex C .

2. Reaction \mathcal{R}_{off} does the inverse.
3. Reaction \mathcal{R}_{cat} transforms the complex \mathcal{C} into the free enzyme \mathbf{E} and two instances of the product \mathbf{P} .

The reactions follow the mass action law kinetics which state that the speed is directly proportional to the product of the concentrations of reactants [WG64].

Definition 1.9.5 (Complete reaction network). *A reaction network is said to be complete if all the parameters (kinetics constants and initial concentration) are given a value.*

The following property (well-formedness) ensures the consistency of the description of the reactions with their kinetic expression. It will be an essential precondition in [chapter 3](#) to retrieve an influence graph from a reaction network such that it is correct, with regard to the dynamics of the model.

Definition 1.9.6 (Well-formed reaction network). *A reaction network is said to be well-formed if each of its reaction \mathcal{R} respects the following criteria [FGS12]:*

1. Its kinetic expression e is well-defined, positive, and partially differentiable.
2. A species \mathbf{Y} belongs to the set of reactants or activators of \mathcal{R} if and only if $\frac{\partial e}{\partial \mathbf{Y}} > 0$ for some positive values of concentration.
3. A species \mathbf{Y} belongs to the set of inhibitors of \mathcal{R} if and only if $\frac{\partial e}{\partial \mathbf{Y}} < 0$ for some positive values of concentration.

Example 1.9.7 (Not well-formed reaction). *Let us consider the following reaction. $\mathcal{R} = (\mathbf{X}, k \times \mathbf{Y}, _)$. It reads “ \mathbf{X} disappears at a speed $k \times \mathbf{Y}$ ”. \mathbf{Y} appears in the kinetic expression, and has an impact on the degradation of \mathbf{X} (i.e., $\frac{\partial k \times \mathbf{Y}}{\partial \mathbf{Y}} \neq 0$). But despite this, it is not listed as a reactant nor modifier of \mathcal{R} . This breaks the above mentioned criteria 2 and 3. \mathcal{R} is thus not well-formed.*

Extended Reaction Networks In [section 2.3.1.2](#), we present extensions of core reaction networks with features such as additional rules and discontinuous events.

1.9.2 Differential Semantics

As mentioned before, a reaction network defines a transition function on a set of configurations. A semantics is what defines exactly how to interpret and execute the reaction network. A reaction network is usually analysed with one of the following semantics: stochastic, discrete, Boolean and differential (see [section 2.3.1.3](#) for more details).

The differential semantics is the one that is of particular interest in this thesis. It consists of a system of deterministic ordinary differential equations (ODEs), one per species, that are composed of the kinetic expressions of the reactions producing or consuming each species:

$$\text{odes}(\mathcal{R}) =_{\text{def}} \bigwedge_{\mathbf{A} \in \mathcal{S}} \dot{\mathbf{A}} = \sum_{(\mathbf{R}_{\mathcal{R}}, e_{\mathcal{R}}, \mathbf{P}_{\mathcal{R}}) \in \mathcal{R}} (P_{\mathcal{R}}(\mathbf{A}) - R_{\mathcal{R}}(\mathbf{A})) \times e_{\mathcal{R}}$$

This uses the syntax defined in [section 1.5](#) on page 27. In particular, \dot{A} denotes the evolution of the species A over time. With the differential semantics, the rewriting defined by the different reactions is applied synchronously, and the numbers computed are positive reals (that can be interpreted as the amount or concentration of the species).

Example 1.9.8 (\mathcal{R}_{enz} : ODE system). *The differential semantics of \mathcal{R}_{enz} ([example 1.9.4](#) on page 44) consists of the following ODE system:*

$$\begin{aligned}\dot{S} &\stackrel{\circ}{=} -k_{\text{on}} \times E \times S + k_{\text{off}} \times C \\ \wedge \dot{E} &\stackrel{\circ}{=} -k_{\text{on}} \times E \times S + k_{\text{off}} \times C + k_{\text{cat}} \times C \\ \wedge \dot{C} &\stackrel{\circ}{=} k_{\text{on}} \times E \times S - k_{\text{off}} \times C - k_{\text{cat}} \times C \\ \wedge \dot{P} &\stackrel{\circ}{=} 2 \times k_{\text{cat}} \times C\end{aligned}$$

A possible parametrisation of this system is

$$k_{\text{on}} = 10^6 \quad k_{\text{off}} = 0.2 \quad k_{\text{cat}} = 0.1$$

The units of these parameters are respectively mol s^{-1} , s^{-1} , s^{-1} if we are interested in the amount of species (the unit of the derivatives is thus mol s^{-1}), but k_{on} is in $\text{L mol}^{-1} \text{s}^{-1}$ if we deal with concentrations instead.

Chapter 2

Formal Modeling of Biological Systems

Contents

2.1	Introduction	47
2.2	Success Stories in Systems Biology	48
2.3	A Dichotomous Zoo of Model Blueprints	50
2.3.1	Blueprints of Mechanism-Based Models	50
2.3.2	Blueprints of Influence-Based Models	56
2.4	Model Synthesis as a Parameter Fitting Task	58
2.4.1	Input: Available Knowledge and Data	58
2.4.2	Scoring Functions	64
2.4.3	Search Algorithms	65
2.5	Model Analyses and Transformations	67
2.5.1	Structure Analyses	67
2.5.2	Dynamic Analyses	67
2.5.3	Model Simplification	68
2.5.4	Aggregation of Models	71
2.5.5	Conversion Between Models	71
2.6	Wrap-up	73

The main challenges of model validation are the achievement of a match between the precision of model predictions and experimental data, as well as the efficient and reliable comparison of the predictions and observations.

Batt et al. in [Bat+05]

2.1 Introduction

What I propose in this thesis is to convert a given reaction network to a set of corresponding Boolean networks. The method is directly inspired by existing synthesis methods. The goal of the chapter is thus threefold:

1. to justify the translation of a reaction network to a Boolean network,
2. to review the kind of information used to synthesise Boolean networks,
3. to review how existing methods use this information to synthesise Boolean networks.

In the following, I will thus give an overview of the formal modelling of biological systems. Of course, the selection of works presented in this chapter is somewhat arbitrary and not exhaustive. Indeed, it is difficult to define the “state-of-the-art” of systems biology, as the field is constantly evolving and there are many different subfields within it.

This chapter discusses many types of models. A model is a specific instance of a given *blueprint*. The term blueprint refers to a template which defines the common concepts and properties of the models within it. Many blueprints have been used to model biological systems. As mentioned in the general introduction, they can be categorised into two categories, namely mechanism-based and influence-based. We focus on the blueprint of reaction networks and Boolean networks, since they are of prime interest in this thesis. These blueprints are the respective epitomes of mechanism-based and influence-based models.

Each blueprint defines some parameters that have to be adjusted. A good model is a model parametrised such that it reproduces the idiosyncrasies of the system under study. More specifically, the construction of a model starts with information about the *structure* and the *dynamics* of the system under study. The model synthesis is defined as a parameter fitting task, which goal is to optimise a function that evaluates the merit of the candidate models. The optimisation itself is performed by an optimisation algorithm.

Once constructed, a model can be analysed. There are contexts in which the Boolean networks can be considered to be “better” than reaction networks. More generally, model analyses are usually considered simpler to perform on influence-based models rather than on mechanism-based models. Moreover, given two blueprints, it is quite natural to compare them, and to attempt converting models from one blueprint to another blueprint. This is what motivates the need to develop an automatic conversion from reaction networks to Boolean networks.

Outline Section 2.2 reviews outstanding applications of formal modelling of biological systems. Section 2.3 presents various model blueprints informally, but focuses on those of reaction networks and Boolean networks. Then, in section 2.4, we discuss the synthesis of such formal models. In section 2.5, we review the analyses one can perform on the constructed models as well as the formal relationship and conversion between models of different blueprints. Finally, in section 2.6, we summarise the chapter.

2.2 Success Stories in Systems Biology

So far, models have been used extensively to study various biological systems, at various scales, and of various organisms.

At a cellular level, we can mention models of the specific pathways involved in the control of cellular differentiation processes [Che+19b] (especially for T cells [Men06; Nal+10; Abo+14] and blood cells [Col+17]), the cell cycle in several kinds of yeasts (budding yeast *Saccharomyces cerevisiae* in [Kau+03], fission yeast in [DB08b], and *Candida albicans* yeast [Woo+21]) but also in humans. We can also study specifically the mechanisms that go wrong and lead to cancer

development [Coh+15] and other diseases. The map presented in [Ost+20] models the interaction mechanisms between the SARS-CoV-2 virus and its host.

At the organ level, there are for example models of the heart [Nob60; HN60], and the brain from the EPFL’s Blue Brain Project¹⁰ and the Human Brain Project¹¹. Models have been used to study tissues and organisms development. For example, [MÁ98; MTA99] model the morphogenesis of the flower *Arabidopsis thaliana*. A model published in 2014 shed light on finger formation during the embryonic development of animals [Ras+14]. Findings derived from this model confirmed hypotheses (the reaction-diffusion process) formulated as early as 1952 by Alan Turing [Tur52].

The work in [Kar+12] was driven by the ultimate goal of getting the *complete* picture of a living organism, namely *Mycoplasma pneumoniae*, which is a small bacterium. It is the first-ever comprehensive model of a whole cell. It includes all its 525 genes, gene products, and their interactions. It is organised into 28 submodels. Each submodel corresponds to a functional unit considered independent of the others. More recently, Szigeti et al. proposed to adapt the methodology used for *Mycoplasma pneumoniae* to build a human whole-cell model [Szi+18].

At an even larger scale, models have been applied to ecology (with predatory-prey system [Lot25; Vol26]), and epidemiology (through disease-spreading models such as in [Bea+22; Mun+09]).

Numerous other examples of models can be found in BioModels database [Mal+20], especially in the “Model of the Month” section. Several molecular systems biology models are also described on Nicolas Le Novère’s blog [Nov13]. Finally, a report from the Infrastructure for Systems Biology Europe (ISBE) highlights some success stories [ISB15]. By tracking down the story of the construction of models, this report particularly highlights the interplay between the modelling and the wet experiments which is, as mentioned in the general introduction (figure 1), at the heart of the approach of systems biology. Moreover, it highlights the three interconnected purposes that a model usually has:

1. compiling the knowledge one has about the system under study,
2. explaining observed phenomena,
3. predicting some behaviour based on changes in the input parameters.

The last item is particularly useful in pharmacology for drug discovery [BD17], response [Blo+18] and resistance [GSA18]. In this context, using models before starting wet experiments is safe, ethical, as well as money and time-saving.

Among all these models, we refer to those that aim at describing the detailed mechanisms as *mechanism-based models*. They usually represent the mechanisms as pools of objects transformed into other pools through processes. This representation maps very well with concepts from (bio-)chemistry (species, solutions, reactions and rates). Hence, mechanism-based models have been extensively used to study all kinds of systems at the (bio-)chemical level (metabolic or gene regulatory networks). However, they are not limited to this sole case-study [MFS22; Lot25; Vol26; DW05].

¹⁰ <https://www.epfl.ch/research/domains/bluebrain/>

¹¹ <https://www.humanbrainproject.eu/>

Influence-based models, however, abstract away details about the underlying mechanisms. They can thus be considered simpler than mechanism-based models. Yet, they are not simplistic, and like mechanism based-models, they have been used to study various systems at various scales. However, as we will see later, the difficulty and cost (time and money wise) of collecting data for constructing a mechanism-based model are usually higher than for the construction of an influence-based model. This is why it is quite rare for a mechanism-based model to be built exhaustively. Instead, these models usually focus on a subpart of the system, while influence models can be used for bigger systems.

2.3 A Dichotomous Zoo of Model Blueprints

A whole bunch of model blueprints has been proposed to study biological systems. The blueprints can roughly be categorised into two major classes, namely: mechanism-based (such as reaction networks) and influence-based (such as Boolean networks). Models from these two classes are in fact often used in parallel because they fulfil different needs. Indeed, they have different philosophies and ultimately different use-cases.

The review [Bar+20] is very interesting, as it presents how to model one biological system (the lactose metabolism of *Escherichia coli*) using different model blueprints (including differential equation system, Boolean network, Petri net, P-system, ...).

2.3.1 Blueprints of Mechanism-Based Models

The models in this category try to mimic, with a lot of details, the underlying mechanisms of the system under study. As mentioned earlier, they use concepts from (bio-)chemistry: Roughly speaking, the models in this category describe a system as sets of entities (chemical *species* in solutions) acted upon by processes (chemical *reactions*).

The construction of such models is a slow and intensive process, following a bottom-up approach. Once reconstructed, the overall structure of mechanism-based models is usually not likely to change much (other than adding more reactions), since the construction is directly based on biochemical evidence. However, the construction process particularly suffers from the paradox that the more and more data produced is often insufficient to create such detailed models. The construction of such a model requires significant and precise knowledge. Indeed, not only the structure (a reaction graph, see [definition 2.4.2](#) on page 59) but also the kinetic parameters and concentrations must be determined. Simple omics data is clearly not sufficient to determine all of this, and since the biochemical evidence that is required might be difficult and costly to obtain (time and money-wise), mechanism-based models usually focus on a small part of the system. Despite all the efforts, the parameters may not be measured exactly and have to be fitted to available data, following the procedure detailed in [section 2.4](#).

Outline In the following, we define the core concepts of mechanism-based models by focusing on reaction networks ([section 2.3.1.1](#)). Then we introduce several extensions of the core blueprint ([section 2.3.1.2](#)). Finally, we present several semantics with which mechanism-based models can be analysed ([section 2.3.1.3](#)).

2.3.1.1 Core Reaction Networks

The blueprint of reaction network models (see [definition 1.9.3](#)) is the epitome of mechanism-based models. The concepts used in core reaction networks are the base concepts used in the other models from this category.

Example 2.3.1 (Reaction networks modelling the enzymatic process). *In this example, the system in consideration is the simple enzymatic process. Several reaction network models can be used in parallel, depending on the desired level of detail and complexity, but the underlying mechanism remains the same: it consists of the conversion of a substrate S to a product P through the action of an enzyme E . Three possible models of the enzymatic process are depicted in [figure 2.1](#).*

Model 1 (\mathcal{R}_{enz}) is the one introduced in [example 1.9.4](#) on page 44. It is probably the most common model of the enzymatic process. It has four species $\mathcal{S} = \{S, E, C, P\}$ and the three reactions: reaction \mathcal{R}_{on} transforms a pair of a substrate S and an enzyme E to a complex C , reaction \mathcal{R}_{off} does the inverse, and reaction \mathcal{R}_{cat} transforms the complex C into the free enzyme E and two molecules of the product P . The reactions follow the mass action law kinetics [WG64] which states that the speed is directly proportional to the product of the concentrations of reactants. This modelling is based on some assumptions, such as well-stirred solutions, fast-enough diffusion, and constant temperature.

With additional assumptions (explicated in [example 2.5.2](#)), it is possible, while having some mathematical guarantees [Sri22], to simplify Model 1 into Model 2, where the components E and C have been removed. Still, E remains a modifier of the reaction as it appears in the kinetic law.

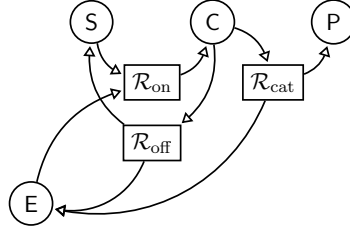
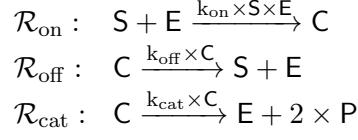
On the contrary, Model 3 is more detailed than Model 1, and closer to reality (yet still minimalistic). It uses two intermediary complexes C and C' that can be seen as the enzyme binding the substrate and the product, respectively. Of course, even more detailed models are possible, including more intermediary complexes.

2.3.1.2 Extended Reaction Networks and Related Formalisms

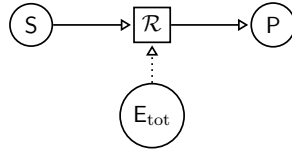
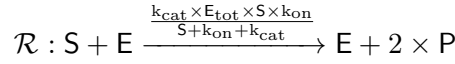
There exist several extensions of the core definition of reaction networks. These extensions add support for additional features. In particular:

Configurations in different spaces of values So far, the reaction networks were modeling changes of amount or concentrations of the species. But depending on the system under study, a reaction does not necessarily describe a change between configurations with positive values. For example, a reaction network can be used to model cell-to-cell communication through action potential. In this case, a configuration consists of continuous real values which are not necessarily positive.

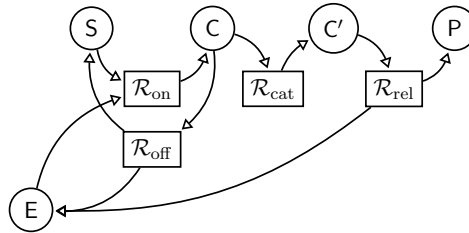
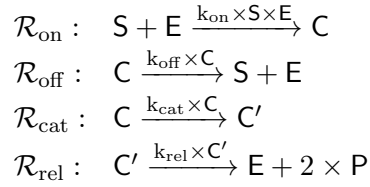
Conversaly, the formalism of reaction systems, introduced in [ER05], narrows the space of values so that a configuration does not track the multiplicity of the species, but only their presence. This impacts the semantics of inhibitors which are no longer considered to slow down a reaction, but to prevent it from happening. Ultimately, this substantially affects the dynamics of the model, as some configuration transitions will be inaccessible, unlike what can be observed otherwise.



(a) Model 1 (\mathcal{R}_{enz})



(b) Model 2



(c) Model 3

Figure 2.1: Reactions list and graph of three possible reaction models of the enzymatic process. Circle nodes represent species and rectangle nodes reactions. A plain edge from a species to a reaction (resp. a reaction to a species) indicates the species is a reactant (resp. product) of the reaction. A dotted edge from a species to a reaction indicates the species is a modifier of the reaction: it has an impact on the reaction rate, but its net stoichiometry remains unchanged.

Additional rules They define relationships among the variables of the model (species values, or parameters) that must hold at all times. Such rules are particularly useful when the exact reaction mechanism is not known. Three kinds of rules are usually distinguished: algebraic, assignment, and rate. We define them briefly in the table below, where x stands for a variable, f a numerical function, \mathbf{V} a vector of variables that does not include x , and \mathbf{W} a vector of variables that may include x .

Denomination	Description	General Form
Algebraic	Left-hand side is zero	$0 \doteq f(\mathbf{W})$
Assignment	Left-hand side is a scalar	$x \doteq f(\mathbf{V})$
Rate	Left-hand side is a rate-of-change	$\dot{x} \doteq f(\mathbf{W})$

Example 2.3.2. The model from [Nov+01]¹² describes the cell cycle of fission yeast and uses an assignment rule to set the value of a parameter σ from the amount of two species (`cdc13T` and `rum1T`) and a parameter (K_{diss}) along the entire simulation: $\sigma \doteq \text{cdc13T} + \text{rum1T} + K_{diss}$.

Compartments The system can be partitioned into multiple regions with limited communication. Typically, the species need to be in the same compartment to interact. P-system models, introduced in [Pău00], make use of such compartments.

Events They model instantaneous discontinuous changes in the dynamics of the system. An event defines the assignments to do when a condition becomes true. Conditions are usually represented as propositional formulas.

Example 2.3.3. The model from [Nov+01]¹³ describes the cell cycle of fission yeast and two events are used to reset the cell mass M (divide it by two) when MPF decreases below 0.1 at the end of the cycle:

	Condition	Assignment(s)
Event 1	$(MPF \leq 0.1) \wedge (flag_{MPF} \doteq 1)$	$M = M/2$ $flag_{MPF} = 0$
Event 2	$MPF > 0.1$	$flag_{MPF} = 1$

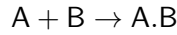
Additional details about local species arrangements In some systems (in particular in signalling cascades) some reactions might depend on some precise local arrangement of the components. With classic reaction networks, a species identifier refers to a pool of components in a specific arrangement, and a distinct identifier has thus to be used for each distinct arrangement of each component. The rule-based system κ (*kappa*) [Dan+07] and the *multi* extension of the SBML language [Zha+20] intend to avoid this. As for a reaction network model, each arrangement can be given an identifier, but in addition, an

¹² <https://www.ebi.ac.uk/biomodels/BIOMD0000000111>

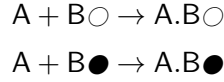
¹³ <https://www.ebi.ac.uk/biomodels/BIOMD0000000111>

identifier can also refer to several arrangements of the same component. A κ model is a list of interaction rules (instead of reaction rules). An interaction rule is very similar to a reaction rule, but since an identifier can correspond to several species identifiers, it thus corresponds to a (possibly infinite) set of reaction rules. The translation from an interaction rule to the corresponding reaction rules is done by expanding the identifiers by the possible correspondent species identifiers.

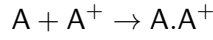
Example 2.3.4. Let A and B be two components. B can be in its phosphorylated or its unphosphorylated form (respectively noted $B\circ$ and $B\bullet$). A and B can bind, independently of the specific arrangement of B . In κ , this is modeled with a single interaction rule:



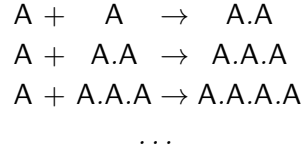
while in a classic reaction network, we would need the two following reactions, involving six identifiers instead of three:



Example 2.3.5. Let us consider a component A that can bind to whatever component made of A (denoted A^+). In κ , this is encoded by the following interaction rule:



The interaction rule expands to an infinite set of reaction rules, involving an infinite number of identifiers:



2.3.1.3 Semantics of Mechanism-based Models

A mechanism-based model (and a reaction network model in particular) defines a transition function on its set of configurations. Computing the transition graph of a mechanism-based model consists in computing how its configurations evolve according to the consumption and production of the species when triggering the reactions.

Let \mathcal{R} be a set of reactions, and $N = \biguplus_{(R_i, e_i, P_i) \in \mathcal{R}} R_i$ the multiset union of the reactants of the reactions in \mathcal{R} . N represents the minimal amount of reactant species needed to trigger the reactions in \mathcal{R} . We say a set \mathcal{R} of reactions is triggerable in a configuration C if C contains enough of the needed reactant species:

$$\forall X \in N : N(X) \leq C(X).$$

To compute the dynamics of the system (the transition graph), we have to rely on a *semantics*, which defines how exactly to interpret and execute a model. For example, the exact definition

(and meaning) of a configuration (especially the domain of the values in use) depends on the chosen semantics. Mechanism-based models are usually analysed with the following semantics: stochastic, discrete, Boolean, and differential.

Below, we illustrate how these semantics work by focusing on core reaction networks models (introduced in [section 2.3.1.1](#)), but the semantics can be adapted to extended reaction networks ([section 2.3.1.2](#)). Moreover, note that we assume the considered models are *complete* (all the kinetic parameters are given a value) because the stochastic and differential semantics can only be used with complete models.

Stochastic Semantics The model is interpreted as a continuous-time Markov chain (CTMC), which computes the *amount* of each species by triggering the reactions according to their speed, here interpreted as the number of reactions per time unit [AK11]. The classic simulation algorithm used is the Stochastic Simulation Algorithm (SSA) [Gil76].

Discrete Semantics The model is interpreted as a Petri net [Pet62; RML94], which computes the *amount* of each species, by triggering any possible triggerable set of reactions non-deterministically, and without taking their speed into account.

Boolean Semantics Both the precise amount and the kinetics are ignored, and the model is interpreted as a non-deterministic (asynchronous) transition system between Boolean configurations which abstracts the precise amount of the component as just “presence” and “absence”.

Example 2.3.6. *For a model composed of the following reaction: $A + B \rightarrow C$ the configurations $[A:1, B:1, C:0]$ and $[A:1, B:1, C:1]$ (A and B present, C don't care) are connected to the four following configurations: $[A:0, B:0, C:1]$, $[A:1, B:0, C:1]$, $[A:0, B:1, C:1]$, $[A:1, B:1, C:1]$ because C is for sure present, A and/or B might be fully consumed or not.*

Differential Semantics The model is interpreted as the following system of first-order ordinary differential equations (as presented in [section 1.9.2](#) on page 45):

$$\text{odes}(\mathcal{R}) =_{\text{def}} \bigwedge_{A \in S} \dot{A} \stackrel{\circ}{=} \sum_{(R_{\mathcal{R}}, e_{\mathcal{R}}, P_{\mathcal{R}}) \in \mathcal{R}} (P_{\mathcal{R}}(A) - R_{\mathcal{R}}(A)) \times e_{\mathcal{R}}$$

With this semantics, the rewriting defined by the different reactions is applied synchronously, and the values in the successive configurations are positive real numbers.

For extended reaction networks ([section 2.3.1.2](#)), the semantics are extended accordingly. In particular:

- The differential semantics of a reaction network complemented of algebraic and assignments rules will consist of a differential-algebraic system.
- A reaction network with events is typically analysed with methods and tools for hybrid systems [LZ05]. In the differential semantics, the time-continuous subsystem is modelled by differential equations as described previously, while the discrete events are modelled by finite state machines. The semantics of the events might differ depending on the exact

formalism and tool used. Some consider the events have to be triggered *as soon as* the guard condition becomes true, while some others consider an event simply *permits* the transition to be taken. During a simulation, the latter results in the transition to be taken at an arbitrary time after the guard becomes true, leading to a non-deterministic behaviour.

2.3.2 Blueprints of Influence-Based Models

The class of influence-based models consists of the model blueprints that focus on the relationships between the components of the system, abstracting the exact processes away. In this category, we find all the classic models from machine learning and statistics (such as regression models, and probabilistic graphical models), but also simple influence models and Boolean Networks (BN).

In contrast to mechanism-based models, influence-based models are usually simpler to create and to analyse. They are more broadly applicable, even at large scale (even genome-wide), and even when the system is not very well characterised. They relate more directly to the data than mechanism-based models, and new data can lead to significant rewiring of the model. They are close to the common reasoning schemes of biologists. Indeed, it is not rare for biologists to naturally abstract the exact reactions and focus directly on the influence among the components: “gene X is expressed in the absence of repressor Y ”. However, these models are not (necessarily) mechanistic, even though mechanistic knowledge can be included to constrain the dependencies of the components. A pleasing side effect of this additional knowledge is that it helps the construction of the model, as it reduces the search space.

A common representation for these models is the influence graph, presented in [definition 1.7.7](#) on page [39](#).

2.3.2.1 Correlation Models

A model from this category does not take the directionality of the edges of the influence graph into account, as it only works on the (partial) *correlations* between the species. With such models (Markov random fields models, for example), we can infer the conditional independence of two sets of species A and B given a third set of species C if all paths between the species in A and B are separated by a species in C .

2.3.2.2 Causal Probabilistic Models

A model from this category distinguishes causation from mere correlation by taking the directionality of the edges of the influence graph into account. In a Bayesian network for example, it is assumed that the values x_i of each species X_i depends on the values of its parents (denoted as $\text{parents}(X_i)$), and only on them. As a result, the probability of observing a configuration $[X_1 : x_1, \dots, X_n : x_n]$ can be computed by factorising the conditional probability of having observed each species X_i with value x_i :

$$P([X_1 : x_1, \dots, X_n : x_n]) = \prod_{i=1..n} P(X_i = x_i \mid \text{parents}(X_i)).$$

The blueprint of Bayesian networks was developed to study systems with discrete-valued and continuous-valued components. For discrete-valued variables, the conditional probabilities can be

represented as a table, which lists the probability that the child node takes on each of its different values for each combination of values of its parents. For continuous-valued variables, the most common distribution is the Gaussian distribution. For discrete nodes with continuous parents, we can use the logistic/softmax distribution.

Once constructed, Bayesian networks can be used to answer complex questions such as: “what is the probability to have X with value x given A has the value a and B the value b ?”.

2.3.2.3 Logical Automata Networks

A logic automata network of size n is composed of n components, referred to as *automata*. The status of each automaton is governed by its associated *transition function*, in which the influences between the automata are encoded using logics (such as Boolean logic, multivalued logic, or fuzzy logic).

The goal of this thesis is ultimately to produce a specific kind of such models, namely Boolean (automata) Networks (BNs). The recent review [Sch+20] is a gentle introduction to Boolean network models, and [section 1.8](#) on page 39 presents the concepts used in this thesis more in-depth. Boolean networks were originally introduced by Kauffman to model genetic regulatory networks [Kau69]. As presented in [section 1.8](#), the transition function associated to each automaton is a *Boolean* function computes the next status of the automaton (either 0 or 1). The meaning associated to the values 0 and 1 differs depending on the context: “absence/presence”, “inactive/active”, “below/above a threshold”.

In the literature, a Boolean network of size n is sometimes defined alternatively, as a function $\mathbb{B}^n \rightarrow \mathbb{B}^n$ [Mel+16]. Both definitions are actually equivalent. Indeed, a function $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ is the Cartesian product of n functions $f_1 \times \dots \times f_n$, and it comes equipped with natural projections so that, for a given f there are n functions $f_i : \mathbb{B}^n \rightarrow \mathbb{B}$ that factors through the projection on \mathbb{B} . In other words, each f_i is defined as the projection mapping of f on its i -th coordinate.

The simulation of a Boolean network returns a Boolean *transition graph* (see [definition 1.7.5](#) on page 39), similar to what is obtained with the Boolean semantics of reaction networks ([section 2.3.1.3](#) on page 55) except that this time, the updates are computed from the species themselves, without triggering any reaction. The order in which the species are updated depends on the chosen *update scheme*. There are an infinite number of them. In the *synchronous* update scheme [Kau69], the transition functions are applied all at once. However, as noticed by René Thomas [Tho91], the choice of the synchronous semantics is not always justified because “there is absolutely no reason why the time delays (...) should be equal”. He thus proposed the strict *asynchronous* scheme, where the transition functions are applied one by one. In the *general-asynchronous* update scheme, any number of species can be updated at each step. It is the most general classic update scheme in the world of automata networks, as it captures *all* the transitions allowed from all the other update schemes (including the synchronous and strict asynchronous update schemes).

The blueprint of qualitative networks is a refinement of the blueprint of Boolean networks, in which one can use more than two levels (and thus multivalued logic) to represent the status of the species. The blueprint of probabilistic Boolean networks allows several Boolean functions to be associated to each species. Each function can have a different probability of being triggered.

2.4 Model Synthesis as a Parameter Fitting Task

In this section, we are interested in how models are usually constructed. As mentioned in [section 2.3](#), models from each of the blueprints we presented have some parameters which must be fitted to the biological system under study. Examples of such parameters are: the kinetic parameters of a reaction network, joint probabilities of a probabilistic network, or the transition functions of a Boolean network.

For some influence models (Boolean networks in particular), the construction process can be made very simple by deciding on the structure (influence graph) and applying a *generic construction procedure*. For example, one can produce a Boolean network model from a given prior influence graph by using threshold functions encoding situations such as “X is activated if more of its activators than its inhibitors are activated” or “X is always activated, except if at least one of its inhibitors is activated”. The latter translates in propositional logics as $X \Leftrightarrow \neg(i_1 \vee i_j)$, where $i_1 \dots i_j$ are inhibitors for X. But in general, one needs to browse the space of all the possible models (or possible combinations of parameters values) and attempts to find the “best” one, that is, those that fit the best to both the knowledge and data we have on the system. In [section 2.4.1](#), we summarise what this knowledge and data classically are.

The task of searching for the best model corresponds to an *optimisation problem*. It has been widely studied in many domains because it is fundamental in many applications. Formally, the problem is expressed as :

$$m^* = \arg \text{optim}_{m \in M} f(m)$$

where M is the search space, and f is a scoring function that evaluates the merit of a candidate model m . Depending on the chosen scoring function, the value $f(m)$ has to be either maximised or minimised, and *optim* is changed to *min/max* accordingly.

Finding good combinations of parameter values is a hard problem, and there is an entire field of research dedicated to this. The principal reason for this problem to be so hard is that the search space for these parameters might be enormously large, and there is no hope of stumbling on a good combination of parameters by blindly twiddling the parameters. More clever search algorithms can be applied. We present them briefly in [section 2.4.3](#). In addition to the choice search algorithm, the scoring function f must be defined. We present some possible choices of scoring functions in [section 2.4.2](#).

All model construction methods roughly follow the same principles, but in the following, we focus on the construction of reaction networks and Boolean networks. [LAM19] is a recent comprehensive review focusing on parameters’ estimation in reaction networks, while [Hem+22] focuses on BN synthesis.

2.4.1 Input: Available Knowledge and Data

For a model to be a good model of a biological system, it has to comply with what is known of the system. The available knowledge and data about the system under study are thus of prime importance for the model construction step. It can be divided into two categories: the knowledge about the *structure* of the system (list of species of interest and how they interact with one another) and the data about the *dynamics* of the systems (how the species evolve over time). In [sections 2.4.1.1](#) and [2.4.1.2](#), we present these two categories and detail how it can be retrieved.

On top of these categories, modellers often use additional constraints for the sake of parsimony. This is presented in [section 2.4.1.3](#).

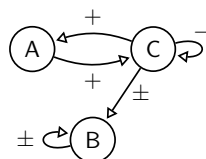
2.4.1.1 Structural Knowledge

Structure Graphs — Definitions

Structural knowledge concerns the species to take into account (which genes, proteins, ...) as well as their interactions. The most abstract form of structural knowledge is only about the *influences* between the species. An influence can be directed (parent-child relationship) to express the causation instead of mere correlation. Additionally, the influences can be labelled depending on whether a species favours or disfavors the presence of another species. This knowledge is classically represented with a Prior Influence Graph (PIG), also called prior knowledge network in the literature [Ost+16]. The definition of influence graph was given in [definition 1.7.7](#). What changes is only the meaning of what the influence graph encodes. In particular, given a prior influence graph $\mathcal{P} = (\mathcal{S}, E)$ and a species $X \in \mathcal{S}$, the parents of X are in fact the species that *potentially* influence X . Moreover, the signs on the edges are the *putative* polarity of the influences.

Example 2.4.1 (Prior influence graph). *Below is an example PIG for a system of three species. Note that the graph is the same as the one in [example 1.7.8](#) on page 39, but that the meaning of the edge changes slightly. For example, C and A are the potential parents of C.*

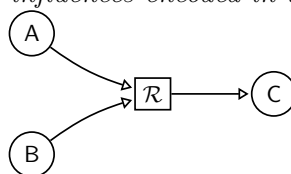
- A potentially *activates* C
- B potentially *interacts with itself*
- C potentially *activates* A
- C potentially *interacts with* B
- C potentially *inhibits* itself



A prior influence graph is the minimal necessary knowledge to build influence-based models. Mechanism-based models, however, need more detailed knowledge, because they need to know in which reactions the species are involved precisely. This information is given as a graph referred to as the Prior Reaction Graph (PRG).

Definition 2.4.2 (Prior reaction graph). *A reaction graph that breaks down a given mechanism as a directed bipartite graph with distinct nodes for reactions and species.*

Example 2.4.3. *Below is an example of PRG for a system of three species. In this PRG, A and B are involved as input (substrate) of a reaction noted \mathcal{R} which outputs some C. Note that this PRG is not meant to agree with the influences encoded in the PIG of [example 2.4.1](#).*



Structure Graph — Construction

We now discuss the construction of these graphs. We also discuss how to reduce the number of objects (species and/or reactions) taken into account. This is of prime interest because the more objects, and the more difficult will be the construction of the models and their analyses.

It is often the case that models are used to study mechanisms that are in fact continuous, such as chemical transformations, for example. As mentioned in the introduction, the study of the exact continuous (bio-)chemical processes is the realm of molecular dynamics. But in systems biology, modellers focus on a *finite* set of species. Typically, this set is chosen arbitrarily. It consists of the species considered to be the most important to model the system under study. One can use a predefined list of species that have been characterised in databases (list of genes, or proteins). Unfortunately, for a given organism, the number of such elements is often large. For example, the number of protein-coding genes in the human genome is 20 000 [Nur+22], and even the modelling of *Mycoplasma pneumoniae* that have 525 genes was already tedious [Kar+12].

Subset lists have been proposed. For example, the list L1000 was proposed as a highly representative subset of the human transcriptome [Sub+17]. It has been shown to be able to capture approximately 80% of the relationships found when measuring all transcripts directly. One can also decide to work only with transcription factors, because of their key role in the regulation of gene expression [Kru+11]. DoRothEA is a curated collection of transcription factors for human and mouse [Gar+19]. If we aim to model a specific pathway, it is possible to use pathway databases as a starting point [Wix01] (such as KEGG [Kan00], Reactome [Gil+22], Pathway Commons [Rod+20], WikiPathway [Mar+21]).

These databases aim at being exhaustive. Yet, the *core* of the mechanism is often governed by a small number of species. For example, models about yeast cell cycle often contain less than a dozen of species [Nov+01; DB08b], while around 800 genes are involved in the process [Spe+98]. Yet, such reduction is most of the time only doable through expert knowledge, which is not directly accessible in databases. One way that is sometimes exploited, thought, is to use an existing reduced model of the pathways of interest for another organism, and to start the model construction from the equivalent species (homologue or analogue) in the organism of interest.

As we just saw, the list of important species can be defined a priori, but it can also be reduced a posteriori. Two common approaches exist. They are mostly applied to influence-based models and rely on dynamic information. First, one can simply remove the species which are always observed constant. Their presence increases the model search space, while they are not really necessary to explain the dynamical data at hand (since the data can be reproduced by simply keeping such species constant). Hence, they can be safely removed. This approach is for example used in [Che22]. Second, an easy trick to reduce the number of species is to merge together those with similar dynamics. Indeed, such species are often co-regulated. This approach is for example in use in [Hua+03]. In [Mar+07], the clusters are found using the k -means algorithm, but another study reports strong agreement ($\sim 80\%$) when the clustering is done with other methods such as hierarchical clustering and self-organising maps [MDW04].

Once the list of species of interest has been defined, they are, by default, considered to all influence one another. However, one can decipher how the species influence each other more precisely, through analyses such as correlation analyses on dynamical data. This is enough to build a prior influence graph, but not to build a prior reaction graph, though. Indeed, the latter is more complicated to obtain, as more precise knowledge (in particular, molecular binding evidence) is required to decipher the precise mechanisms [Fei+09]. If the system under study is known well enough, one can also rely on the information from the pathway databases mentioned before.

2.4.1.2 Dynamical Data

Presentation of the Data

Dynamical data can take various forms. Three common forms are: lists of input/output configurations (used in [AD22; AMK00], for example), sequences of configurations, and multivariate timeseries (used in [Ost+16] for example).

Definition 2.4.4 (Sequence of configurations). *It is a sequence of configurations indexed by discrete time (which may correspond to a given continuous time). A sequence of configurations is said to be deduplicated if all adjacent configurations C and C' are different.*

Example 2.4.5. $[A:0, B:1, C:0] \rightarrow [A:0, B:1, C:1] \rightarrow [A:1, B:0, C:0] \rightarrow [A:0, B:0, C:1]$ is a deduplicated sequence of Boolean configurations for a system with three components.

Definition 2.4.6 (Timeseries, binarised timeseries). *A (multivariate) timeseries is a list of values indexed by both time and species name. Each entry thus represents the value of one species at a given time point. The value can represent various things, such as a concentration or an activity, depending on the context. A timeseries is a binarised timeseries if it only takes values in $\mathbb{B} = \{0, 1\}$.*

Example 2.4.7. An example of a multivariate timeseries is given below. The continuous concentrations of three species (A, B and C) have been sampled for 20 timesteps. Here, all the observations range from 0 to 100.

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	0	3	7	13	20	30	49	61	100	63	36	25	2	3	1	1	3	0	0	0
B	100	86	64	57	54	53	51	49	45	37	33	28	22	19	14	12	9	5	2	0
C	0	27	36	42	60	75	54	44	38	48	60	72	88	90	100	100	100	100	100	100

The resulting binarised timeseries with a threshold of 50 is shown below, where the colors blue and red encode respectively for “below” and “above” the threshold.

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	blue	blue	blue	blue	blue	blue	blue	red	red	red	blue	blue	blue	blue	blue	blue	blue	blue	blue	blue
B	red	red	red	red	red	red	red	red	red	red	blue	blue	blue	blue	blue	blue	blue	blue	blue	blue
C	blue	blue	blue	blue	red	red	red	red	red	red	red	red	red	red	red	red	red	red	red	red

Here there are four configurations ($[A:0, B:1, C:0]$, $[A:0, B:1, C:1]$, $[A:1, B:0, C:0]$, $[A:0, B:0, C:1]$) lasting respectively 4, 3, 3 and 10 timesteps. Vertical bars indicate a change of configuration.

The most common form of dynamic data is a multivariate timeseries. It can be obtained from direct measurements of the concentrations/activities of the components over time, or be accessed from databases. The exact *preprocessing* of the data (noise reduction, normalisation, binarisation...) heavily depends on the technology used. When the technology cannot provide several measures from the same sample (for example, single-cell measurement technics involve the destruction of the cells), computational approaches have been developed to reconstruct timeseries out of it [Che+19a].

Furthermore, timeseries bears a specific issue, namely the *sampling rate*, that is, the number of data points that are collected per unit of time. Indeed, it is not possible to keep track of

the concentrations/activities of the components at all points in time. An important task when designing an experiment is thus to determine the lowest sampling rate that would not miss key information. The sampling rate can be non-uniform, and adapt to the different timescales of the mechanisms under study. It is usually higher at the beginning of an experiment than towards the end, because it is usually assumed that the system reaches a steady-state.

Finally, various elements can have an impact on the dynamics of a system under study. Data may for example differ for different experimental conditions, such as temperature, and pH, *in vitro* and *in vivo* experiments, and for different strains or mutants. The data are sometimes associated with an experimental context, corresponding to several experimental replicates, or different perturbation experiments (different stimuli, or knockout experiments).

Extraction of Dynamical Patterns

Modelling studies rely on dynamical patterns to construct and validate models. These patterns can be extracted through the analysis of timeseries data.

First, given a timeseries, one has direct access to the *successive configurations* of the system (i.e., the successive values taken by the components over time). When data is missing (for example, because of a not well-adapted sampling rate), one can focus on the *reachability* of the configurations, instead of the exact transitions. This is what the method **Caspo-TS** focuses on to synthesise Boolean networks [Ost+16]. Intuitively, this corresponds to the introduction of a wildcard configuration between each pair of successive configurations in the sequence. For example, the configuration sequence from [example 2.4.5](#) would become $[A:0, B:1, C:0] \rightarrow * \rightarrow [A:0, B:1, C:1] \rightarrow * \rightarrow [A:1, B:0, C:0] \rightarrow * \rightarrow [A:0, B:0, C:1]$. Each wildcard can then be expanded to any sequence of configurations.

Other key patterns concern the long-term (asymptotic) behaviour of the system. In particular, the notion of *attractor* is a central concept in the study of dynamical systems. Informally, an attractor is a periodic sequence of configurations in which the dynamical system loops. When the attractor is reduced to one configuration, it is called a *fixed-point attractor*. Otherwise, it is called a *cyclic attractor*. A lot of biological systems have been found to be driven by attractors. For example, during the cell division cycle, a cell awaits in a specific configuration (G0) until it receives a division signal [Li+04]. Consequently, the cell goes into a sequence of configurations (referred to as G1, S, G2, M). Once the division process is done (configuration M is reached), each daughter cell awaits again in the configuration G0.

Other dynamical patterns sometimes taken into consideration, involve so-called confined dynamics (when some components of the system stop evolving after some time, which thus guarantees that the attractors will also inherit of this value) and bifurcation (when a small change in the system results in strong changes in its behaviour). In her thesis, Chevalier makes use of such dynamical patterns adapted to the study of cell differentiation to build Boolean networks [Che22].

2.4.1.3 Additional Constraints

Additional constraints can be added to reduce the search space further. Such constraints are particularly popular for determining the kinetic parameters of mechanism-based models. In this context, the constraints usually correspond to physicochemical knowledge, such as enzymatic capacity and reaction directionality.

For influence-based models, we already mentioned structural constraints used to reflect the

known influences between the species. These constraints are already quite helpful in reducing the search space. In addition, two additional kinds of constraints are commonly used: *monotony* and *minimality*. In the literature, such constraints are justified by the parsimony principle, which states that the simplest solutions are the best ones. We now illustrate the minimality and monotony constraints by focusing on Boolean network synthesis. These concepts were formally introduced in [sections 1.6.3](#) and [1.6.4](#) respectively.

Minimality All the methods I am aware of claim to synthesise the simplest models. In the case of Boolean network synthesis, this usually translates into synthesising transition functions represented as minimal Boolean expressions, or as truth tables with the smallest number of inputs. To achieve minimality, some tools directly hardcode the minimality constraint, while some others generate functions that are later minimised using dedicated external tools. For example, in [MDW04], Martin et al. generate truth tables which are subsequently minimised (approximately) with the Espresso tool [Bra+82].

In theory, the number k of inputs for the transition function of each species should be unbounded, and might even be equal to the number n of species considered. But for computational complexity reasons (there are 2^{2^k} Boolean functions with k inputs), several approaches propose to limit k . In particular, $k = 5$ in [Mar+07], but MIBNI opts for $k = 10$ [BK17], while REVEAL [LFS98], and Best-Fit [LSY03] recommend $k = 3$, and BIBN considers k only up to 2 [Han+14]. Most of the time, these limitations are justified by the fact that in biology, the number of regulators (parents) for a component is usually small. Indeed, several studies have found that the number of regulators for components of gene regulatory networks follows a power law x^{-y} with $y \geq 2$ [BO04]. In other words, except for some hub components, most of the components have a small number of regulators. In [figure 2.2](#), we plot the log of the power law distribution for several values of the parameter y .

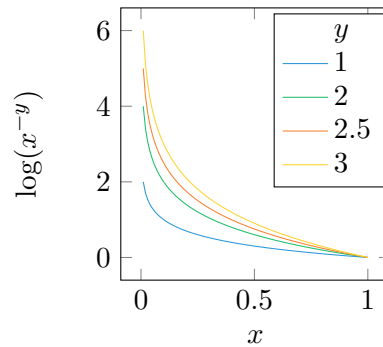


Figure 2.2: Power law distribution.

Monotony In biology, a given component is usually considered as either an activator or an inhibitor of another species [Alo19, p. 12]. When the Boolean network synthesis is constrained by the monotony of the transition functions, it means that even if the given prior influence graph specifies several possible signs for an influence, at most one will be effectively used in the constructed model. Examples of Boolean network synthesis methods that consider the monotony of the influences as a hard constraint are [Ost+16; Che22]. Such hard monotony constraints drastically reduce the search space, as presented in [section 1.6.4](#).

To the best of my knowledge, minimality is not questioned, except when it is achieved by hard-bounding k with a very small value. The same cannot be said for monotony. Indeed, some authors advocate for the consideration of non-monotone Boolean networks since they identified some biological systems where monotony does not seem justified, in particular genetic regulation systems [NRS11].

2.4.2 Scoring Functions

As presented in the previous section, the available information on the system under study typically involves structural knowledge and dynamic data. Assuming they are correct, the best models are the ones that reflect them the best.

Various measures have been used to quantify the merit of a candidate model toward given knowledge and data [LMY12].

Several model construction methods use metrics founded on information theory [Sha48]. Mutual information criterion is very commonly used by methods that attempt to find, for each species X , a set P of species such that the species in P explain fully the known behaviour of X . For example, ARACNE [Mar+06], which reconstructs backbone networks from expression profiles, and REVEAL [LFS98], NNBNI [BK20], and MIBNI [BK17], which construct Boolean network models, are all using mutual information. Their goal is to find the set P by maximizing the mutual information $M(X, P)$ between X and the species in P . Classically, the methods iterate on each species N (not yet in the set P) and adds it to P if it improves $M(X, P \cup \{N\})$, or equivalently, if it reduces $H(X|P \cup \{N\})$.

Various formulas for M can be derived from entropy, including

$$M(X, P) = H(P) - H(X|P),$$

where $H(P)$ is the individual entropy of the components in P , and $H(X|P)$ is the conditional entropy. The latter measure the information about X that is still missing when we keep track of the components in P . Figure 2.3 is a visual description of the relationships of these quantities.

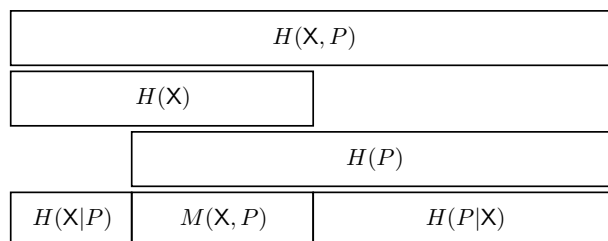


Figure 2.3: How the mutual information relate to the join entropy, individual entropies, and conditional entropies. The figure is adapted from [Mac03].

Alternatively, one can use the maximum likelihood estimation to find the model that, given the data, maximise the likelihood function [CL12]. In order to balance the likelihood with the parsimony of the candidate model, other methods use measures that include an extra penalty term for the size of the model. Common metrics in this category are the Akaike information criterion (AIC) [Aka74], used, for example, in [Gui+13; PM14] and Bayesian information criterion

(BIC) [Sch78] used, for example, in [MM06].

The metrics mentioned so far only quantify the relationship between variables. Conversely, correlation measures distinguish between more specific relationships as they account for the *signs* of the relationships. For instance, Pearson correlation metric is used in [Mau+11] to synthesise Boolean network models.

Finally, let us mention the Root-Mean-Square Error (RMSE), Mean Square Error (MSE), and r^2 score (a normalised version of MSE), which quantify the differences between predicted and observed continuous values. They are popular metrics classically used for fitting regression models (continuous values), but they were also adapted for the construction of other models. For example, the method **Caspo-TS** uses the MSE to evaluate how well the dynamics of the Boolean networks it synthesises fit the values in the provided timeseries [Ost+16].

2.4.3 Search Algorithms

Many possible search strategies can be used to find models with good parameters values. The strategy must be carefully chosen, especially when the search space is large (and it often is). A few examples of large search spaces are:

- the reaction kinetic parameters are continuous values which can range over different orders of magnitude (see Table 2.2 in [Alo19]),
- the structure of a Bayesian network is an acyclic graph of size n , but the number of such graphs is exponential in n ,
- there are 2^{2^n} Boolean functions of n nodes, and thus much more possible minimal DNF expressions.

Outline In the following, we describe exhaustive, local, and global search strategies. They are presented in [sections 2.4.3.1 to 2.4.3.3](#) respectively. One can incorporate additional constraints to limit the search space. This is presented in [section 2.4.1.3](#). Finally, [section 2.4.3.4](#) details Bayesian inference, where the search strategies are informed about the *a priori* location of the best parameter values.

2.4.3.1 Exhaustive Search Strategies

Exhaustive search strategies consist in testing all the possible candidates. This ensures finding the global best candidates, but it is often considered intractable if the search space is too large.

In recent years, a lot of effort has been made to propose clever exhaustive search strategies, in particular when the search space is finite. This was probably driven by the desire to solve hard combinatorial search problems such as the satisfiability problem, which is of prime importance in many domains. Many of these clever approaches use the divide-and-conquer principle.

Among them are the branch-and-bound-based algorithms. Such an algorithm constructs a tree whose branches represent the possible decisions (parameter assignments) that can be made at each step, and whose leaves represent the possible candidate models (combination of parameter values). The algorithm explores the tree recursively. At a given node, it estimates what would be

the score of the solutions contained in the node. With this information, it can prune the branches that contain non-promising candidates.

Algorithms inspired by SAT-solvers, such as Conflict-Driven Clause Learning (CDCL)-like algorithms, can be seen as a very specific variant of the branch-and-bound algorithm. The branches represent the possible assignments of the variables of a logic formula which encodes the specifications of the problem at hand. The task of the underlying solver is to find the variables' assignments that make the formula evaluate to 1 (True). If the formula evaluates to 0 (False) after having tried an assignment, CDCL backjumps and tries a different value for a variable. It also learns from the encountered conflict by updating the formula to keep track of the failed assignment. This constrains the future decisions, and helps to avoid the same conflict in the future.

Formal verification tools can be used to find appropriate parameters values. For example, in [Cal+06], the tool **Biocham** [CFS06] was used to describe a reaction network model of the cell cycle, and temporal logic [Cla+05] was used to formally encode the properties of the system and validate the model relatively to the available data.

2.4.3.2 Local Search Strategies

Local search strategies aim at finding the lowest value obtainable in the *neighbourhood* of the starting point. These strategies often have fast convergence to a minimum, that might not be the global minimum.

Gradient-based methods can be used when the gradient of the objective function can be computed. These methods traverse the search space by following the path where the gradient vanishes.

2.4.3.3 Global Search Strategies

A global search aims to overcome the “local minima trap” from which local search strategies suffer. To do so, they use a stochastic sampling of candidates across the entire search space.

Simulated annealing is a slightly educated random walk [LA87]. It was adapted by Kirkpatrick et al. in the eighties for function minimisation. The term “annealing” comes from an analogy made between the search for the minimum value of the objective function and the process of heating-and-cooling used in metallurgy to find stable configurations of the atoms of a material. The algorithm uses a particular schedule fixed by a heating-and-cooling parameter to jump over the solution space, without taking into consideration where it landed before. Simulated annealing is for example used by the method **ATEN** to construct Boolean networks [Shi+20].

Evolutionary algorithms work on a population of candidate models. At each iteration, they select a subset of candidate models based on their merit (fitness), and produce a new population of candidate models based on them. Genetic algorithms are the most common type of evolutionary algorithms [SP94]. The methods **GABNI** [BK18], **CGA-BNI** [TK21], **SgpNet** [Gao+22], and **GAPORE** [Liu+21] all make use of a genetic algorithm to build Boolean networks.

2.4.3.4 Bayesian Inference

By default, the search strategies presented before are not tailored to use prior belief about where the best models are in the search space. On the contrary, Bayesian inference allows for an *explicit* incorporation of such prior belief [Wil07].

Using Bayes' theorem, Bayesian inference combines the prior probability distribution (the initial belief about the models), the data, and the likelihood function (which represents the probability of an observation given the data) to update incrementally the probability distribution. Bayes' theorem states that the posterior probability (the conditional probability of an event A occurring given that event B has occurred) is proportional to the product of the likelihood (the conditional probability of an event B occurring given that event A has occurred) and the prior probability (the probability of A before any additional information is taken into account). It is formulated as

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}$$

or, in more mathematical terms,

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}.$$

In place of the denominator, the marginal likelihood $P(B)$ of the event B occurring (also called marginal evidence in the literature) acts as a normalising constant that ensures the result is a valid probability distribution (that the posterior adds up to 1). It can be computed by summing up the numerator over all possible values of B .

Bayesian inference has for example been used to construct Boolean networks in [Han+14], and to fit the values of reaction networks in [Jia+22; WBS19].

2.5 Model Analyses and Transformations

2.5.1 Structure Analyses

Various qualitative analyses can be performed as soon as the structure of a model is known. For reaction networks, let us cite flux balance analysis [VP94], elementary mode analysis [Sch99], extreme pathway analysis [SLP00], and analysis from chemical organization theory [DD07; KRD09]. For a Boolean network, it is for example possible to determine the number of fixed points by simply looking at whether its influence graph contains cycles or not [Tho81; Ric19].

2.5.2 Dynamic Analyses

One of the main research trends in dynamical systems is the study of their dynamics. This includes tasks such as retrieving the time course of the system [Fer+19], finding the attractors that are reached ultimately (as mentioned in [section 2.4.1.2](#), they are often given a biological interpretation), checking the adaptation of the system to various perturbations, and control [Bia18]. Other studies focus on deciding if the system can present a particular dynamical behaviour [Sal13a] [Sal13b], checking the complexity of the behaviours obtainable with limited resources [EMR11],

finding bifurcation in the dynamics [Ben+22], or computing the probability for a system to reach a halting state [EMR10], which is particularly useful when studying cancer [Uth21].

The full potential of mechanism-based models is only unlocked when they are complete, i.e., when all the parameters (initial concentrations and reaction rates, but also diffusion coefficients for spatial model) are given a value (definition 1.9.5 on page 45). Among the analyses that are not possible to apply to mechanism-based models with unknown parameters, we can mention their simulation with the differential and stochastic semantics. However, motivated by the fact that the essential dynamical properties of a biological system are generally robust against moderate changes in the parameters value, technics such as parameter sensitivity analyses can be used to find the invariant properties [DR06].

Various complexity results showed that, thanks to their discrete and finite nature, the dynamics of Boolean network models are simpler to analyse, especially for large models.

2.5.3 Model Simplification

The complexity of a model is generally defined in terms of the number of objects involved (components, reactions, ...). Like other modelling approaches, reaction network models and Boolean network models are subject to simplification. Any reduction in model complexity is generally desirable because the more objects are involved, the more the model is difficult to understand and analyse. The time that an algorithm takes to identify attractors of a Boolean network of size n by exhaustive search in its transition graph is $\mathcal{O}(2^n)$. Therefore, for large Boolean networks, it quickly becomes unfeasible, hence, the interest in simplification techniques to reduce the number of species. Indeed, the number of configurations in the transition graph (hence the complexity) is halved for each species removed.

Several methods have been proposed to simplify Boolean network and reaction network models, while maintaining some of their properties. See, for example, [Ric05; SAR13; Vel+14; PV22; Vel11; Nal+11; ZA13; YYC19] for studies about the simplification of Boolean networks and [OM98; Rad+12; Val+06; Mad+16; Cho+08] for studies about the simplification of reaction networks.

In the following, we just illustrate how much simpler the simplification process is on a Boolean network than on a reaction network. We use a reaction network and a Boolean network of the enzymatic process, shown respectively in figure 2.1 on page 52 (Model 1) and example 2.5.1. In these models, it is quite natural to seek to eliminate the enzyme in both its free form E and its bounded form C.

In the Boolean network case, removing a component generally consists in attributing the effect of its parents directly to its children. Some components are particularly straightforward to remove, such as those without a child or always constant.

Example 2.5.1 (Boolean network simplification). *We start from the following Boolean network:*

$$\begin{aligned}\mathcal{B}_1 = & \left(S_{\text{next}} \Leftrightarrow 1 \right) \\ & \wedge \left(E_{\text{next}} \Leftrightarrow 1 \right) \\ & \wedge \left(C_{\text{next}} \Leftrightarrow S \wedge E \right) \\ & \wedge \left(P_{\text{next}} \Leftrightarrow C \right)\end{aligned}$$

which can be simplified to:

$$\begin{aligned}\mathcal{B}_2 = & \left(S_{\text{next}} \Leftrightarrow 1 \right) \\ & \wedge \left(P_{\text{next}} \Leftrightarrow S \right)\end{aligned}$$

where the variable E has been removed and C was directly substituted by the result of its transition function.

In the reaction network case, the simplification usually exploits a bit of algebra, the law of mass conservation, as well as the different time scales of reaction speed between the reactions [Gun14c]. The latter means that we state that the speed of change of a slow variable is null (and hence the variable stays constant), and we assume the fast variables to be in steady-state. This corresponds to the necessary conditions for the quasi-steady-state approximation [Pel+21]. This type of simplification has also been proposed in the stochastic case [RA03; Don+07; KJB14] (where it is particularly useful to speed up the simulation with the Gillespie algorithm [Aga+12]).

Example 2.5.2 (Reaction network simplification). *We start from the following ODE system, reproduced from [example 1.9.8](#):*

$$\dot{S} \doteq -k_{\text{on}} \times E \times S + k_{\text{off}} \times C \quad (2.1)$$

$$\wedge \dot{E} \doteq -k_{\text{on}} \times E \times S + k_{\text{off}} \times C + k_{\text{cat}} \times C \quad (2.2)$$

$$\wedge \dot{C} \doteq k_{\text{on}} \times E \times S - k_{\text{off}} \times C - k_{\text{cat}} \times C \quad (2.3)$$

$$\wedge \dot{P} \doteq 2 \times k_{\text{cat}} \times C \quad (2.4)$$

$$(2.5)$$

where

$$k_{\text{on}} = 10^6 \text{ L mol}^{-1} \text{ s}^{-1}$$

$$k_{\text{off}} = 0.2 \text{ s}^{-1}$$

$$k_{\text{cat}} = 0.1 \text{ s}^{-1}$$

Here, C is a fast variable (it rises rapidly to its maximum value) while S and P are slow variables (they respectively decrease and increase slowly). We thus assume C is constant (its speed of

formation equals its speed of degradation)

$$\begin{aligned}
 C &= \text{constant} \\
 \Leftrightarrow \dot{C} &= 0 \\
 \Leftrightarrow k_{\text{on}} \times E \times S &= k_{\text{off}} \times C + k_{\text{cat}} \times C
 \end{aligned} \tag{2.6}$$

Now, what we want is to remove the variable E from [equation \(2.6\)](#). The conservation of mass applies on the enzyme (which is, at all times, either in its free form E or bounded form C). Indeed, adding [equation \(2.2\)](#) and [equation \(2.3\)](#), we obtain that, at all time:

$$\begin{aligned}
 \dot{E} + \dot{C} &= 0 \\
 \Leftrightarrow E + C &= \text{constant} = E_{\text{tot}} \\
 \Leftrightarrow E &= E_{\text{tot}} - C.
 \end{aligned}$$

Therefore, in [equation \(2.6\)](#):

$$\begin{aligned}
 k_{\text{on}} \times (E_{\text{tot}} - C) \times S &= k_{\text{off}} \times C + k_{\text{cat}} \times C \\
 \Leftrightarrow k_{\text{on}} \times E_{\text{tot}} \times S - k_{\text{on}} \times C \times S &= k_{\text{off}} \times C + k_{\text{cat}} \times C && (\text{left side developed}) \\
 \Leftrightarrow k_{\text{on}} \times E_{\text{tot}} \times S = k_{\text{on}} \times C \times S + k_{\text{off}} \times C + k_{\text{cat}} \times C && (\text{sent } -k_{\text{on}} \times C \times S \text{ on the left}) \\
 \Leftrightarrow k_{\text{on}} \times E_{\text{tot}} \times S = C \times (k_{\text{on}} \times S + k_{\text{off}} + k_{\text{cat}}) && (\text{right side factorised}) \\
 \Leftrightarrow E_{\text{tot}} \times S = C \times \left(S + \frac{k_{\text{off}} + k_{\text{cat}}}{k_{\text{on}}} \right) && (k_{\text{on}} \text{ sent on the other side}) \\
 \Leftrightarrow E_{\text{tot}} \times S = C \times (S + K_M) && (\text{posing } \frac{k_{\text{off}} + k_{\text{cat}}}{k_{\text{on}}} = K_M) \\
 \Leftrightarrow C = \frac{E_{\text{tot}} \times S}{S + K_M} && (\text{isolate } C) \tag{2.7}
 \end{aligned}$$

From [equation \(2.4\)](#), we have that

$$\dot{P} = 2 \times k_{\text{cat}} \times C$$

If we replace C by what we obtained in [equation \(2.7\)](#), we have that

$$\dot{P} = \frac{2 \times k_{\text{cat}} \times E_{\text{tot}} \times S}{S + K_M}$$

This equation corresponds to the so-called Michaelis-Menten kinetics [MM13b] (see [PG51; Mic+11; MM13a] for English translations). The product $k_{\text{cat}} \times E_{\text{tot}}$ is often abbreviated V_{max} because it is the maximal speed of creation of the product (when all the enzyme is saturated). K_M is a characteristic constant for a given enzyme and substrat. It gives the value of S for which the speed of the reaction is half V_{max} . It is indirectly related to the affinity for E for S : a low (resp. high) K_M value reflects the high (resp low) affinity of E for S . One can interpret V_{max}/K_M as an effective rate of capture of S by E [Nor98]. This kinetics also models a more realistic, yet still minimalistic, model of the enzymatic reaction shown in [figure 2.1](#) (Model 3). Of course, more detailed models are possible, including more intermediary complexes.

The Michaelis-Menten simplification of the reaction network for the enzymatic process is a

small example. Yet, it has terrified biochemistry students in spades (including myself). In contrast, the process of simplification of influence-based models (and Boolean networks in particular) is considerably simpler to grasp.

2.5.4 Aggregation of Models

On the opposite of model simplification, it can be useful to aggregate different models when they model different parts of the same biological system (recall the model of *Mycoplasma pneumoniae* mentioned in [section 2.2](#) on page 48 [Kar+12]).

Aggregated models can be constructed from several mechanism-based models [Poo+22]. Dedicated tools (such as MFO or SBML Harvester) have been proposed to merge reaction networks encoded in the Systems Biology Markup Language (SBML) and annotated using ontologies. Unfortunately, not all SBML models have sufficient annotations for these tools to work. In [Chi+15], the authors investigate the composition of reaction network models with heterogeneous semantics. In any case, one needs to be careful to check if the parameters were fitted for similar physicochemical conditions.

The aggregation of influence-based models is much easier. For example, one can simply merge several Boolean network models by constructing a probabilistic Boolean network model, where each species is associated with several transition functions (one from each initial model) that are triggered with a given probability.

2.5.5 Conversion Between Models

Several models are often used in parallel to study a given biological system. Indeed, as we saw in the previous sections, the different models have their own strengths and weaknesses. In particular, we saw how different is the respective philosophy of influence-based models (such as Boolean networks) and mechanism-based models (such as reaction networks). The latter are more detailed than the formers, but their construction and analysis is more difficult, especially when the tasks at hand are related to model checking, control, or model aggregation.

The formal relationship and conversion between various models have been addressed in several studies. For example, Li et al. compare probabilistic Boolean networks and dynamic Bayesian networks in terms of inference accuracy from timeseries data [Li+07], and Lähdesmäki et al. discuss their formal relationship in terms of joint probability distribution [Läh+06]. Krumsiek et al. explain how to convert a Boolean network model to ODEs [Kru+10], Barbuti et al. show how to use a specific kind of reaction network model (namely the blueprint of reaction systems, which ignores stoichiometry and kinetics) as an alternative way of exploring the dynamics of threshold Boolean networks [BGM21; Bar+21]. Finally, let us mention Fages et al. who show that an influence-based model with forces (similar to [definition 1.7.7](#) but with a term to value the force, and where inhibitors are not considered to prevent reactions from happening) can be simulated by a core reaction network with either the stochastic, discrete, Boolean or differential semantics, but that the converse is not true, except under the differential semantics where both have the same expressive power [Fag+16].

Regarding the different semantics of core reaction networks ([section 2.3.1.3](#)), they correspond to different abstraction (details) levels [FS08a]. They compute different transition graphs,

and different aspects of the system can be studied [CFS05; CFS06]. The formal analysis and comparison of the semantics (and of the transitions they lead to) was performed by Fages et al. in [FS08a] by applying the principles of abstract interpretation. This framework was originally introduced for the analysis of programs [CC77]. It consists of approximating the computation of a program using computation with finite-lattice intervals. Fages et al. show how the stochastic, discrete and Boolean semantics of a core reaction network are in fact successive hierarchical abstractions [FS08a]. As a result, given two semantics, the absence of a behaviour in the most abstract one entails its absence in the most concrete semantics, but the converse is not true. For example, a behaviour which is not possible in the Boolean semantics, is also not possible in the stochastic semantics, whatever the kinetic expressions are. The differential semantics, however, is not connected in the same way. This is because it does not really “behave” the same as the other semantics. Indeed, with the differential semantics, all the reactions are triggered in parallel, and the values assigned to the species are positive reals computed deterministically. It has been shown that the differential semantics is in fact a continuous and deterministic approximation of the mean stochastic behaviour for a large number of molecules [Kur70; DN08]. In other words, it computes the *average* number of copies of the species in an infinite number of stochastic simulations. The connections are depicted in [figure 2.4](#).

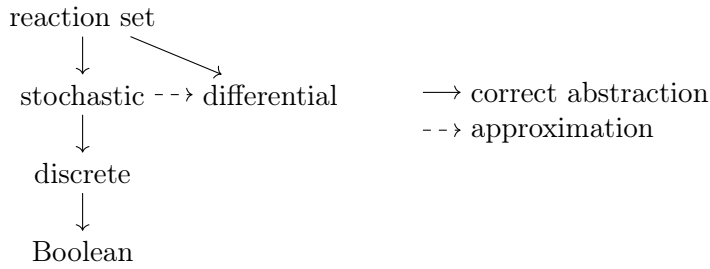


Figure 2.4: Formal relationships between reaction networks and their stochastic, discrete, Boolean, and differential semantics.

Qualitative abstraction of the continuous dynamics of various models has drawn strong interest over time, particularly in the Boolean domain. It is used to simulate the model abstractly [NVV22; DB08a], explore the possible behaviours of a reaction network [FS08a], and study the complexity of dynamical analyses [FMP14; Den+15b; Den+15a; Den+19]. Such Boolean abstractions include the classic Boolean semantics of reaction networks presented previously, but also others we introduce hereafter.

In [DB08a], the coarse-grain interpretation of a differential-algebraic system (normalised between 0 and 1) leads to a partial function from $\mathbb{B}^n \rightarrow \mathbb{B}^n$. This function is similar to a Boolean network such as introduced in [section 1.8](#) on page 39, but without the specification of a local transition function for each species. The conversion was successfully applied to equations modelling the cell division cycle of fission yeast. However, we cannot use this abstraction on a given differential-algebraic system automatically, because the process relies on expert choices, such as deduplication of some species, and the inclusion of some kinetic parameters as if they were species.

More generally, connections have been made between the dynamics of piecewise linear differential equations and asynchronous logical models: the asynchronous transition graph of the

logical models is seen as an approximation of the phase space of the differential equations [Sno89; JSB12]

Model simplification (section 2.5.3) and aggregation (section 2.5.4) can be seen as special cases of model conversion. The simplification process illustrated in example 2.5.2 is one way to grasp the relationship between reaction networks and Boolean networks. Indeed, the simplified reaction network we obtained in example 2.5.2 is an abstraction of the reaction network model we started with. Since the intermediary component was removed, it mechanically contains fewer reactions. At the limit of this abstraction-of-reaction reasoning, a Boolean network can in fact be seen as the *ultimate abstraction* of a reaction network, where no reaction remains.

Yet, in the abstraction we want to perform in this thesis, we want to get rid of the reaction without losing any species.

2.6 Wrap-up

In this thesis, we aim to convert existing reaction network models into corresponding Boolean network models. As we saw in this chapter, reaction networks and Boolean networks are the respective epitomes of mechanism-based models and influence-based models. They have very different philosophies, and the conversion from one to the other is relevant, at least for the following three reasons.

1. From an application standpoint, it would be an *ad hoc* solution to ease some analyses that are hard to run on reaction network models, instead of relying on the Boolean semantics, or model simplification. Such analyses are related to the control, or the exploration of the dynamic properties of the system.
2. From a theoretical standpoint, having access to such a conversion would certainly help to understand better the relationship between reaction networks and Boolean networks.
3. In turn, I think exploring the conversion from a reaction network to Boolean networks is useful to improve the Boolean network synthesis methods that use real biological data. Indeed, due to the proximity of reaction networks with their underlying biological systems, one can consider every Boolean network synthesis study to correspond indirectly to an implicit conversion from a reaction network to a Boolean network. Hence, any finding in a “controlled” synthesis (using inputs extracted from existing reaction network models) might be adapted back to “uncontrolled” synthesis (using real biological data).

In the studies mentioned before (section 2.5.5), the results of the Boolean abstractions of reaction networks are not really comparable with Boolean networks such as those synthesised in this thesis. Indeed, they are Boolean transition systems, with no guarantee about the possibility of deriving a Boolean transition function for each species of the original model.

The method I propose in this thesis takes inspiration from existing model synthesis methods that usually start from information about the structure and the dynamics, define a merit function, and use a dedicated parameter fitting algorithm to find the best models (the best combination of parameter values) that fit the data. Hence, in chapter 4 we will use an exhaustive clever search to synthesise *all* the Boolean networks that fit the best to a given structure and dynamics. But

before that, the next chapter ([chapter 3](#)) shows how to extract a structure and a dynamics from an existing reaction network model.

Chapter 3

Extracting the Structure and Dynamics of a Reaction Network

Contents

3.1	Introduction	75
3.2	The Structure of a Reaction Network	76
3.3	The Dynamics of a Reaction Network	77
3.3.1	From ODEs to Boolean Configuration Transitions	78
3.3.2	Concrete Numerical Simulation and Binarisation	89
3.3.3	Abstract Simulation	91
3.4	Related Works on the Extraction of a Transition Graph from a Reaction Network	96
3.5	Summary	97
3.5.1	Wrap-up	97
3.5.2	Discussion and Perspectives	98

3.1 Introduction

As we saw in the previous chapter, existing Boolean network synthesis methods rely on knowledge and data to guide the synthesis ([section 2.4.1](#) on page 58). This usually concerns the structure and the dynamics of the system. The knowledge about the structure is usually represented as a *prior influence graph* ([definition 1.7.7](#) on page 39), and dynamic data are usually represented as *times-series* ([definition 2.4.6](#) on page 61), *sequences of transitions* ([definition 2.4.4](#) on page 61) or *truth tables* ([definition 1.6.1](#) on page 29).

In this thesis, we want to synthesise Boolean networks starting from a given reaction network. The goal of this chapter is thus to show how to retrieve the structure and the dynamics from an existing reaction network.

We propose to retrieve the structure by parsing the input reaction network. The influence graph collects all the direct influences encoded by the model. If the input is a well-formed core reaction network ([definition 1.9.6](#) on page 45), the obtained influence graph is an overapproximation of the

sign abstraction of the Jacobian matrix of the (event hybridised algebraic-)differential equation system.

Concerning the extraction of the dynamics, what is studied here is technically an instance of the general discretisation problem, which consists of the computation of an abstract transition graph from a given reaction network. Various ways were proposed to retrieve such a graph. Here we propose two methods. Both rely on the differential semantics of the input reaction network model \mathcal{R} , which consists of an ODE system $odes(\mathcal{R})$ automatically reconstructed from the model.

The first method starts with a classic *concrete numerical simulation* of the system $odes(\mathcal{R})$. This method has a good trade-off between the accuracy and efficiency of the simulation. The data are then binarised with a midrange procedure.

The second method is referred to as *abstract simulation*. It is based on the abstract interpretation of a first-order formula built from the system $odes(\mathcal{R})$. However, the interpretation uses only three values $\mathbb{S} = \{\nearrow, \rightarrow, \searrow\}$. These values abstract the concrete values of the variables (species value and their derivatives values) to their signs (above/equal/below zero). We call the formula a First-Order Boolean Network with Non-deterministic Updates (FO-BNN), because it defines a transition function from which we ultimately build a Boolean transition graph. We prove that this graph is correct with regard to the Euler simulation of the original ODEs. Thanks to the abstract simulation, we can characterise the complete abstract behaviour of the ODEs. This is not possible with a concrete simulation since there are infinitely many initial configurations to start from. However, the abstract simulation imposes some restrictions on the input reaction network. In particular, it cannot involve any discontinuous event. Both simulation methods are thus complementary to one another.

Outline First, in [section 3.2](#), we explain how to construct a prior influence graph from the provided reaction network. Then, in [section 3.3](#), we explain how to retrieve the dynamics of a reaction network. Related work on the computation of transition graph from a reaction network is presented in [section 3.4](#). Finally, in [section 3.5](#), we conclude the chapter and discuss some choices about our strategy to retrieve the dynamics as well as future outlooks.

3.2 The Structure of a Reaction Network

The advantage of starting from an existing reaction network is that we have access to the structure of the system under study. However, this structure is a *reaction graph* ([definition 2.4.2](#) on page 59), whereas what we want is a prior *influence graph* ([definition 1.7.7](#) on page 39).

The approach we propose to retrieve the prior influence graph \mathcal{P} from a given reaction network \mathcal{R} needs to parse \mathcal{R} only once. For a given reaction network \mathcal{R} with species \mathcal{S} , the nodes of \mathcal{P} are the species of \mathcal{R} . As for the edges, they are obtained by applying the following routines. In what follows, X and Y are two species in \mathcal{S} .

On each reaction $\mathcal{R} = (R, e, P)$ of \mathcal{R} :

1. **if** Y is a reactant or an activator of \mathcal{R} and $P(X) - R(X) < 0$ (meaning X disappears)
then $Y \xrightarrow{-} X \in \mathcal{P}$
2. **if** Y is an inhibitor of \mathcal{R} and $P(X) - R(X) > 0$ (meaning X appears)
then $Y \xrightarrow{-} X \in \mathcal{P}$

3. **if** Y is a reactant or an activator of \mathcal{R} and $P(X) - R(X) > 0$ (meaning X appears)
then $Y \xrightarrow{+} X \in \mathcal{P}$
4. **if** Y is an inhibitor of \mathcal{R} and $P(X) - R(X) < 0$ (meaning X disappears)
then $Y \xrightarrow{-} X \in \mathcal{P}$

When the reaction uses a modifier whose exact role is unknown, the modifier is considered as both an activator and an inhibitor.

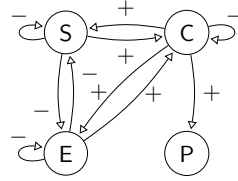
On each rule of \mathcal{R} :

5. **if** Y is a species which appears on the right side of a rule defining X
then $Y \xrightarrow{+} X \in \mathcal{P}$ and $Y \xrightarrow{-} X \in \mathcal{P}$

On each event of \mathcal{R} :

6. **if** Y is a species which appears in a condition triggering a discontinuous change of X
then $Y \xrightarrow{+} X \in \mathcal{P}$ and $Y \xrightarrow{-} X \in \mathcal{P}$

Example 3.2.1. The influence graph \mathcal{P}_{enz} constructed from \mathcal{R}_{enz} (example 1.9.4) is



Recall that a core reaction network consists only of reactions. For such a network, only the first four routines apply. These routines are those proposed by Fages et al. to retrieve the so-called *syntactic influence graph* of a reaction network [FS08b]. Moreover, they showed that the resulting influence graph is an overapproximation of the *differential influence graph*, which corresponds to the sign abstraction of the Jacobian matrix of the differential systems of a *well-formed* reaction network (definition 1.9.6 on page 45). However, if the reaction network is not well-formed, there is no such guarantee.

Example 3.2.2. Consider the reaction network model presented in example 1.9.7 on page 45. As mentioned previously, this model is not well-formed because according to the kinetic expression, Y is involved in the reaction ($\frac{\partial c}{\partial Y} > 0$), but this information is neither reflected by the list of reactants nor by the list of modifiers (Y is listed in none of the lists). The influence graph constructed from the reaction network is:



In this graph, Y is not even a parent of X , while according to the dynamics of the system, it should be its inhibitor (since Y activates the degradation of X).

3.3 The Dynamics of a Reaction Network

In this section, the goal is to extract the Boolean dynamics of a given reaction network. We propose two methods for this task. In both cases, we consider the differential semantics of the input reaction network (interpretation with a system of differential equations, see section 1.9.2),

and we end up with binarised trajectories represented by a (partial) Boolean transition graph. Unlike the continuous trajectories of a reaction network with the differential semantics, which are infinite objects, and for which there are infinitely many trajectories depending on the choice of the initial concentrations, a Boolean transition graph is necessarily finite. Each node of the resulting Boolean transition graph is a Boolean vector (i.e., a vector of Boolean values, one for each species) that represents infinitely many concrete configurations (i.e., a vector of real values, one for each species). A Boolean vector states, for each species, whether its concentration is above or below a given threshold. The specific value of the threshold depends on the context, and sometimes, it will be zero. In this case the Boolean vector states, for each species, if it is present or absent.

The first method is referred to as *concrete simulation*. It uses a classic clever numerical algorithm which makes a good approximation of the analytic solution of the ODE system. The concrete trajectories are then binarised after having defined a threshold for each species.

The second method exploits *abstract simulation*. Given a reaction network, the process starts with the construction of a FOL formula (section 3.3.3.1). Then, we interpret this formula using only three values $\mathbb{S} = \{\nearrow, \rightarrow, \searrow\}$. These values abstract the concrete values of the variables to their signs. The result of this interpretation is a list of transitions between abstract (Boolean) configurations of the reaction network. Moreover, by using John's soundness theorem for the abstract interpretation of FOL formulas [All21], we proved that the resulting transition graph is a sound overapproximation of the Euler simulation of the ODEs of the reaction network (section 3.3.3.3). However, for now, the method can only run on *core* reaction networks, those differential semantics only use functions defined from \mathbb{R}^+ (time) to \mathbb{R}^+ (amount or concentration).

Outline First, in section 3.3.1, we introduce and formalise three different relations between the configurations of the system. The first relation (*concrete temporally following*) relates the *concrete* configurations. The two others (*abstract temporally next* and *abstract causally next*) relate *abstract* (Boolean) configurations (those obtained after binarisation). We illustrate on a simple example how these relations are intertwined with one another. The goal of the concrete simulation followed by a binarisation step is to approximate the abstract temporally next as best as possible. In contrast, the goal of abstract simulation is to build an abstract transition graph that over approximates the causal abstract next relation. These two simulations are presented in sections 3.3.2 and 3.3.3 respectively.

3.3.1 From ODEs to Boolean Configuration Transitions

In this section, we illustrate the intuition behind the two methods we propose to derive a (partial) Boolean transition graph from the ODEs retrieved from a given reaction network. The section will use a simple example of ODEs modelling an object moving along an axis.

Outline Section 3.3.1.1 presents the example and recalls some basics about ODEs and their solutions. In particular, the fact that ODEs specifies a relation between functions and their derivative, and that solving an ODE system consists in finding the functions such that the specifications are met. However, what we want ultimately is to construct *transitions* between the Boolean configurations of the system. Thus, we need to change our perspective about what an ODE system represents. This is initiated in section 3.3.1.2, where we start to consider the values of the solution functions indexed by time, instead of the function themselves. In the

literature, this interpretation of differential equations is popular for formal reasoning about ODEs and hybrid systems in general [Pla18]. Based on the change of perspective, we introduce in [section 3.3.1.3](#), a relation between the successive concrete configurations of the system. We refer to this relation as the *concrete temporally following* relation. This relation can be computed exactly, but only partially thanks to the analytical solution of the ODEs. We illustrate this in [section 3.3.1.4](#). Unfortunately, we cannot always get access to an analytical solution of the ODEs. This is where numerical simulations come to the rescue. In [section 3.3.1.5](#), we use the Euler algorithm to approximate (part of) the concrete temporally following relation,. We will see that the Euler algorithm is not particularly efficient, and that the approximation errors it introduces actually reflect the causation of the changes.

In [section 3.3.1.6](#) we introduce the binarisation of the concrete configurations we were dealing with up to now. In the abstract setting, the number of configurations is finite, and each Boolean configuration represents an infinite number of concrete configurations. By abstracting the solutions computed with the analytical solution and with the Euler algorithm, we define two relations: the *abstract temporally next* and the *abstract causally next* respectively. They differ in that the temporally next relation washes away the causation that is kept by the abstract causally next relation.

These two relations are the basis of the two methods we propose to retrieve the dynamics of a reaction network. Indeed, the first method aims at approximating the abstract temporally next relation, while the second method overapproximates the abstract causally next relation.

3.3.1.1 ODEs: Back to Basics

As seen in the preliminaries ([section 1.5](#) on page 27), the general form of an ODE is $\dot{y} = f(y)$. Moreover, recall that an ODE system can be encoded as a first-order formula ϕ whose variables represent real-valued functions.

Example 3.3.1 (An object moving along an axis). *The following system of differential equations models an object moving along an axis. The position is denoted x , and its derivative \dot{x} equals the velocity v , whose derivative \dot{v} equals the acceleration a , which is constant:*

$$\begin{aligned}\dot{a}(t) &= 0 \\ \dot{v}(t) &= a(t) \\ \dot{x}(t) &= v(t)\end{aligned}\tag{3.1}$$

Equivalently, in logic, this is encoded as

$$\dot{a} \doteq 0 \wedge \dot{v} \doteq a \wedge \dot{x} \doteq v\tag{3.2}$$

The variables of an ODE system are *functions* (in the general mathematical sense, not in the logic sense). Solving an ODE system consists in finding functions such as the specifications encoded by the ODEs are met. In other words, a function s is *solution* of the ODEs on any duration r iff for all times $0 \leq z \leq r$, $\dot{s}(z) = f(s(z))$, such as specified by the ODEs. In logic, an ODE formula φ is solved on the structure $S_{\mathbb{R}_+ \rightarrow \mathbb{R}_+}$.

Example 3.3.2 (Moving object: analytical solution). We continue [example 3.3.1](#). The analytical solution of [equation \(3.1\)](#) is

$$\begin{aligned} a(t) &= C_1 \\ v(t) &= C_1 t + C_2 \\ x(t) &= C_1 \frac{t^2}{2} + C_2 t + C_3 \end{aligned} \tag{3.3}$$

with C_1, C_2, C_3 being constant. [Appendix B](#) on page 179 shows in detail how to derive it.

3.3.1.2 ODE: change in perspective

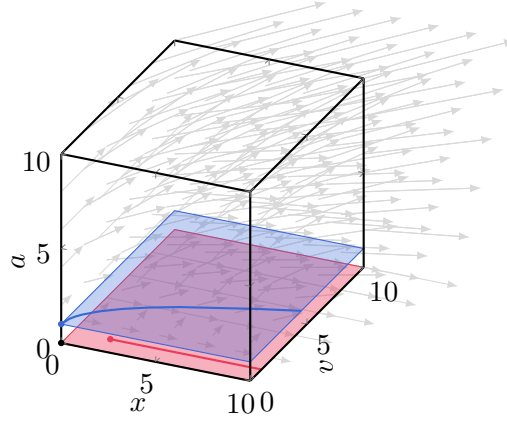
The change of perspective consists in stopping considering the functions solving the ODE system, but their values at all points in time.

These values correspond to the *configuration* at which the system is at a given time. Formally, a *base configuration* is a vector storing the values of the base variables (those not rooted by the dot operator). A configuration that also stores the values for the derivatives is referred to as a *enriched configuration*.

Example 3.3.3 (Moving object: configurations). For the system presented in [example 3.3.1](#), a *base configuration* is a vector (x_t, v_t, a_t) . An *extended configuration* is a vector $(x_t, v_t, a_t, \dot{x}_t, \dot{v}_t, \dot{a}_t)$.

We can visualise all the possible base configurations of an ODE system as a n -dimensional space, with n being the number of functions involved in the system. Moreover, we can see the ODE system as a *vector field*. Indeed, the ODEs associate to each base configuration, a vector $(\dot{x} \mid x \in V)$ whose coordinates are the values of the derivative. These vectors state how each configuration tends to change: as the system evolves from some initial configuration, the vector field “guides” the point along some trajectory such that, at every moment, its direction and speed matches the vectors from the vector field. Here, the words “direction” and “speed” do not refer to the physical position and velocity of the moving object in the running example, but to an abstract change and rate of change of the configuration of the system.

Example 3.3.4 (Moving object: vector field). Below, we visualise the vector field of [equation \(3.1\)](#), along with three trajectories starting from three different initial configurations. Each point of coordinates (x_t, v_t, a_t) in the three dimensional space, is associated to a vector of coordinates $(\dot{x}_t, \dot{v}_t, \dot{a}_t)$. However, to prevent clutter, the vectors are only drawn for some of the points and their length has been scaled (divided by 2). The black dot has the coordinates $(0, 0, 0)$. If the system starts in this configuration, it stays there. The trajectory starting from the initial conditions $(0, 0, 1)$ and $(2, 1, 0)$ are respectively drawn in blue and red. Since $\dot{a} = 0$, the trajectories stay in the plan whose coordinate for a is equal to the initial value of a ($a = a_0$). The plan is drawn in the same colour as the corresponding trajectory.



In logic, the change of perspective is reflected by changing the representation of the ODEs. For any ODE system φ , we introduce its corresponding enriched ODE formula, where a term \dot{x} will not only be an expression but a variable in its own right (now denoted \dot{x}). Now, an assignment for enriched ODE formula is thus both about base variables (the ones without the dot operator) and the dotted variables. Also, we will stop assigning functions, and we will use values (either reals, or signs, depending on the context).

Definition 3.3.5 (Enriched ODE). *An enriched ODE is an expression $\dot{\varphi} \in \mathcal{F}_{\Sigma_{arith}}$. It is obtained by replacing any subexpression of the form \dot{x} in ϕ by the variable \dot{x} :*

$$\dot{\phi} =_{def} \phi[\dot{x}/\dot{x} \mid x \in fv(\phi)]$$

The set of variables \mathcal{V} of $\dot{\varphi}$ is thus $V \cup \dot{V}$ instead of V for the not enriched ODE φ .

Note that in our case, no dot operators remain in $\dot{\phi}$, since all subexpressions of ϕ that are rooted by the dot operator must be of the form \dot{x} with $x \in V$ by the definition of ODEs (definition 1.5.2 on page 28). However, the transition semantics can generally be applied to higher-order ODEs, using \dot{e} with e being any expression (not necessarily a variable) [Pla18].

The change in perspective can be summarised by the following lemma, where we index the values of the functions by the time.

Lemma 3.3.6. *If $\beta \in sol^{\mathbb{R}_+ \rightarrow \mathbb{R}}(\phi)$ then $\beta_t \in sol^{\mathbb{R}}(\dot{\phi})$.*

Proof. The lemma comes from the indexation of the functions values by time. Indeed, we have that, for any assignment $\beta : V \rightarrow (\mathbb{R}_+ \rightarrow \mathbb{R}_+)$ to an ODE formula, and time point $t \geq 0$ we have an assignment for an extended ODE formula $\beta_t : \{x, \dot{x} \mid x \in V\} \rightarrow \mathbb{R}$ such that: $\beta_t(x) = \beta(x)(t)$ (in \mathbb{R}_+) and $\beta_t(\dot{x}) = \cdot^{\mathbb{R}_+ \rightarrow \mathbb{R}}(\beta(x))(t)$ (in \mathbb{R} , since derivatives values of positive function might be negative). \square

The concepts of concrete base configuration and concrete enriched configuration introduced earlier can be defined from an enriched ODE formula.

Definition 3.3.7 (Concrete configuration of an enriched ODE $\dot{\varphi}$). *Let $\dot{\varphi}$ be an enriched ODE formula with variables $\mathcal{V} = V \cup \dot{V}$. We call a function $\omega : \mathcal{V} \rightarrow \mathbb{R}$ an enriched concrete configuration of $\dot{\varphi}$. We denote $\mathbb{R}^{\mathcal{V}}$ the infinite set of all possible concrete enriched configurations.*

The restriction $\omega|_V$ of an enriched concrete configuration ω on the base variables V is a base concrete configuration of $\dot{\varphi}$.

Similarly, we also define the concept of abstract enriched configuration.

Definition 3.3.8 (Abstract configuration of an enriched ODE $\dot{\varphi}$). *Let $\dot{\varphi}$ be an enriched ODE formula with variables $\mathcal{V} = V \cup \dot{V}$. We call a function $\omega : \mathcal{V} \rightarrow \mathbb{S}$ an enriched abstract configuration of $\dot{\varphi}$. We denote $\mathbb{S}^{\mathcal{V}}$ the finite set of all possible abstract enriched configurations. The restriction $\omega|_V$ of an enriched abstract configuration ω on the base variables V is a base abstract configuration of $\dot{\varphi}$.*

Given an enriched configuration ω (either concrete or abstract), $\omega(x)$ indicates the sign value of the variable $x \in \mathcal{V}$ (which can be rooted by the dot operator or not) in the ω .

3.3.1.3 The Concretely Following Relation

One of the relations we are interested in is the one that relates a configuration of the system to all the configurations that are accessible from it. In other words, a configuration ν follows a configuration ω if we reach ν when starting in ω and following the differential equations for some duration r . We say that ν *concretely follows* ω , and we refer to this relation as the *concrete following* relation.

On the vector field presented earlier, it means that, starting from ω , ν is on the trajectory defined by the vector field for a duration r . That is, a configuration ν concretely follows a configuration ω if we end up in ν when following the ODEs from ω for some time. In our setting, the time is continuous. The successive configurations ν can be ordered by the duration r during which we followed the equations. However, it makes no sense to talk about the *next* configuration. This is why we use the term *following*.

Example 3.3.9 (Moving object: concretely following configurations). *In the vector field from [example 3.3.4](#), all the configurations laying on the coloured lines concretely follow the initial configuration of the corresponding colours.*

In logic, this translates as follows. We define the concrete following relation as a binary reachability relation $\mathbb{R}^{\mathcal{V}} \times \mathbb{R}^{\mathcal{V}}$ over the enriched configurations of the enriched ODE. The relation relates the configurations connected by the ground-truth analytical solution of the ODE system.

Definition 3.3.10 (Concretely following relation). *Let φ be an ODE formula over variables V , $s = \beta(x)$ a function from $\mathbb{R}_+ \rightarrow \mathbb{R}_+$ satisfying φ for some variable $x \in V$ on a time interval $[t_1, t_2]$. The enriched counterpart of φ is $\dot{\varphi}$, those variables are $\mathcal{V} = V \cup \dot{V}$. Let $\omega, \nu : \mathcal{V} \rightarrow \mathbb{R}$ be two concrete extended configurations of $\dot{\varphi}$. We say that ν is temporally reachable from ω (denoted $(\omega, \nu) \in \text{sol}(\dot{\varphi})$) if for all variables $x \in V$, $\omega(x) = s(t_1)$ and $\nu(x) = s(t_2)$. Moreover, the value of \dot{x} at any time $t'_2 \in [t_1, t_2]$ is constrained along all the solution s : its value is equal to the analytic time-derivative at time t'_2 : $\dot{x}(t'_2) =_{\text{def}} \frac{ds(t'_2)}{dt'_2}$.*

3.3.1.4 Compute the Concretely Following Relation

We can compute part of the concretely following relation thanks to the analytical solution of an ODE system, in which can simply plug a duration r and an initial configuration ω to compute the reached configuration ν .

The computation is only partial, and this for two reasons:

- There are an infinite number of initial configurations, and each of them has its own set of concrete following configurations.
- Starting from a given configuration, there are an infinite number of durations, and except in some very specific cases (fixed point) each of them yields a different reached configuration.

From this follows one of the crucial points of this section: ODEs are infinite objects, and it is not possible to explore their dynamics exhaustively by simply changing the initial configuration and durations.

Example 3.3.11 (Moving object: configurations that concretely follow $[x:0, v:0, a:1]$). We start from [equation \(3.3\)](#). Let us say the initial configuration is that at $t = 0$, the point lays at the origin, has no velocity, but an acceleration of 1:

$$a(0) = 1 \quad v(0) = 0 \quad x(0) = 0$$

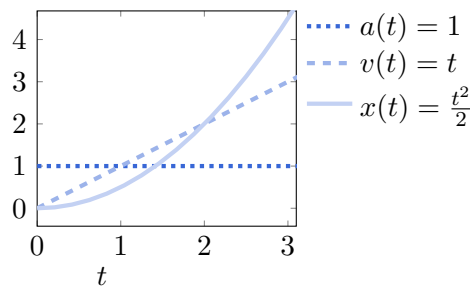
We thus have:

$$\begin{aligned} C_1 &= a(0) & \Rightarrow C_1 &= 1 \\ C_2 &= v(0) - C_1 t = 0 - 1 \times 0 & \Rightarrow C_2 &= 0 \\ C_3 &= x(0) - C_1 \frac{t^2}{2} - C_2 t = 0 - 0 - 0 & \Rightarrow C_3 &= 0 \end{aligned}$$

In conclusion:

$$a(t) = 1 \quad v(t) = t \quad x(t) = \frac{t^2}{2}$$

The plot of these functions against time is shown below, and [figure 3.1a](#) summarises the configurations reached when starting from $[x:0, v:0, a:1]$ and following the differential equations for a duration $r \in \{0, 1, 2, 3\}$.



Example 3.3.12 (Moving object: configurations that concretely follow $[x:5, v:1, a:0]$). We start from [equation \(3.3\)](#) and we change the initial conditions to

$$a(0) = 0 \quad v(0) = 1 \quad x(0) = 5$$

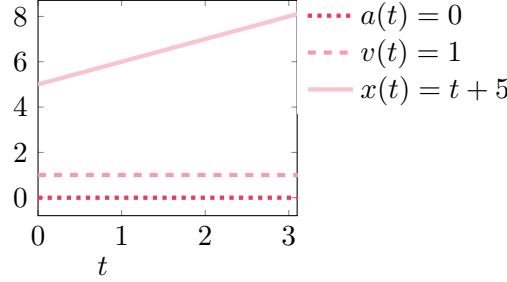
The constants now have the values:

$$C_1 = 1 \quad C_2 = 0 \quad C_3 = 0$$

The analytical solution thus changes to:

$$a(t) = 0 \quad v(t) = 1 \quad x(t) = t + 5$$

We can reflect the changes visually by plotting x , v , a against time as well:



3.3.1.5 Approximate the Concretely Following Relation

We saw how to retrieve the analytical solution of an ODE and use it to compute (part of) the *concretely following* relation. However, as mentioned in [section 1.5.2](#), it is not always possible to find an analytical solution. Despite this, there is still a meaningful way to get an idea of what the dynamics looks like, namely, *numerical simulation*. A numerical simulation consists in iterating an algorithm that approximates successive configurations. Among existing possible numerical methods, the Euler method is the simplest.

Example 3.3.13 (Moving object: Euler approximations). *Figure 3.1 shows the approximated values for x , v and a at $t \in 0, 1, 2, 3$ when starting from $[x:0, v:0, a:1]$ at $t = 0$, and using different time discretisation steps ($\Delta \in \{2, 1, 0.5\}$).*

(a) With the analytical solution

t	0	1	2	3
a	1	1	1	1
v	0	1	2	3
x	0	0.5	2	4.5

(c) With Euler and $\Delta = 1$

t	0	1	2	3
a	1	1	1	1
v	0	1	2	3
x	0	0	1	3

(b) With Euler and $\Delta = 2$

t	0	1	2	3
a	1	1	1	1
v	0	0	2	2
x	0	0	0	0

(d) With Euler and $\Delta = 0.5$

t	0	1	2	3
a	1	1	1	1
v	0	1	2	3
x	0	0.25	1.5	3.75

Figure 3.1: Successive configurations of [equation \(3.1\)](#) according to the analytical solution ([equation \(B.1\)](#) on page 180) and to the Euler algorithm using different time discretisation steps ($\Delta \in \{2, 1, 0.5\}$).

Now, we show how the Euler algorithm approximates the successive configurations. We illustrate the algorithm on a scalar function, whose configurations thus consist of a vector of one value. However, everything we will derive from how the algorithm works with a scalar function

will also hold when dealing with a *system* of differential equations. In particular, it applies to what we will derive in terms of accuracy and efficiency.

The Euler algorithm works by iterating as follows. Starting from a given configuration it takes a step for a duration Δ based on the direction encoded by the vector on which the configuration sits. The point where it lands has the value $\hat{f}(t + \Delta)$, which is an *approximation* of $f(t + \Delta)$. The procedure for one iteration is visualised in [figure 3.2](#) and summarised by the following formula:

$$\hat{f}(t + \Delta) = \hat{f}(t) + \Delta \times \dot{f}(t).$$

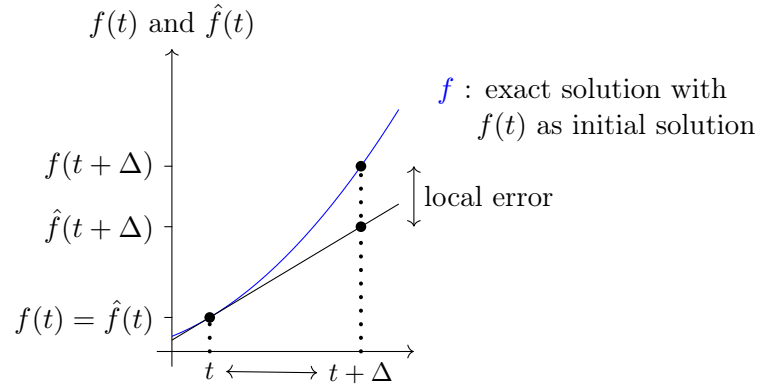


Figure 3.2: Procedure of the Euler algorithm on one iteration. The scalar function f to approximate is plotted in blue.

Error accumulation At each iteration of the procedure, the error accumulates. Hence, after several iterations, one might be quite far from the actual value. In [figure 3.3](#), we plot the Euler approximation on three iterations. The function to approximate is in blue. The black points are the approximations for $t \in \{0, \Delta, 2\Delta, 3\Delta\}$. The *local error* at a time t is computed in regard to what f would have looked like if starting from $\hat{f}(t)$, while the *global error* is simply the difference between $f(t)$ and $\hat{f}(t)$.

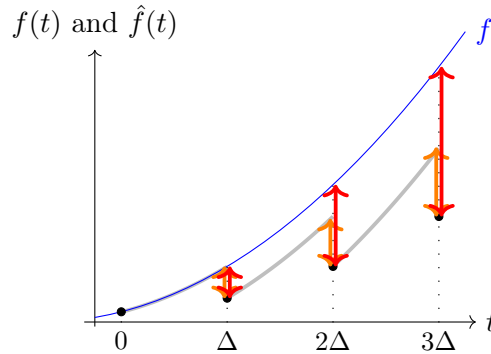


Figure 3.3: Error accumulation after several iterations of the Euler algorithm. The scalar function f to approximate is plotted in blue. The local and global errors at each iteration are respectively plotted in orange and red. The local error are computed in regard to the solution with $f(t)$ as initial condition (in gray).

Accuracy and Causation For a given step size Δ , the accuracy of the approximation heavily depends on the *stiffness* of the problem (i.e., how fast are the changes occurring). Conversely, the accuracy depends on the choice of Δ itself: the approximations approach the true solution as Δ gets smaller. This is illustrated in [figure 3.4](#).

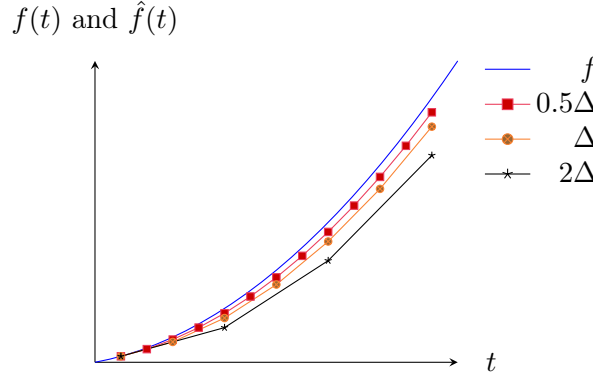


Figure 3.4: More accurate approximation with the Euler algorithm using smaller Δ .

However, no matter how small Δ is, the function is arbitrarily far from the true solution. In particular when dealing with a system of ODE, the Euler algorithm captures the *direct causation* between the variables changes.

Example 3.3.14. *Whatever the choice of Δ , and for any initial condition such that $a(0) > 0, v(0) = 0, x(0) = 0$, the Euler algorithm will always need two iterations to set x at a value different from 0. This is because it is the change in speed that produces the change in position, but the speed first needs to be updated from the acceleration. In contrast, with the analytical solution, the only time point where $x(t) = 0$ is at $t = 0$. This is reflected in [figure 3.1](#).*

Efficiency of the Computation Relatively to the Step Size We just saw that the smaller the Δ , the more accurate the approximations are. However, the computation is also less efficient since it requires more iterations to compute the values on the same time interval.

Remark 3.3.15. *More sophisticated numerical methods can reduce the error made by the Euler methods, and balance the tradeoff between accuracy and efficiency. We will be using one of such methods in [section 3.3.2](#).*

3.3.1.6 Abstraction

Recall that what we want ultimately is to get access to transitions between Boolean configurations. Yet, for now, we are only dealing with transitions between concrete configurations (on reals).

Let us consider the following binarisation, where the function $h_{\mathbb{S}} : \mathbb{R} \rightarrow \mathbb{S}$ abstracts a given concrete value $r \in \mathbb{R}$ relatively to a threshold θ :

$$h_{\mathbb{S}}^{\theta}(r) = \begin{cases} \nearrow & \text{if } r > \theta \\ \rightarrow & \text{if } r = \theta \\ \searrow & \text{if } r < \theta \end{cases}$$

The binarisation transforms infinite sequences of configurations into finite sequences of configurations. Indeed, several concrete configurations map to one abstract configuration. Moreover, after the binarisation, it is possible to define the *next* abstract configuration.

Example 3.3.16 (Moving object: abstraction of the sequences of configurations). *For each initial concrete base configuration such that $[x:0, v:0, a>0]$, the binarisation of an Euler simulation will always return*

$$[x:0, v:0, a:1] \rightarrow [x:0, v:1, a:1] \rightarrow [x:1, v:1, a:1] \dot{\cup}.$$

The configuration $[x:0, v:1, a:1]$ is the next configuration of $[x:0, v:0, a:1]$. The binarisation of the analytical solution from the same initial condition is:

$$[x:0, v:0, a:1] \rightarrow [x:1, v:1, a:1] \dot{\cup}.$$

The abstraction of the analytical solution and Euler solutions are the base of our formal definitions for the *abstract temporally next* relation and the *abstract causally next* relation, respectively. We now give the formal definitions of these relations in the two following sections.

3.3.1.7 Abstract Temporally Next Relation

The abstraction of the analytical solution is the base of our formal definition for *abstract temporal next*.

Definition 3.3.17 (Abstract temporally next relation *tnext*). *Let $\gamma_1, \gamma_2 : \mathcal{V} \rightarrow \mathbb{S}$ be two abstract enriched configurations of an enriched ODE formula $\dot{\phi}$. We call γ_2 a temporally abstract next configuration of γ_1 with respect to $\dot{\phi}$ and write $(\gamma_1, \gamma_2) \in \text{tnext}_{\dot{\phi}}$ if there exists a real-valued function $\beta \in \text{sol}^{\mathbb{R} \rightarrow \mathbb{R}}(\varphi)$ and two time points $0 \leq t_1 < t_2$ such that, for all variables $x \in \mathcal{V}$ and time points $t'_2 \in]t_1, t_2]$: $\gamma_1(x) = h_{\mathbb{S}}^{\theta}(\beta(x)(t_1))$ and $\gamma_2(x) = h_{\mathbb{S}}^{\theta}(\beta(x)(t'_2))$.*

A visual illustration of the definition can be found in [figure 3.5](#).

As we will see after, the causation of the changes can be deduced from the signs of the derivatives. In the meantime, if not interested in the values of the derivatives, we can consider the restriction of the successor relation on the base variables (those without the dot operator). A visual illustration of this can be found in [figure 3.6](#).

Remark 3.3.18. *The next base configurations might not be unique.*

3.3.1.8 Abstract Causally Next Relation

The following relation is different from the abstract temporally next in that it captures the causation of the changes from the derivatives, as illustrated before (see [example 3.3.14](#)).

Definition 3.3.19 (Abstract causally next relation *cnext*). *Let $\dot{\phi}$ be an extended ODE system with variables $\mathcal{V} = V \cup \dot{V}$, and $\gamma_1, \gamma_2 : \mathcal{V} \rightarrow \mathbb{S}$ be two abstract extended configurations. We call γ_2 a causally-next extended configuration of γ_1 and write $(\gamma_1, \gamma_2) \in \text{cnext}_{\dot{\phi}}$ if $\gamma_2 \in \text{sol}^{\mathbb{S}}(\dot{\phi})$ and there exists an abstract configuration γ'_2 such that $(\gamma_1, \gamma'_2) \in \text{tnext}_{\dot{\phi}}$ and for all base variable $x \in V$: $\gamma_2(x) = \gamma_1(x)$ if $\gamma'_2(\dot{x}) = 0$ and $\gamma_2(x) = \gamma'_2(x)$ otherwise.*

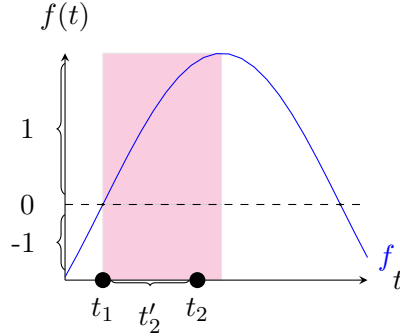


Figure 3.5: Illustration of the abstract temporally next relation (definition 3.3.17). Let f be the function assigned by β for some variable v in the set V of some ODE formula ϕ , such that $\beta(v)$ is solution of ϕ . The function f is plotted in blue, and the threshold θ is plotted with a dashed line. At time t_1 , the abstract configuration is $[x:0, \dot{x}:1]$. At time t_2 and at all timepoints $t'_2 \in]t_1, t_2]$, the abstract configuration is $[x:1, \dot{x}:1]$. The abstract configuration $[x:1, \dot{x}:1]$ is thus a temporal abstract next configuration of the abstract configuration $[x:0, \dot{x}:1]$. In fact, all the pink area corresponds to time points those corresponding abstracted configuration is the temporal abstract next configuration of the abstract configuration that corresponds to time point t_1 . The pink area stops as soon as \dot{x} switches from 1 to 0.

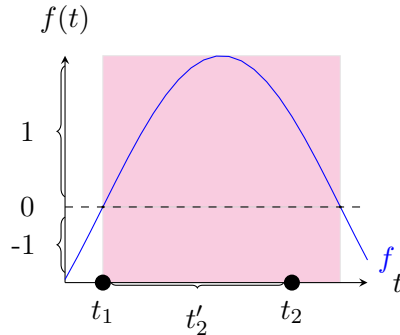


Figure 3.6: Illustration of the abstract temporally next relation (definition 3.3.17) after restriction on the base variables. We use the same setting as before: f is the function assigned by β for some variable v in the set V of some formula ϕ , such that $\beta(v)$ is solution of ϕ . The function f is plotted in blue, and the arbitrary threshold θ is plotted with a dashed line. All values above or equal the threshold map to 1 and all below map to 0. We thus end up with abstract configurations using values $\{0, 1\} \in \mathbb{S}$. At time t_1 , the abstract *base* configuration is $[x:0]$. It can be thought of as the last time point before it switches to $[x:1]$. At time t_2 and at all timepoints $t'_2 \in]t_1, t_2]$, the abstract configuration is $[x:1]$. The abstract configuration $[x:1]$ is thus a temporal abstract next base configuration of $[x:0]$. Again, the pink area corresponds to the time points those corresponding abstracted configuration is the temporal abstract next configuration of the abstract configuration that corresponds to time point t_1 . But this time, the pink area is wider since the values of the derivative are not considered.

Again, the configurations can be restricted to consider only the base variables: we say that $\gamma_{2|V}$ is a causally-next *base* configuration of $\gamma_{1|V}$ and write $(\gamma_{1|V}, \gamma_{2|V}) \in \text{cnext}_\phi$ if $(\gamma_1, \gamma_2) \in \text{cnext}_\phi$.

Remark 3.3.20. *We believe that $\text{tnext}_\phi \subseteq (\text{cnext}_\phi)^*$ holds for any ODEs for which the numerical simulation by Euler’s algorithm is sound, that is, when using exact arithmetic and an adapted step size.*

3.3.2 Concrete Numerical Simulation and Binarisation

The goal here is to approximate the relation abstract temporally following (definition 3.3.17) as well as possible. To do so, we first run a deterministic numerical simulation of the differential-algebraic system of equations retrieved from the reaction network. Then, we binarise the resulting timeseries.

For the numerical simulation, we want to use a method that works well for all kinds of reaction networks. In particular, it is not rare for the reaction networks from the literature to model phenomena that involve mechanisms with varying time scales. This ultimately leads to ODE systems with stiff dynamics [GW82] (i.e., with both rapid and slow variations). Unfortunately, to perform an accurate simulation of a problem that might be stiff, the Euler algorithm used in the section 3.3.1.5 would require an impractical amount of small steps, because the step size needs to be small enough so the algorithm does not miss sudden changes. We thus propose to use the Livermore Solver for Ordinary Differential Equations (LSODE for short) [RH93]. It is part of the clever algorithms mentioned in section 3.3.1, that achieve a good tradeoff between the accuracy of the approximation and the efficiency of the computation. In particular, LSODE automatically adapts the step size to limit the local error to a predefined value. Moreover, it uses Backward Differential Formula (BDF) method for stiff solving and the Adam-Moulton (AM) method for non-stiff solving.

In the literature, stiff problems are often referred to as systems with large Lipschitz constants [AFK93]. But the stiffness property is actually local in that the condition above may be true in some intervals and not in others. LSODE automatically detects the stiffness of an ODE $\dot{y} = f(t, y)$ based on the evaluation of the eigenvalues of the Jacobian matrix of the system i.e., the square matrix J with elements J_{ij} defined as $\partial f_i / \partial y_j$ with $i, j = 1, \dots, N$. This matrix represents the best linear approximation of f at a close neighbourhood of a given configuration C . Hence, for the same function f , the Jacobian can differ depending on C . Once we have this matrix, we can compute its eigenvalues, which represent the scaling factors of the eigenvectors, i.e., the vectors that remain in their own span despite the linear transformation represented by the matrix J . In [SG79], Shampine et al. state:

“By a stiff problem we mean one for which no solution component is *unstable* (no eigenvalue has a real part which is at all large and positive) and at least some component is very stable (at least one eigenvalue has a real part which is large and negative). Further, we will not call a problem stiff unless its solution is slowly varying with respect to the most negative real part of the eigenvalues.”

In other words, negative eigenvalues of the Jacobian mean that the time evolution points back to the configuration C while positive eigenvalues indicate that it points away from it.

In the former case, LSODE switches to non-stiff solving (AM method) while in the latter case, it switches to stiff solving (BDF method). Since BDF has more computational overhead, the ability of LSODE to switch back to non-stiff solving automatically when not necessary anymore is really appreciable.

Another thing to know is that BDF and AM methods are *multistep* and *implicit* methods. This kind of methods is known to achieve better accuracy than one-step and explicit methods (such as the Euler algorithm). In contrast with a one-step method solver, a multistep method calculates an approximation at a given timestep by solving an equation involving the approximation done at several previous timesteps (instead of using only the approximation computed at the previous timestep). The number of timesteps it needs is the *order* of the method. LSODE automatically adjusts the order of BDF and AM. In contrast with an explicit method, an implicit method uses the yet unknown value of $f(t)$ to compute $\hat{f}(t)$. They thus rely on a first guess (prediction step) which is then adjusted later (correction step).

Overall, LSODE is:

Consistent As the step size Δ goes to zero, the local error goes to zero even faster. In other words, the error divided by the integration step size goes to zero as the step size goes to zero.

Stable The numerical solution does not diverge when the exact solution isn't diverging, hence the global error is *bounded* for a given step size and time interval (small changes in the initial condition result in small changes in the computed solution) even for stiff problems, thanks to the use of BDF which is stable.

Convergent The global error goes to 0 when the step size Δ goes to 0.

These are some interesting features which guarantees good numerical approximations, even for stiff problems.

3.3.2.1 Concrete Simulation for Extended Reaction Networks

LSODA can be used for reaction networks those differential semantics involve spaces of values extended to $\mathbb{R} \rightarrow \mathbb{R}$, and with additional rules. Another important extension of the method is LSODAR, standing for Livermore Solver for Ordinary Differential equations, with Automatic method switching for stiff and non-stiff problems, and with Root-finding. It is able to determine with precision when to trigger discontinuous events (even though it remains an approximation), which is precious when simulating reaction networks involving such events.

3.3.2.2 Binarisation

To go from continuous time series to sequences of Boolean configurations, we apply a binarisation procedure. That is, we compute a threshold θ_X for each species. Then, with X_t the value of X at time t in the timeseries, the binarised value \hat{X}_t of X at time t is

$$\hat{X}_t = \begin{cases} 1 & \text{if } X_t \geq \theta_X \\ 0 & \text{otherwise.} \end{cases}$$

Many binarisation approaches exist [BN13]. Yet, we decided to go by default with the midrange threshold.

$$\theta_X = \frac{\min + \max}{2}$$

where min and max are the minimum and maximum values of X observed in the timeseries. The main reason for this choice is that it is quite natural, and usually gives good results despite its simplicity [Tys+96; Vid+15; Ost+16].

3.3.2.3 Application on \mathcal{R}_{enz}

Figure 3.7 shows the result of the numerical simulation of the ODE system in example 1.9.8 (which does not use rules nor events) from $t = 0$ to $t_{\text{max}} = 100$ seconds (chosen arbitrarily) starting from the following initial conditions:

$$\begin{aligned} S(0) &= 1.0 \times 10^{-5} \text{ mol/L} \\ E(0) &= 0.5 \times 10^{-5} \text{ mol/L} \\ P(0) &= 0 \text{ mol/L} \\ C(0) &= 0 \text{ mol/L} \end{aligned} \tag{3.4}$$

figure 3.8 shows the result of the midrange binarisation.

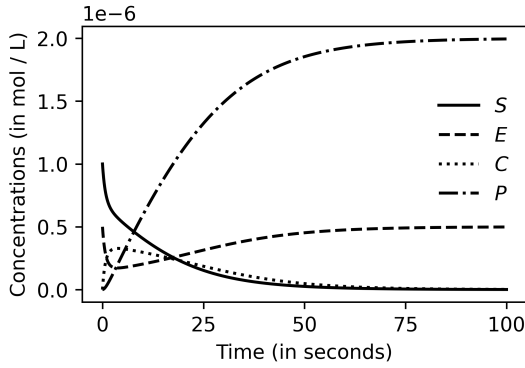


Figure 3.7: Timeseries for \mathcal{R}_{enz} with initial concentrations from equation (3.4).

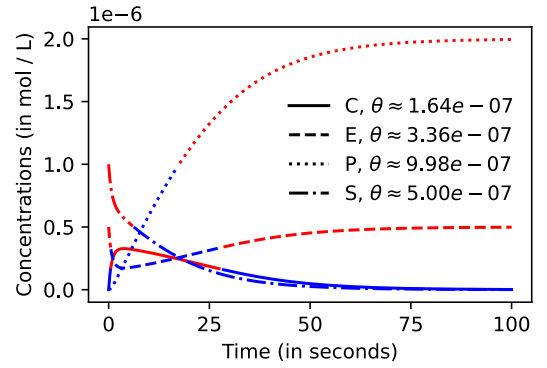


Figure 3.8: Binarised timeseries for \mathcal{R}_{enz} , with midrange-based binarisation thresholds (blue if binarised to 0 and red if binarised to 1).

3.3.3 Abstract Simulation

The goal of the abstract simulation is to build a Boolean transition graph that overapproximates the causal next transition of ODEs (see definition 3.3.19).

Having defined an ODE system as a first-order formula (section 1.5) is justified in this particular section. Indeed, this framework is suitable for the syntactical transformations and the abstract interpretation which we will use here. Indeed, the abstract simulation of a reaction network with the differential semantics consists in interpreting a so-called FO-BNN formula on

the structure of signs. As we will see in [section 3.3.3.1](#), an FO-BNN is a first-order formula ([section 1.2.2](#)) built from an ODE expression. FO-BNN stands for First-Order Boolean Network with Non-deterministic updates. The name comes from the fact we can use them to define a relation on abstract configurations (i.e., a relation $\mathbb{B}^{\mathcal{S}} \times \mathbb{B}^{\mathcal{S}}$, with \mathcal{S} the set of species considered) and that a Boolean network \mathcal{B} with non-deterministic updates and species in \mathcal{S} is also some kind of definition of a Boolean transition graph $\mathcal{G} \subseteq \mathbb{B}^{\mathcal{S}} \times \mathbb{B}^{\mathcal{S}}$ [PS21]. The non-determinism of the updates in an FO-BNN in fact reflects the non-determinism of the interpretation of the formula with the structure of signs. As proven in [section 3.3.3.3](#), this relation consists of a sound approximation of the causal transition of the ODEs from which it is constructed.

Finally, in [section 3.3.3.2](#) we show how to use an FO-BNN to compute a transition graph. The computation can be done only partially, when are only interested in the configurations that are reachable from a given set of abstract configurations.

3.3.3.1 First-Order Boolean Networks with Non-deterministic Updates

We assume that $\mathcal{S} \subseteq \mathcal{V}$ and fix two bijections $\cdot_{\text{next}} : \mathcal{V} \rightarrow \mathcal{V}$ and $\circ : \mathcal{V} \rightarrow \mathcal{V}$ such that $\mathcal{S}, \mathring{\mathcal{S}}, \mathring{\mathcal{S}}_{\text{next}}$, and $\mathring{\mathcal{S}}_{\text{next}}$ are disjoint subsets of \mathcal{V} . Furthermore, we assume that $\cdot_{\text{next}}(\circ(x)) = \circ(\cdot_{\text{next}}(x))$ for any $x \in \mathcal{V}$. In other words, applying the composition $\cdot_{\text{next}} \circ \circ$ is the same as applying the opposite composition $\circ \circ \cdot_{\text{next}}$.

Definition 3.3.21 (FO-BNN of a reaction network). *For any reaction network \mathcal{R} with species $\mathcal{S} = \{A_1, \dots, A_n\}$, we define its FO-BNN $bnn(\mathcal{R})$ as a first-order formula $\phi \in \mathcal{F}_{\Sigma_{arith}}$ with free variables $fv(\phi) = V \cup V_{\text{next}}$ (recall that $V \subseteq \mathcal{V}$):*

$$\begin{aligned} bnn(\mathcal{R}) =_{\text{def}} & \exists \mathring{A}_1. \dots \exists \mathring{A}_n. \text{odes}(\mathcal{R}) \\ & \wedge \exists \mathring{A}_1_{\text{next}}. \dots \exists \mathring{A}_n_{\text{next}}. \text{odes}_{\text{next}}(\mathcal{R}) \\ & \wedge \bigwedge_{i=1}^n \text{next_spec}(A_i, \mathring{A}_i, A_i_{\text{next}}, \mathring{A}_i_{\text{next}}) \end{aligned}$$

with next_spec a formula such as defined in [definition 3.3.23](#).

Remark 3.3.22. *No dot operators can occur in an FO-BNN ϕ . Indeed, the only terms in the ODEs with the dot operators are variables, and all have been replaced with \circ .*

For any $A \in \mathcal{S}$, let vars_A be the sequence of the four variables $A, \mathring{A}, A_{\text{next}}, \mathring{A}_{\text{next}}$.

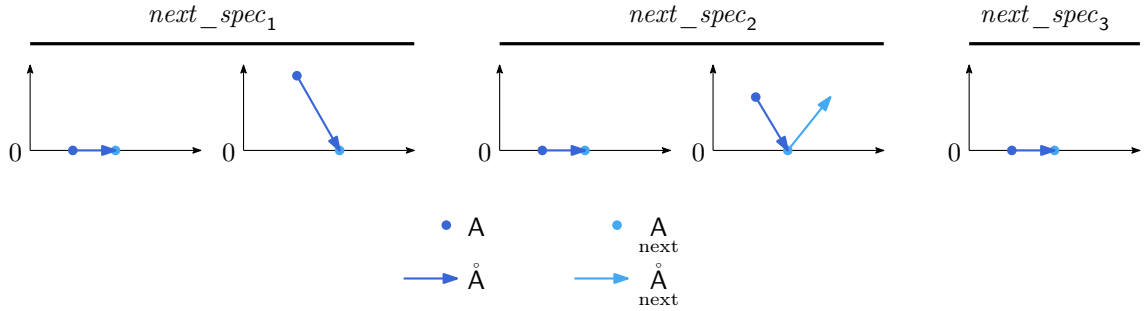
Definition 3.3.23 (Next species formula). *The formula next_spec encodes an equation that relates, for any species $A \in \mathcal{S}$, the value of the variable A_{next} to the values of the variables A, \mathring{A}*

and $\mathring{A}_{\text{next}}$. We propose three different variants:

$$\begin{aligned}
 \text{next_spec}_1(\text{vars}_{\mathbf{A}}) &=_{\text{def}} \psi(\text{vars}_{\mathbf{A}}) \\
 &\quad \wedge \left(\mathbf{A}_{\text{next}} \stackrel{\circ}{=} 0 \rightarrow (\mathbf{A} \stackrel{\circ}{=} 0 \wedge \mathring{\mathbf{A}} \stackrel{\circ}{=} 0) \vee \mathring{\mathbf{A}}_{\text{next}} \stackrel{\circ}{=} -1 \right) \\
 \text{next_spec}_2(\text{vars}_{\mathbf{A}}) &=_{\text{def}} \psi(\text{vars}_{\mathbf{A}}) \\
 &\quad \wedge \left(\mathbf{A}_{\text{next}} \stackrel{\circ}{=} 0 \rightarrow ((\mathbf{A} \stackrel{\circ}{=} 0 \wedge \mathring{\mathbf{A}} \stackrel{\circ}{=} 0) \vee (\mathbf{A} \stackrel{\circ}{=} 1 \wedge \mathring{\mathbf{A}} \stackrel{\circ}{=} -1 \wedge \mathring{\mathbf{A}}_{\text{next}} \stackrel{\circ}{=} 1)) \right) \\
 \text{next_spec}_3(\text{vars}_{\mathbf{A}}) &=_{\text{def}} \psi(\text{vars}_{\mathbf{A}}) \\
 &\quad \wedge \left(\mathbf{A}_{\text{next}} \stackrel{\circ}{=} 0 \rightarrow \mathbf{A} \stackrel{\circ}{=} 0 \right)
 \end{aligned}$$

with $\psi(\text{vars}_{\mathbf{A}}) = \left(\mathbf{A}_{\text{next}} \stackrel{\circ}{=} 1 \rightarrow (\mathbf{A} \stackrel{\circ}{=} 0 \wedge \mathring{\mathbf{A}} \stackrel{\circ}{=} 1) \vee \mathbf{A} \stackrel{\circ}{=} 1 \right)$.

We now detail the steps for obtaining an FO-BNN from a reaction network \mathcal{R} with enriched ODEs $\text{odes}(\mathcal{R})$. First, the formula $\text{odes}(\mathcal{R})$ is added. Second, a copy of $\text{odes}(\mathcal{R})$ is added, in which all variables x are replaced by x_{next} . All variables in $\mathcal{S} \cup \mathcal{S}_{\text{next}}$ are existentially quantified. Furthermore, we pick a formula $\text{next_spec}(\text{vars}_{\mathbf{A}})$ for any species $\mathbf{A} \in \mathcal{S}$. Which variant is applicable or best depends on the properties of the reaction network. In particular, each variant treats differently the cases where the value of the species is set to 0.



Additional knowledge can be included in the FO-BNN, for example, if the kinetics in \mathcal{R} are all mass-action kinetics, we can safely impose the following invariant for all the species:

$$\mathbf{A} \leq \mathbf{A}_{\text{next}}$$

because species \mathbf{A} can never become absent when it was present before. This is for example reflected in next_spec_3 that indeed states that $\mathbf{A}_{\text{next}} = 0$ only if $\mathbf{A} = 0$.

Example 3.3.24 (\mathcal{R}_{enz} : a possible FO-BNN). *Figure 3.9 gives the FO-BNN of the reaction network \mathcal{R}_{enz} with next_spec_3 . Because the kinetics of \mathcal{R}_{enz} are all mass action kinetics, we add the invariant mentioned just before.*

3.3.3.2 Sign Interpretation of FO-BNN

From the perspective of the sign abstraction of a reaction network, the variable $\mathbf{A} \in \mathcal{S}$ states whether species \mathbf{A} is present at the current time point and the variable $\mathring{\mathbf{A}}$ represents the sign of

$$\begin{aligned}
 & \exists \overset{\circ}{S}_{\text{next}} \exists \overset{\circ}{E}_{\text{next}} \exists \overset{\circ}{C}_{\text{next}} \exists \overset{\circ}{P}_{\text{next}} \exists \overset{\circ}{S}_{\text{next}} \exists \overset{\circ}{E}_{\text{next}} \exists \overset{\circ}{C}_{\text{next}} \exists \overset{\circ}{P}_{\text{next}} . \\
 & \begin{aligned}
 & \overset{\circ}{S} \overset{\circ}{=} - \mathcal{R}_{\text{on}} + \mathcal{R}_{\text{off}} \quad \wedge \quad \overset{\circ}{S}_{\text{next}} \overset{\circ}{=} - \mathcal{R}_{\text{on}_{\text{next}}} + \mathcal{R}_{\text{off}_{\text{next}}} \\
 & \wedge \overset{\circ}{E} \overset{\circ}{=} - \mathcal{R}_{\text{on}} + \mathcal{R}_{\text{off}} + \mathcal{R}_{\text{cat}} \quad \wedge \quad \overset{\circ}{E}_{\text{next}} \overset{\circ}{=} - \mathcal{R}_{\text{on}_{\text{next}}} + \mathcal{R}_{\text{off}_{\text{next}}} + \mathcal{R}_{\text{cat}_{\text{next}}} \\
 & \wedge \overset{\circ}{C} \overset{\circ}{=} \mathcal{R}_{\text{on}} - \mathcal{R}_{\text{off}} - \mathcal{R}_{\text{cat}} \quad \wedge \quad \overset{\circ}{C}_{\text{next}} \overset{\circ}{=} \mathcal{R}_{\text{on}_{\text{next}}} - \mathcal{R}_{\text{off}_{\text{next}}} - \mathcal{R}_{\text{cat}_{\text{next}}} \\
 & \wedge \overset{\circ}{P} \overset{\circ}{=} \mathcal{R}_{\text{cat}} \quad \wedge \quad \overset{\circ}{P}_{\text{next}} \overset{\circ}{=} \mathcal{R}_{\text{cat}_{\text{next}}}
 \end{aligned} \\
 & \begin{aligned}
 & \wedge \overset{\circ}{S}_{\text{next}} \overset{\circ}{=} S + \overset{\circ}{S} \quad \wedge \quad S \leq \overset{\circ}{S}_{\text{next}} \\
 & \wedge \overset{\circ}{E}_{\text{next}} \overset{\circ}{=} E + \overset{\circ}{E} \quad \wedge \quad E \leq \overset{\circ}{E}_{\text{next}} \\
 & \wedge \overset{\circ}{C}_{\text{next}} \overset{\circ}{=} C + \overset{\circ}{C} \quad \wedge \quad C \leq \overset{\circ}{C}_{\text{next}} \\
 & \wedge \overset{\circ}{P}_{\text{next}} \overset{\circ}{=} P + \overset{\circ}{P} \quad \wedge \quad P \leq \overset{\circ}{P}_{\text{next}}
 \end{aligned} \\
 & \text{with} \\
 & \begin{aligned}
 \mathcal{R}_{\text{on}} &= 1000000 \times S \times E & \mathcal{R}_{\text{off}} &= 0.2 \times C & \mathcal{R}_{\text{cat}} &= 0.1 \times C \\
 \mathcal{R}_{\text{on}_{\text{next}}} &= 1000000 \times \overset{\circ}{S}_{\text{next}} \times \overset{\circ}{E}_{\text{next}} & \mathcal{R}_{\text{off}_{\text{next}}} &= 0.2 \times \overset{\circ}{C}_{\text{next}} & \mathcal{R}_{\text{cat}_{\text{next}}} &= 0.1 \times \overset{\circ}{C}_{\text{next}}
 \end{aligned}
 \end{aligned}$$

 Figure 3.9: An FO-BNN for the reaction network \mathcal{R}_{enz} ($\text{bnn}(\mathcal{R}_{\text{enz}})$).

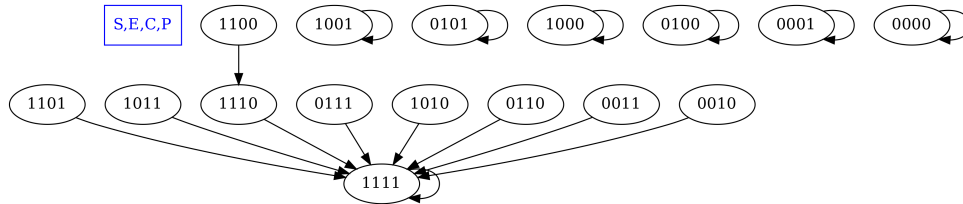
its derivative. The variable $\overset{\circ}{A}_{\text{next}}$ stands for the presence of the species A at the next time point, and similarly, $\overset{\circ}{A}_{\text{next}}$ stands for the sign of its derivative at the next time point.

Any variable assignment $\gamma : V \cup V_{\text{next}} \rightarrow \mathbb{S}$ represents a transition between abstract configurations. We denote the abstract state transition restricted on the base variables with

$$\text{trans}(\gamma) = (\gamma|_V, \gamma|_{V_{\text{next}}}) \in \mathbb{S}^V \times \mathbb{S}^V.$$

The solutions of an FO-BNN interpreted on the structure of signs can be represented as a transition graph, as defined in [definition 1.7.5](#) on page 39.

Example 3.3.25 (\mathcal{R}_{enz} : abstract simulation). *Figure 3.10 shows the abstract transition graph computed using the FO-BNN $\text{bnn}(\mathcal{R}_{\text{enz}})$ shown in figure 3.9. Note that the configurations have been restricted to the base variables only. The set of species of reaction network \mathcal{R}_{enz} has a cardinality of 4. Therefore, the transition graph of the FO-BNN of \mathcal{R}_{enz} has $2^4 = 16$ configurations.*


 Figure 3.10: Abstract simulation of \mathcal{R}_{enz} via $\text{bnn}(\mathcal{R}_{\text{enz}})$.

We can restrict the transition graph to the subgraph that is *accessible* from the abstraction of an initial configuration.

Example 3.3.26 (\mathcal{R}_{enz} : partial abstract simulation). *Figure 3.11 is the subgraph of the configurations reachable from the abstract base configuration [S:1, E:1, C:0, P:0]. It contains three configurations instead of 16.*

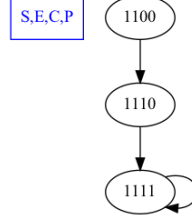


Figure 3.11: Partial abstract simulation of \mathcal{R}_{enz} via $bnn(\mathcal{R}_{\text{enz}})$.

3.3.3.3 The Soundness of the Sign Interpretation of an FO-BNN

In [NVV22], we proved that the sign interpretation of an FO-BNN $bnn(\mathcal{R})$ is sound relative to the causal successor relation $cnext_{odes}(\mathcal{R})$. All the proofs from [NVV22] are reported below and adapted to the notations used in this thesis. The general idea is the following. First, we apply John’s soundness theorem [All21] (about the abstract interpretation of first-order logic formulas) for the abstract interpretation of ODEs over signs. The John’s soundness theorem along with a sketch of the proof were briefly presented in [theorem 1.3.3](#). We show that the solutions we get by interpreting the ODE formula over signs contains the abstraction of the real solutions. Then, we show that the solutions of an FO-BNN formula built from an ODE formula and interpreted on the structure of signs contains the abstract solutions of ODEs.

Let us recall from [section 1.2.2.4](#) that the set of solutions of a formula $\phi \in \mathcal{F}_{\Sigma}(\mathcal{V})$ over a relational structure S with the same signature Σ is the set of assignments of the free variables of ϕ such that ϕ evaluates to 1. Formally: $sol^S(\phi) = \{\alpha|_{fv(\phi)} \mid \alpha : \mathcal{V} \rightarrow dom(S), \llbracket \phi \rrbracket_{\alpha, S} = 1\}$.

Recall that $h_{\mathbb{S}}^0 : \mathbb{R} \rightarrow \mathbb{S}$ is a homomorphism between relational structures with signature Σ_{arith} and that the formula $\dot{\phi}$ has the same signature Σ_{arith} for any ODE ϕ . John’s theorem thus shows:

Corollary 3.3.27. *For any ODE ϕ , it holds that*

$$h_{\mathbb{S}} \circ sol^{\mathbb{R}}(\dot{\phi}) \subseteq sol^{\mathbb{S}}(\dot{\phi}).$$

This corollary states that the set of abstract dotted states of an ODE ϕ can be overapproximated by interpreting $\dot{\phi}$ abstractly in the structure of signs. It can be used to reason about the temporal and causal next transition relations of ODEs as follows:

Lemma 3.3.28. *For any ODE ϕ , if $(\gamma_1, \gamma_2) \in next_{\dot{\phi}} \cup cnext_{\dot{\phi}}$, then $\{\gamma_1, \gamma_2\} \subseteq sol^{\mathbb{S}}(\dot{\phi})$.*

Proof.

1. [Definition 3.3.17](#) of the temporal next relation and [lemma 3.3.6](#) show that for any pair $(\gamma_1, \gamma_2) \in next_{\dot{\phi}}$ that it satisfies $\gamma_1 = h_{\mathbb{S}} \circ \alpha_1$ and $\gamma_2 = h_{\mathbb{S}} \circ \alpha_2$ for some $\alpha_1, \alpha_2 \in sol^{\mathbb{R}}(\dot{\phi})$. [Corollary 3.3.27](#) of John’s soundness theorem for abstract interpretation of logic formulas applied to ODEs shows that $h_{\mathbb{S}} \circ sol^{\mathbb{R}}(\dot{\phi}) \subseteq sol^{\mathbb{S}}(\dot{\phi})$ and thus $\{\gamma_1, \gamma_2\} \subseteq sol^{\mathbb{S}}(\dot{\phi})$.

2. If $(\gamma_1, \gamma_2) \in \mathring{cnext}_\phi$ then $\gamma_2 \in \mathring{sol}(\phi)$ by [definition 3.3.19](#). Furthermore, there exists γ'_2 such that $(\gamma_1, \gamma'_2) \in \mathring{next}_\phi$. The first property shows that $\gamma_1 \in \mathring{sol}(\phi)$ too. \square

First, let's show that all the $\mathit{next_spec}(\mathit{vars}_A)$ proposed in [definition 3.3.23](#) respect the following property:

Property 3.3.29 (*next_spec property*). For all $\gamma_1, \gamma_2 : \mathcal{S} \cup \mathring{\mathcal{S}} \rightarrow \mathbb{S}$ and all reaction networks \mathcal{R} with species in \mathcal{S} :

$$(\gamma_1, \gamma_2) \in \mathring{cnext}_{odes}(\mathcal{R}) \Rightarrow (\gamma_1 \cup_{\mathit{next}} \gamma_2)_{|\{\mathit{vars}_A\}} \in \mathring{sol}(\mathit{next_spec}(\mathit{vars}_A)).$$

Proof. $\mathit{next_spec}_1(\mathit{vars}_A)$ satisfies [property 3.3.29](#) since using causally-next relation (but this would not hold for the temporal-next relation). If all kinetic expressions are infinitely derivable, then, when a concentration becomes 0, the derivative requires an increase immediately after, in order to not become negative. If all reactions follow the mass action law, then non-zero concentrations can never become zero later on, so $\mathit{next_spec}_3(\mathit{vars}_A)$ should satisfy the requirement too. \square

In the following, we assume any formula $\mathit{next_spec}(\mathit{vars}_A), A \in \mathcal{S}$ that respect the property above.

The following theorem states the soundness of an FO-BNN $\mathit{bnn}(\mathcal{R})$.

Theorem 3.3.30 (Soundness of $\mathit{bnn}(\mathcal{R})$). $\mathring{cnext}_{odes}(\mathcal{R}) \subseteq \mathit{trans} \circ \mathring{sol}(\mathit{bnn}(\mathcal{R}))$.

Proof. Let $(\gamma_1, \gamma_2) \in \mathring{cnext}_{odes}(\mathcal{R})$. Then there exists $(\gamma'_1, \gamma'_2) \in \mathring{cnext}_{odes}(\mathcal{R})$ such that $\gamma'_1 = \gamma_1|_{\mathcal{S}}$ and $\gamma'_2 = \gamma_2|_{\mathcal{S}}$. By assumption on $\mathit{next_spec}(\mathit{vars}_A)$, this implies for all $A \in \mathcal{S}$ that $\gamma'_1 \cup_{\mathit{next}} \gamma'_2 \in \mathring{sol}(\bigwedge_{A \in \mathcal{S}} \mathit{next_spec}(\mathit{vars}_A))$. [Lemma 3.3.28](#) shows that $\gamma'_1, \gamma'_2 \in \mathring{sol}(\mathring{odes}(\mathcal{R}))$ so that $\gamma'_2 \in \mathring{sol}(\mathring{odes}_{\mathit{next}}(\mathcal{R}))$. By definition of $\mathit{bnn}(\mathcal{R})$, we obtain $\gamma'_1|_{\mathcal{S}} \cup_{\mathit{next}} \gamma'_2|_{\mathcal{S}} \in \mathring{sol}(\mathit{bnn}(\mathcal{R}))$. Hence, $(\gamma_1, \gamma_2) = (\gamma'_1|_{\mathcal{S}}, \gamma'_2|_{\mathcal{S}}) \in \mathit{trans} \circ \mathring{sol}(\mathit{bnn}(\mathcal{R}))$ as stated by the theorem. \square

3.4 Related Works on the Extraction of a Transition Graph from a Reaction Network

As we saw in [chapter 2](#), the abstraction of models of biological systems is an active field of study. Several technics have been proposed to build abstractions which aim to capture the overall dynamics of the original model. In this section, we focus on abstractions from which we can retrieve an abstract transition graph for a given reaction network.

Mover et al. [Mov+21] develop an efficient implicit method for the exact abstraction of dynamical systems, whose abstract configuration space description and ODE dynamics are restricted to be systems of polynomial equations. While polynomial equations allow the exact description of abstract configuration spaces that are more general and fine-grained than the ones used for Boolean networks, they do not provide enough expressiveness to describe the kinetic expressions frequently used for the modelling of chemical reaction networks.

In [FS08a], the authors build an asynchronous Boolean transition system based on the reactions of the reaction network. Whenever a reaction $A + B \rightarrow C$ exists, the configurations 110 and 111 (A and B present, C don't care) are connected to the following configurations: 111, 101, 011, 001 (C for sure present, A and/or B might be fully consumed). This has been proven to be an overapproximation of the quantitative dynamics of the reaction network. It has been used for model checking [CFS06]. However, the overapproximation is so big that we often can not really derive any useful information to build Boolean networks from it.

It is important to know that different abstractions often lead to conflicting conclusions concerning the dynamics [PHD09]. In particular, when the model consists of non-linear equations with a switch-like behaviour (negligible activity below a threshold, rapid increase near a threshold and level off for higher values), it can be approximated with Piecewise Linear (PL) differential functions [GK73]. The dynamics of a PL model can be studied qualitatively by the mean of a transition graph (TG) constructed as follows: the phase space is divided into qualitative states (regions) according to some given thresholds and there is a transition between two qualitative states if a solution starting in the first region reaches the second region, without passing through a third region. It was shown that qualitative simulation using this transition graph is sound but incomplete. In turn, a PL model can be approximated by a multivalued network (model of Thomas) [Tho73]. Such a model is really similar to a Boolean network, but it relies on discrete functions which describe the influences among the species of the system. Also, the overall dynamics is obtained by updating the status of one species at each timestep. The resulting dynamics is represented by an asynchronous transition graph. Despite the close correspondence between PL equations and models of Thomas [Sno89], it was shown that paths and long-term dynamics do not always agree between the transition graph of a model of Thomas and the transition graph of a PL model [JSB13].

Our abstract simulation approach is different from the ones mentioned above. In particular because (1) it focuses on the Boolean abstraction and does not need to find a set of thresholds for each species; (2) the resulting transition graph we obtain is a sound overapproximation of the numerical Euler simulation of the ODEs.

3.5 Summary

3.5.1 Wrap-up

The goal of the chapter was to retrieve the structure and the dynamics of a given reaction network, so that they can be used in the next chapter to synthesise Boolean networks.

Our strategy to retrieve the structure was presented in [section 3.2](#). We parse the elements of the models to build an influence graph that stores the direct influences among the species of the models.

As for the dynamics, we used the differential semantics of the reaction networks. We saw that an ODE system defines a reachability relation between the configurations of the system. We refer to this relation as the *concretely following* relation. When we have an analytical solution, we can compute part of this relation exactly. Without having access to an analytical solution, it is still possible to *approximate* the concretely following relation using simulation. We presented two approaches. The first approach consists of a concrete numerical simulation followed by a

binarisation step. We do not use the Euler algorithm since it does not balance very well the trade-off between accuracy and efficiency of the computation. Instead, we rely on algorithms from tools tailored for the simulation of reaction networks. As for the second approach, referred to as abstract simulation, it consists of a novel method that characterises the qualitative behaviour of a reaction network abstractly, without any exact knowledge of the initial concentrations. It relies on the *abstract causally next* relation of the ODEs, rather than on the *abstract temporally next* relation. We saw that the relationship between these two relations is linked to approximation errors of Euler’s numerical simulation, and that our approach is sound, yet imprecise. However, in contrast to concrete simulation, it can for now only be used on core reaction networks.

3.5.2 Discussion and Perspectives

3.5.2.1 About the choice of focusing on the differential semantics

Many things are to be discussed about the way we retrieve the structure and the dynamics of a reaction network. One of the things I really want to attract attention to concerns the way we retrieve the dynamical data, and particularly the choice of focusing on the differential semantics of reaction networks to retrieve the dynamics instead of the stochastic, discrete or Boolean semantics. The main reason for that is that the differential semantics is deterministic. It smoothes out the fluctuations (noise) by averaging the behaviour of an infinite number of stochastic simulations [Kur70]. The simulation of an ODE system is thus, in a sense, the idealised representation of what happens under the hood.

Of course, the underlying system might express variability; fluctuations arise naturally in biological systems, even with a fixed reaction network. This was particularly highlighted in recent years through single-cell analyses. Among the typical examples where stochasticity plays a crucial role, let us mention the synaptic plasticity, which is known to be involved in learning and memory [MG20]. At a cellular level, it results from the modulation of information flow between neurons. At a molecular level, it results from reactions taking place at the junction of two neurons: the synapses. The volume in which these reactions take place (and hence the number of molecules involved) is so small¹⁴ that discreteness has to be taken into account for correct analysis. In this particular context, the stochasticity induced was shown to be the reason of efficient information transfer [Fuj+17]. As for other examples, we can mention basically any processes involving a small number of molecule copies, such as protein–DNA binding, transcription and translation

In the context of this thesis, fluctuations would be cumbersome to handle. Of course, we could use extensions of Boolean networks that have been proposed to take variability into account [LH12], but the classic Boolean networks (the ones we synthesise in this thesis) are deterministic by nature, in the sense the update of a given species in a given configuration will always produce the same configuration. Note that this has nothing to do with the choice of update scheme which only accounts for different updating periods of the species.

Another characteristic of the differential semantics of reaction networks is that it washes away structural ambiguity of the underlying mechanisms. By this, we mean that the same set of ODEs can be produced from different reaction networks that will thus present the same behaviour

¹⁴Typically from $0.01 \mu\text{m}^3$ to $0.8 \mu\text{m}^3$ of the dendritic spine head, while the volume of the cell body is typically $5000 \mu\text{m}^3$, hence 10^4 -fold bigger.

when studied with the differential semantics. This contrasts with the stochastic semantics which requires the correct structure.

Example 3.5.1. *The following system of ODEs*

$$\begin{aligned}\dot{A} &\doteq -k \times A \\ \wedge \dot{B} &\doteq k \times A\end{aligned}$$

can be induced by (at least) these two reaction networks:



The system of ODE mentioned earlier can be directly produced from \mathcal{R}_1 , and from \mathcal{R}_2 , it can be produced after simplifying the following:

$$\dot{A} \doteq \underbrace{-k \times A}_{\text{from } \mathcal{R}_{2.1}} + \underbrace{k \times A - k \times A}_{\text{from } \mathcal{R}_{2.2}}$$

These two reaction networks thus have the same deterministic behaviour (thus the same mean stochastic behaviour) but not the same local stochastic behaviour.

In this thesis, we used models from BioModels. However, it is not unlikely that some of these models fall into what Kohl et al. refer as the *plausibility trap* [Koh+10]. That is, the fact that just because a model correctly predicts observed behaviour, this does not mean the mechanisms it posits—hence, its structure—are involved in the system under study.

Indeed, most of the models in BioModels were in fact published as ODEs built such that they fit given dynamical observations. They were unpacked as reaction networks only for their submission to BioModels, using the differential semantics. But in fact, a reaction network is only uniquely defined under very specific conditions [SH10]. Unpacking a given ODE model as a reaction network is a difficult problem because ODEs are ambiguous [CP08]. Automatic procedures have been proposed for this task, when the ODE uses mass action kinetics in [HT79] and with general kinetics in [FGS15].

For the models in BioModels, this conversion is made by hand most of the time, and the curation process of BioModels does not ensure the validity of the models in terms of structure. It only requires the model to be able to reproduce the figures from the original publication. Since a lot of the models were actually studied with the differential semantics, we think it is safer to use the differential semantics as well to retrieve the dynamics.

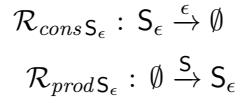
However, because of the way we retrieve the prior influence graph and perform the abstract simulation, we do not benefit from the ODEs washing away the structure ambiguity: the structure of the underlying reaction network will play a crucial role in the conversion. Thus we can only hope for the structure to be correct.

3.5.2.2 About the Binarisation

In the method presented in [section 3.3.2](#), the values obtained after the concrete simulation step are continuous while we need them to be binarised so we can apply the Boolean network synthesis procedure. We proposed to use the midrange procedure, but many other binarisation procedures exist. However, it is hard to pick the appropriate one automatically. In our context, *appropriate* means that ideally, we would like the binarised values to preserve the causation of the events that produce the dynamics we obtain.

In the method presented in [section 3.3.3](#), the underlying binarisation of the abstract simulation is the “zero/non-zero” binarisation. By doing so, we showed that we preserve the causation of the events. However, one may argue that this is too drastic and does not carry a lot of information. We started to investigate this problem in [NVV22]. The idea is to add phantom species such that their derivative equals the species of interest minus a given threshold. We show this on a simple example.

Example 3.5.2 (\mathcal{R}_{enz} : abstract simulation with threshold on S). We start with \mathcal{R}_{enz} . Let’s say that we would like to know whether the concentration of S is above, equal, or below a given threshold $\epsilon > 0$, say $\epsilon = 0.3$. This is the same as asking whether $S - \epsilon < 0$. We add the species S_ϵ to the reaction network, such that $\dot{S}_\epsilon = S - \epsilon$. This can be done by adding the following two reactions:



We can run the abstract simulation algorithm on the ODE system of the extended reaction network. The system contains the equation $\dot{S}_\epsilon = S - \epsilon$ in addition of the same ODEs than before. When S_ϵ is negative (\searrow), it means that $S - \epsilon = \searrow \Leftrightarrow S < \epsilon$. Conversely, a positive sign $S - \epsilon = \nearrow$ means $S > \epsilon$, and $S - \epsilon = \rightarrow$ means $S = \epsilon$. The abstract simulation starting from the initial abstract configuration set to [S:1, E:1, C:0, P:0, $S_\epsilon:\searrow$], yields to the transition graph in [figure 3.12](#). In this picture, we write $S - \epsilon$ instead of \dot{S}_ϵ .

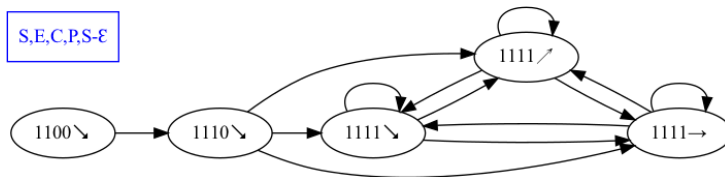


Figure 3.12: Abstract simulation of \mathcal{R}_{enz} with one threshold (on S).

Unfortunately, in this small example, not much can be concluded. This is due to the overapproximation made by the abstract interpretation. This leads to the following perspective.

3.5.2.3 Better Approximation of *cnext* Using Abstract Interpretation

We saw that our abstract simulation is only an overapproximation of the abstract causally next relation. One open question is whether one can compute the abstract causally next relation *exactly*, or at least whether more accurate approximations may be achieved.

This question is related to the exact computation of the Boolean abstraction of linear equation systems, explored for example in [ANV21]. We also started to investigate the matter in [NVV22]. The proper addition of thresholds, combined with the utilisation of the exact Boolean abstraction algorithm from [ANV21] considerably leads to more fine-grained abstract simulation of the input reaction network. The automatic application of the exact Boolean abstraction algorithm to the simulation of Boolean networks with thresholds require however an extension of the algorithm to the inhomogeneous case, which is under implementation. Hence, for now, we have manually rewritten the inhomogeneous equations into some homogeneous ones for the algorithm to be applied.

Example 3.5.3. We continue [example 3.5.2](#). With the addition of a further threshold for P and of an upper bound on the sum of the initial concentrations of S, C, P , the abstract simulation allows us to conclude that only one final state may be reachable during the abstract simulation, where the concentration of S is below the given threshold, as shown in [figure 3.13](#).

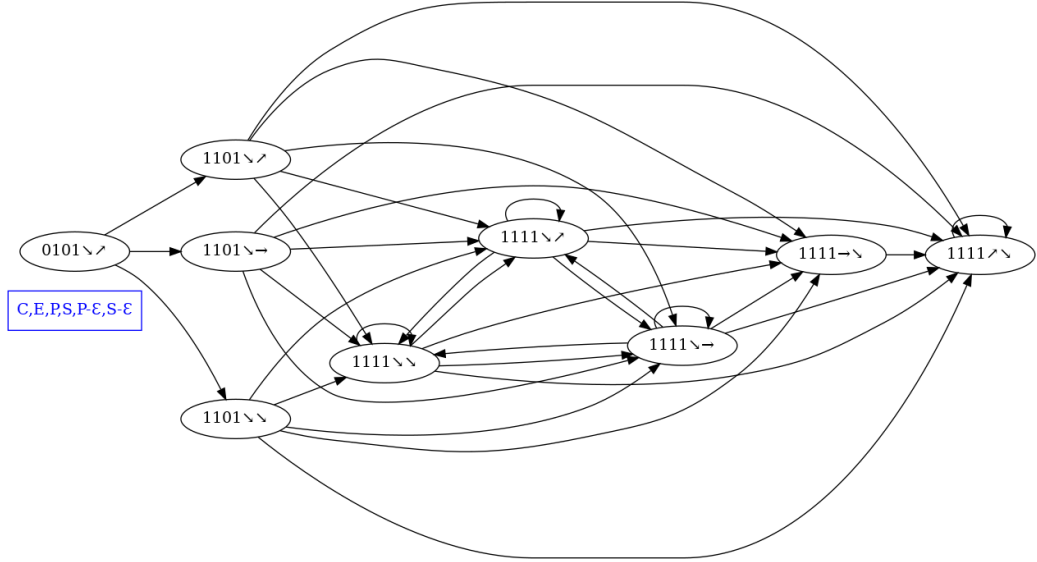


Figure 3.13: Abstract simulation of \mathcal{R}_{enz} with two thresholds (on P and S).

3.5.2.4 Process More Reaction Networks

Another perspective would be to be able to process more reaction networks. (1) First, we want to be able to process extended reaction networks. However, they make use of various elements, and even the dedicated tools sometimes struggle to implement them. For example, not all simulators allow the use of discontinuous events. Our abstract simulation also does not take it into account either. We think a better formal semantics of these extended reaction networks would help, and we started to look in this direction with Joachim Niehren and Cedric Loushaine. (2) Another future work would be to lift the pipeline to process reaction networks for which the kinetic expressions are only partially known. The most frequent case is that some parameters of the kinetic expressions are unknown. Alternatively, the form of the kinetic expressions may be known

only up to similarity [Nie+16; ANV19]. Such networks cannot be simulated concretely without estimating the missing kinetic information from data, so abstract simulation may provide an interesting alternative to retrieve the qualitative behaviour of such networks.

3.5.2.5 Make Use of More Knowledge and Data

First, we could take more dynamical patterns into account such as fixed points and cyclic attractors. There are some tools dedicated to the numerical computation of the attractors of a reaction network [Hoo+06]. However, we could also investigate ways to compute them abstractly, with our FO-BNNs.

Second, for now, the duration of each numerical simulation has to be provided or retrieved manually from publication. The procedure is error-prone, but we could extract the information automatically from Simulation Experiment Description Markup Language (SED-ML) documents, if available. These documents describe the models to use, the modifications made to those models, the order in which all simulation procedures were applied, how the raw results were processed and a description of the final output. They implement what the Minimum Information About a Simulation Experiment (MIASE) standard proposes as a minimal set of information for the simulations to be reproducible.

Chapter 4

Boolean Network Synthesis from Given Structure and Dynamics

Contents

4.1	Introduction	103
4.2	Presentation of ASK&D-BN	105
4.2.1	Step 1: Binarisation of the Given Dynamics	105
4.2.2	Step 2: Synthesis of Transition Functions	106
4.2.3	Step 3: Global Boolean Network Assembly	118
4.2.4	Evaluation of the Synthesised Boolean Networks	118
4.3	Related Works	119
4.3.1	Non-redundant Boolean Networks	120
4.3.2	Fitting the Structure	121
4.3.3	Fitting the Dynamics	121
4.3.4	Partial Good-Enough versus Exhaustive Optimal Synthesis	124
4.3.5	Simplicity	125
4.4	Evaluation of ASK&D-BN	126
4.4.1	Boolean Network Synthesis Methods for the Comparison	126
4.4.2	Datasets	126
4.4.3	Evaluation Procedure	128
4.4.4	Results on the Two Real Datasets	129
4.4.5	Results on the Generated Datasets	129
4.5	Wrap-Up	132

4.1 Introduction

The current chapter is dedicated to *how* to synthesise Boolean networks compatible with a structure and a dynamics. We present the method ASK&D-BN (read “asked B-N”). The name

stands for Automatic Synthesis of Boolean Networks compatible with given structure Knowledge and dynamics Data. The first version of the method was introduced in [VBS21a].

As presented in [section 1.8](#) on page 39, a Boolean network is seen as a set of transition functions (one per species). Its structure is defined as an *influence graph*, which gives the influences of the species on one another. Its dynamics is given by a *transition graph*, which yields the possible changeovers between the different configurations the Boolean network can take, according to the chosen update scheme.

The notion of *compatibility* used in this chapter is closely related to the one given in [chapter 2](#). As seen there, various strategies have been proposed to synthesise Boolean networks (and models in general). However, all roughly consist in enforcing the influence graph and the transition graph of the synthesised Boolean networks to contain specific edges, those “allowed” by the given structure (known influences between the species) and those “required” by the given dynamics (known behaviour of the species).

In this chapter, the structure and dynamics are assumed to be given, but they are not necessarily obtained from a reaction network. Nevertheless, the constraints used by ASK&D-BN are tailored for this specific use case. This is different from the existing Boolean network synthesis methods. In particular, when dealing with timeseries, we assume that they are *complete* (i.e., without missing timesteps). Moreover, despite the structure constraint, the Boolean network synthesis problem is usually largely under-constrained: there is usually only one timeseries available, which usually does not visit much of the system’s dynamic landscape. We thus focus on the synthesis of Boolean networks with *cardinal-minimal* DNFs: the smallest DNFs compatible with the *partial* dynamic observations. With this constraint, we can avoid the synthesis of too many Boolean networks. However, if we are willing to synthesise many Boolean networks, it is of course possible to relax the mincard constraint and synthesise *all* the minDNFs instead.

For example, the reaction network model of enzymatic reaction for the running example [example 1.9.4](#), there are $2^{2^4} = 65\,536$ possible functions a priori (hence much more subset-minimal DNFs!) of which 128 satisfy the influence graph in [example 3.2.1](#). Using the sequence of configurations in [figure 3.11](#), this number can be reduced to 14 Boolean networks with subset-minimal DNFs. Unfortunately, it is quite a difficult task to analyse too many Boolean networks (even though some works go in that direction [Che+20]). Thanks to the mincard constraint, we can reduce the number of Boolean networks to 1.

In this chapter, the evaluation of ASK&D-BN mostly consists of sanity check experiments, but in-depth experiments about Boolean network synthesis from structure and dynamics directly retrieved from reaction network models are presented in the next chapter ([chapter 5](#)).

Outline In [section 4.2](#), we present our method ASK&D-BN. In [section 4.3](#), we discuss some related work and justify some modelisation choices made when designing ASK&D-BN. In [section 4.4](#), we conduct experiments to compare ASK&D-BN with three state-of-the-art algorithms for the same task, namely REVEAL, Best-Fit and Caspo-TS. We conclude in [section 4.5](#) with some additional discussion and perspectives.

4.2 Presentation of ASK&D-BN

As mentioned earlier, ASK&D-BN synthesises a set of Boolean networks *compatible* with a given structure and a given dynamics. The structure is given as an influence graph ([definition 1.7.7](#) on page 39) and the dynamics either as a (partial) truth table ([definition 1.6.1](#) on page 29) or a timeseries ([definition 2.4.6](#) on page 61), which can be continuous or binarised.

The notion of compatibility used by ASK&D-BN is closely related to what we presented in [chapter 2](#). A Boolean network synthesised by ASK&D-BN obeys the next two principles:

Structure compatibility: its influence graph is a spanning subgraph of the input structure.

Dynamics compatibility: its general-asynchronous transition graph contains as many transitions of configurations retrieved from the input dynamics as possible.

ASK&D-BN is composed of three steps, summarised in [figure 4.1](#):

- (1) The binarisation of the provided dynamical data, when it consists of a continuous timeseries. This is detailed in [section 4.2.1](#).
- (2) The *local synthesis of transition functions* for each species, such that they comply with the given structure and dynamics. This is the subject of [section 4.2.2](#).
- (3) The *global Boolean network(s) assembly*, that consists in the generation of all the Boolean networks from the functions synthesised in the second step. This is presented in [section 4.2.3](#).

Note that the Boolean network synthesis is done *species-wise*. However, the notion of compatibility has only been defined at the Boolean network level, and not at the species level. To cope with this issue, we will introduce in step 2 the notion of *local compatibility* of a transition function with a structure and a dynamics. Then, in step 3, we will combine locally compatible transition functions to reach compatible Boolean networks.

At the end of the synthesis, ASK&D-BN summarises the quality of the synthesised Boolean networks using criteria presented in [section 4.2.4](#).

4.2.1 Step 1: Binarisation of the Given Dynamics

This step only applies when the given dynamics is not already binarised. That is, in the case of continuous timeseries. In this case, ASK&D-BN applies a binarisation procedure to get the corresponding Boolean observations.

Let \mathcal{S} a set of species and \mathcal{T} a continuous timeseries. For each species $X \in \mathcal{S}$, its binarisation threshold θ_X is either given, or computed as the midrange of the observations of X :

$$\theta_X = \frac{\min + \max}{2}$$

where min and max are the minimum and maximum values of X observed in \mathcal{T} . Then, with X_t the value of X at time t in the timeseries, the binarised value \hat{X}_t of X at time t is

$$\hat{X}_t = \begin{cases} 1 & \text{if } X_t \geq \theta_X \\ 0 & \text{otherwise.} \end{cases}$$

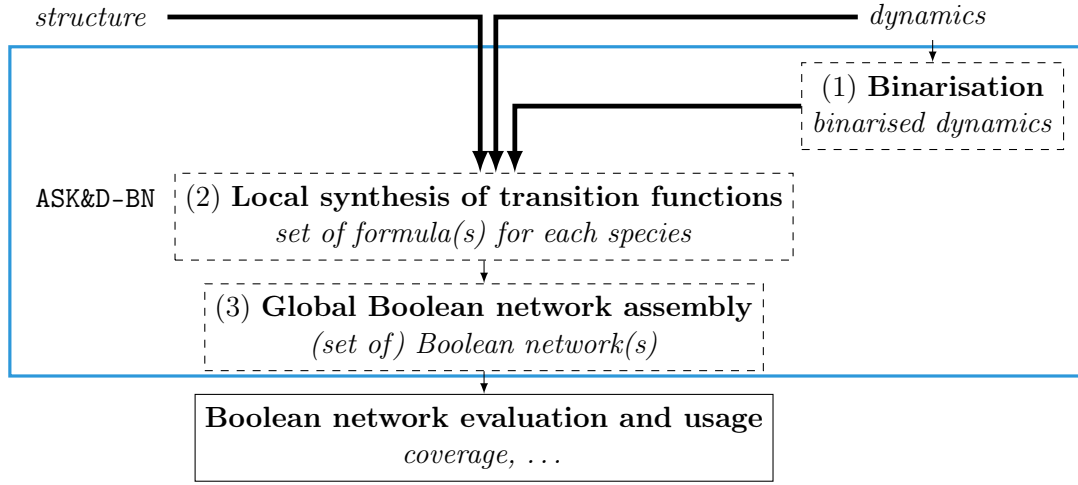


Figure 4.1: Workflow of ASK&D-BN (blue box). The local synthesis takes the following inputs (thick arrows): a structure (influence graph) and a dynamics (either a truth table or a timeseries and its binarisation). Once ASK&D-BN has terminated, we have a set of Boolean networks compatible with the given structure, and explaining the dynamics as well as possible. These Boolean networks can then be evaluated and used for further analyses.

4.2.2 Step 2: Synthesis of Transition Functions

In ASK&D-BN, this step is done species-wise: for each species, the goal is to construct *all* the simplest transition functions that are compatible with the given structure, and that explain the dynamic data with as little error as possible.

The general principle of this step is the following. For a given species X , ASK&D-BN searches among all possible transition functions f_X for X . The candidates that do not respect the given structure constraints are ruled out. Then, each remaining candidate is evaluated on its ability to reproduce the given observations. It might be that no candidate function can explain *all* the given dynamical data. In this case, we aim to minimise the error of the candidate functions with respect to the data.

When the given dynamics is a truth table, the error of a candidate function f_X is the number of lines where the truth table of f_X disagrees with the given truth table. When the given dynamics is a timeseries, the error of f_X is weighted by “how far” the continuous values at the unexplained timesteps are from the binarisation threshold. Finally, among the candidates that have the smallest error, we select the simplest ones, that is, those that use the smallest number of influences.

In [algorithm 1](#), we give a general procedure to solve the transition function synthesis problem. The algorithm returns, for each species, the exhaustive set of the simplest transition functions that minimises the error regarding the given dynamics, and respects the given structure. This problem consists in both a combinatorial problem (structure constraint) and an optimisation problem (dynamics and minimality constraints). The Answer-Set programming framework (ASP) provides a convenient declarative language to describe and solve this kind of problems. The constraints are encoded in logic, and thanks to heuristics (inspired from SAT solvers), an ASP

Algorithm 1 Naive imperative algorithm for the local synthesis of ASK&D-BN.

Require: a species \mathbf{X} , structure knowledge \mathcal{P} , dynamical data \mathcal{T}

```

1: function LOCAL_SYNTHESIS( $\mathbf{X}$ ,  $\mathcal{P}$ ,  $\mathcal{T}$ ,  $\text{dnf} \in \{\text{mincard}, \text{all}\}$  )
2:   if  $\text{dnf} = \text{all}$  then
3:      $D \leftarrow \text{all\_completion}(\mathcal{T})$ 
4:   else
5:      $D \leftarrow \{\mathcal{T}\}$ 
6:   for  $d$  in  $D$  do
7:      $\text{min}_e \leftarrow \infty$  ▷ stores the smallest error computed so far
8:      $\text{min}_l \leftarrow \infty$  ▷ stores the smallest cardinality seen so far
9:      $C \leftarrow \text{generate\_candidates}(\mathbf{X}, \mathbf{X}.\text{parents})$ 
10:    ▷ all combinaisons of all possible conjunctions on  $\mathbf{X}.\text{parents}$ 
11:    for  $c$  in  $C$  do
12:      if  $\text{compatible}(c, \mathcal{P})$  then
13:         $c.\text{error} = \text{compute\_error}(c, d)$ 
14:        if  $c.\text{error} < \text{min}_e$  then
15:           $\text{min}_e \leftarrow c.\text{error}$ 
16:           $\text{min}_l \leftarrow \text{len}(c.\text{literals})$ 
17:        else if  $c.\text{error} = \text{min}_e \wedge \text{len}(c.\text{literals}) < \text{min}_l$  then
18:           $\text{min}_l \leftarrow \text{len}(c.\text{literals})$ 
19:    yield [ $c$  for  $c$  in  $C$  if  $c.\text{error} = \text{min}_e \wedge \text{len}(c.\text{literals}) = \text{min}_l$ ]
20:    ▷ candidates with smallest error and the smallest number of literals

```

solver performs a *clever* exhaustive search of all the solutions that satisfy the given constraints. ASP is very powerful and quite straightforward to use for implementing the transition function synthesis step.

In the following, we detail the constraints and describe their encoding in ASP. We split the explanations into five steps:

1. (section 4.2.2.1) The encoding of the given structure and dynamics.
2. (section 4.2.2.2) The generation of the search space.
3. (section 4.2.2.3) The hard structure constraint.
4. (section 4.2.2.4) The fit of the candidates to the dynamics.
5. (section 4.2.2.5) The minimisation of the size of the candidate.

Recall that the Boolean network synthesis is done *species-wise*. However, the notion of compatibility has only been defined at the Boolean network level (section 4.2), and not at the species level. To cope with this issue, we will introduce the notion of local compatibility of a transition function with a structure and a dynamics. Then, we will combine locally compatible transition functions to reach compatible Boolean networks.

4.2.2.1 Encoding the Local Structure and Dynamics

In this section, we explain how to encode the *local* structure and dynamics in ASP, i.e., the subset of the given structure and dynamics that relates to a given species. Recall that the global structure is given as an influence graph (definition 1.7.7 on page 39), and the global dynamics as either a (partial) truth table (definition 1.6.1 on page 29) or a (binarised) timeseries (definition 2.4.6 on page 61).

Local Structure

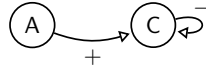
Given an influence graph $\mathcal{P} = (\mathcal{S}, E)$ and a species $X \in \mathcal{S}$ the *local structure* relatively to X is given by the local influence graph \mathcal{P}_X defined as follows.

Definition 4.2.1 (Local influence graph). *Let X be a species. The local influence graph \mathcal{P}_X is a subgraph of \mathcal{P} defined over $\{X\} \cup \text{parents}(X)$ and which contains every edge of \mathcal{P} of the form $Y \xrightarrow{s} X$ where Y is a parent of X .*

In ASP, each node Z of \mathcal{P}_X is encoded with a predicate `nodeID(Z)`, and each of its edges $Y \xrightarrow{s} X$ with a predicate `sig(Y, X, v)`, where

$$v = \begin{cases} +1 & \text{if } s \in \{+, \pm\} \\ -1 & \text{if } s \in \{-, \pm\} \end{cases}$$

Example 4.2.2 (Encoding the local structure in ASP). *We consider the influence graph \mathcal{P} shown in example 2.4.1 on page 59 and the species C . According to \mathcal{P} , the local structure relative to C is:*



The following predicates encode the graph above:

```

1 nodeID(a). nodeID(c).
2 sig(a, c, 1). sig(c, c, -1).
```

Local Dynamics

The local dynamics relative to a species X , is the subset of data concerning the following set of species: $\mathcal{I} = \{X\} \cup \text{parents}(X)$. If the dynamics is given as a timeseries, the ASP encoding consists of predicates `obs(t, Y, v)` which give the value v , stretched between -100 and 100, of each species $Y \in \mathcal{I}$ at time t . Similarly, for a binarised timeseries, we use predicates `obsBinarised(t, Y, v)` which give the binarised value (either -1 or 1) of each species $Y \in \mathcal{I}$ at time t . As for the case where the dynamics is given as a (partial) truth table, we use a set of predicates `ttin(i, Y, v)` and `ttout(i, Y, v)`, which give the value v (either 1 or -1) of the species $Y \in \mathcal{I}$ in the i -th line of the truth table.

Example 4.2.3 (Encoding of a timeseries and its binarisation in ASP). *The following predicates encode the first few steps of the times series in example 2.4.7 on page 61, along with its binarisation.*

```

1 time(1). time(2). time(3).
2 obs(1, a, -100). obs(1, b, 100). obs(1, c, -100).
3 obs(2, a, -94). obs(2, b, 72). obs(2, c, -46).
4 obs(3, a, -86). obs(3, b, 28). obs(3, c, -28).
5 obsBinarised(1, a, -1). obsBinarised(1, b, 1). obsBinarised(1, c, -1).
6 obsBinarised(2, a, -1). obsBinarised(2, b, 1). obsBinarised(2, c, -1).
7 obsBinarised(3, a, -1). obsBinarised(3, b, 1). obsBinarised(3, c, -1).

```

Example 4.2.4 (Encoding a truth table in ASP). *The following predicates encode the truth table given in [example 1.6.9](#) on page 33.*

```

1 ttin(0, a, -1). ttin(0, b, -1). ttin(0, c, -1). ttout(0, x, -1).
2 ttin(1, a, -1). ttin(1, b, -1). ttin(1, c, 1). ttout(1, x, 1).
3 ttin(2, a, -1). ttin(2, b, 1). ttin(2, c, -1). ttout(2, x, 1).
4 ttin(3, a, -1). ttin(3, b, 1). ttin(3, c, 1). ttout(3, x, 1).
5 ttin(4, a, 1). ttin(4, b, -1). ttin(4, c, -1). ttout(4, x, 1).
6 ttin(5, a, 1). ttin(5, b, -1). ttin(5, c, 1). ttout(5, x, 1).
7 ttin(6, a, 1). ttin(6, b, 1). ttin(6, c, -1). ttout(6, x, 1).
8 ttin(7, a, 1). ttin(7, b, 1). ttin(7, c, 1). ttout(7, x, -1).

```

4.2.2.2 Search Space Generation

At this step, we generate the set of all candidate transition functions for a given species $X \in \mathcal{S}$. This set is finite since it consists of all the *non-redundant* DNF expressions one can construct from the set $\text{parents}(X)$ ([definition 1.2.12](#) on page 16). First, we discuss how we represent a non-redundant DNF expression, then we show how the formalisation translates to ASP, and how we ask ASP to consider each candidate.

Let \mathcal{V} be a set of variables. A non-redundant DNF over \mathcal{V} is represented as a set of elementary conjunctions with variables in \mathcal{V} . Each elementary conjunction consists of an assignment $c_{\mathcal{V}} : \mathcal{V} \rightarrow \{-1, 1, 0\}$, which encodes the polarity of each variable $v \in \mathcal{V}$. That is, whether v appears negatively, positively, or does not appear in the elementary conjunction. Note that, by construction, $c_{\mathcal{V}}$ is satisfiable because it cannot contain a literal and its negation.

Example 4.2.5 (Elementary conjunction as assignment). *Let $\mathcal{V} = \{a, b, c\}$. The assignment $(a : 1, b : 1, c : 0)$ represent the elementary conjunction $a \wedge b$. The assignment $(a : 1, b : -1, c : 1)$ represents the elementary conjunction $a \wedge \neg b \wedge c$.*

Example 4.2.6 (DNF expression as set of assignments). *Let $\mathcal{V} = \{a, b, c\}$. The set $\{(a : 1, b : 1, c : -1)\}$ represent the DNF $a \wedge b \wedge \neg c$ and $\{(a : 1, b : 1, c : -1), (a : 1, b : 0, c : 0)\}$ represent the DNF $(a \wedge b \wedge \neg c) \vee (a)$.*

The DNF represented by the empty set states that the component under study is constant, but the value (either always 1 or always 0) is left non-determined. This differs from the usual interpretation in logic, where the empty set represents only the constant 0.

To represent a DNF in ASP, we use a set of predicates `conjTaken(i, Y, v)` which state the polarity v of each species Y in the conjunction with ID i . Then, we use a choice rule to generate

all the possible candidates DNFs from the $3^{|\text{parents}(X)|}$ possible elementary conjunctions. This number is stored in `maxNbPossibleConj`. As for the possible elementary conjunctions are given by a set of predicates `possibleConj/3` (with the same arguments as the predicate `conjTaken/3`). In the current implementation, we generate the predicates `possibleConj/3` with a Python program from the list of parents of the component under study. Generating them directly in ASP is a future work.

```

1 1{conjIDTaken(0..maxNbPossibleConj)}; % (choice rule)
2 conjTaken(I, X, V) :- possibleConj(I, _, _); conjIDTaken(I).

```

Example 4.2.7 (A candidate transition function). According to the influence graph in *example 4.2.2*, the species *C* has two parents: *A* and itself. Thus, there are $3^2 = 9$ possible elementary conjunctions. They are encoded by the following predicates:

```

1 #const maxNbPossibleConj=9.
2 % 0:constant      1:a      2: not a
3 possibleConj(0, a, 0). possibleConj(1, a, 1). possibleConj(2, a, -1).
4 possibleConj(0, c, 0). possibleConj(1, c, 0). possibleConj(2, c, 0).
5 % 3:c      4: not c      5: a and c
6 possibleConj(3, a, 0). possibleConj(4, a, 0). possibleConj(5, a, 1).
7 possibleConj(3, c, 1). possibleConj(4, c, -1). possibleConj(5, c, 1).
8 % 6: a and not c      7: not a and c      8: not a and not c
9 possibleConj(6, a, 1). possibleConj(7, a, -1). possibleConj(8, a, -1).
10 possibleConj(6, c, -1). possibleConj(7, c, 1). possibleConj(8, c, -1).

```

If the choice rule has produced `conjIDTaken(6)` and `conjIDTaken(8)`, the candidate DNF is $(A \wedge \neg C) \vee (\neg A \wedge \neg C)$.

The empty conjunction is always given the ID 0, so we can enforce the choice rule not to pick it along another elementary conjunction:

```

1 :- conjIDTaken(0); conjIDTaken(N); N != 0.

```

We now have generated a finite search space of candidate transition functions. However, some of these candidates might be encoded by a DNF expression that is not minimal (*definition 1.6.7*). We postpone this problem to *section 4.2.2.5*, where we will show how we enforce the synthesis of transition functions in minimal DNF. For now, we will tackle another issue, namely, that the search space may contain transition functions that are not compatible with the input structure. In the next section, we thus add constraints to discard these incompatible candidates.

4.2.2.3 Structural Constraint

What are the best functions regarding the given structure?

Global Structure Compatibility

Recall that we intend to synthesise Boolean networks whose respective structure is compatible with the given structure. That is, the influence graph of each synthesised Boolean network has to

be a spanning subgraph of the given influence graph \mathcal{P} . Since we work species-wise, we introduce the notion of *local compatibility* of a transition function with respect to \mathcal{P} .

Local Structure Compatibility

Let $\mathcal{P} = (\mathcal{S}, E)$ be the given influence graph, X a species from \mathcal{S} , f_X a candidate transition function for X represented by a DNF E . We say that f_X is locally compatible with \mathcal{P} if it only involves influences that are allowed by \mathcal{P} . Formally, the notion of local compatibility is based on the syntax of the DNF expression E that encodes f_X . For each conjunction $c_{\mathcal{P}(X)}$ in E , we want that, for each parent Y of X ,

$$(c_{\mathcal{P}(X)}(Y) \neq 0) \implies \left(Y \xrightarrow{c_{\mathcal{P}(X)}(Y)} X \in \mathcal{P} \right).$$

A Boolean network is locally compatible with \mathcal{P} if each of its transition function f_X is locally compatible with \mathcal{P} . Enforcing the local compatibility species-wise implies the global compatibility of each synthesised Boolean network \mathcal{B} as the influence graph of \mathcal{B} will be a spanning subgraph of the input influence graph \mathcal{P} .

Translated to ASP, we have:

```
1 interactionUsed(ParentID, x, V) :- conjTaken(ConjID, ParentID, V); V!=0.
2 :- interactionUsed(ParentID, x, V); not pig(ParentID, x, V).
```

Example 4.2.8. We continue [example 4.2.7](#). The candidate DNF $(a \wedge \neg c) \vee (\neg a \wedge \neg c)$ would be ruled out because the rules above generate the following predicates:

```
1 % because of conjIDTaken(6): % because of conjIDTaken(8):
2 interactionUsed(a, c, 1).      interactionUsed(ParentID, c, -1).
3 interactionUsed(c, c, -1).      interactionUsed(ParentID, c, -1).
```

but `pig(a, c, -1)` is not a valid predicate, as one can see in [example 4.2.2](#).

Note that if X has no parents, then $\text{parents}(X) = \emptyset$, and the only compatible conjunction is the one with ID 0. Hence, the only transition functions that can be synthesised are the constant functions (always 1 or always 0).

4.2.2.4 Dynamic Constraint

What are the best functions regarding the given dynamics?

Global Dynamic Compatibility

Recall that we intend to synthesise Boolean networks whose respective dynamics is compatible with the given dynamics. More precisely, from the given dynamics, we extract a set of transitions $(i \rightarrow o)$, where i and o are Boolean configurations of size $|\mathcal{S}|$, and we want these transitions to appear in the general-asynchronous transition graph of the synthesised Boolean networks.

When the dynamic data are given as a (partial) truth table, the set of transitions corresponds to the set of (input, output) of the given truth table

Example 4.2.9 (Retrieve the transitions from a truth table). *Consider the following partial truth table:*

	input	output
	A B C	A B C
1	000	000
2	001	100

We want the general-asynchronous transition graph of each Boolean network we synthesise to contain the following transitions: $000 \rightarrow 000$ and $001 \rightarrow 100$.

When the dynamics data are given as timeseries, the transitions to explain is retrieved from a *deduplicated* sequence \mathcal{S} of configurations. In such a sequence, the n -th configuration differs from the $(n - 1)$ -th configuration. This sequence is constructed from the binarised timeseries by discarding the configurations that repeat over several successive timesteps. Ultimately, we want this sequence to be a walk in the general-asynchronous transition graph of the Boolean networks we synthesise.

Example 4.2.10 (Retrieve transitions from a timeseries). *We obtain the following successive binarised observations from the binarised timeseries shown in [example 2.4.7](#) on page 61*

010 \rightarrow 010 \rightarrow 010 \rightarrow 010 \rightarrow 011 \rightarrow 011 \rightarrow 011 \rightarrow 100 \rightarrow 100 \rightarrow 100 \rightarrow
 001 \rightarrow 001 \rightarrow 001 \rightarrow 001 \rightarrow 001 \rightarrow 001 \rightarrow 001 \rightarrow 001 \rightarrow 001 \rightarrow 001.

Based on this sequence, we construct the following deduplicated sequence of configurations:

010 \rightarrow 011 \rightarrow 100 \rightarrow 001.

Note that ASK&D-BN can work with a *partial* truth table. However, it assumes the given timeseries are *complete* (i.e., they do not have missing values). Yet, it allows inconsistent transitions (where the same input leads to different outputs). However, this is not necessarily a problem since we work with the general-asynchronous update scheme, which also introduces non-determinism.

Since we work species-wise, we now introduce the notion of *local compatibility* of a transition function with respect to the given dynamics.

Local Dynamic Compatibility

Given a species X , the species that are important for the synthesis of its transition functions are the species in $\mathcal{I} = \{X\} \cup \text{parents}(X)$. We consider the restriction of the transitions on the set \mathcal{I} . More precisely, given a transition $(i \rightarrow o)$ where $i(X) \neq o(X)$, we restrict the input on $\text{parents}(X)$ and the output on X . The element i is thus a Boolean vector of size $|\text{parents}(X)|$ and o is only one Boolean value. A restricted transition is denoted $(i_{|\text{parents}(X)} \rightarrow o_{|X})$. The set of restricted transitions obtained from a deduplicated sequence of configuration forms a partial truth table which a candidate transition function should be able to explain. This explains why we only consider transitions in which X actually changed: we want to make the fewest assumptions possible and give the candidate the benefit of the doubt.

Example 4.2.11. *We continue [example 4.2.10](#). For the species B whose parents are C and itself ([example 4.2.2](#)), the restricted transitions are: $[B : 1, C : 0] \rightarrow [B : 1]$, $[B : 1, C : 1] \rightarrow [B : 0]$ and $[B : 0, C : 0] \rightarrow [B : 0]$. Only the second one is to explain since in the two others, the value of B has not changed.*

We aim to synthesise transition functions whose truth table corresponds to the set of restricted transitions. Let f_X be a candidate transition function and $(i \rightarrow o)$ a restricted configuration. We say that $(i \rightarrow o)$ is *unexplained* with respect to f_X when the value of X predicted by $f_X(i)$ disagrees with the value o . We have two cases:

Unexplained activation: $f_X(i) = 0$ while $o = 1$ (hence none of the conjunctions of f_X is satisfied by the input i).

Unexplained inhibition: $f_X(i) = 1$ while $o = 0$ (hence at least one of the conjunctions of f_X is satisfied by the input i).

From the set of unexplained transitions, we compute the error $\epsilon \in \mathbb{R}_+$. The way to compute ϵ depends on whether the data are given as a truth table or timeseries. This will be detailed later. For now, we assume the error ϵ of a given candidate transition function is stored in the predicate `error(EPSILON)`. We want to keep all the transition functions that minimise ϵ . In ASP, this is expressed as

```
1 #minimize{EPSILON@2 : error(EPSILON)}. % highest priority
```

Ultimately, this minimisation results in synthesising functions that fit the transitions. Intuitively, there are three cases:

1. $\epsilon = 0$: the candidate f_X agrees perfectly with the given dynamics.
2. $\epsilon > 0$ and there is a transition $(i \rightarrow o)$ with an unexplained activation: f_X misses a conjunction c such that c is satisfied by the assignment i . We would favour a candidate f'_X that contains c , because it would explain the activation.
3. $\epsilon > 0$ and there is a transition $(i \rightarrow o)$ with an unexplained inhibition: f_X contains at least one conjunction c that is satisfied by the assignment i . We would favour a candidate f'_X that does not contain the conjunction c , because it would explain the inhibition.

By fitting each transition functions with this strategy, we assume the synthesised Boolean networks will contain the observed transitions in their transition graph computed with the general-asynchronous update scheme.

We now explain how to compute the error ϵ from a truth table and from a timeseries.

Computation of the Error ϵ Relatively to a (partial) Truth Table

When the dynamics is given as a truth table \mathcal{T} , the error ϵ corresponds to the number of lines in the truth table where the candidate function f_X disagrees with the output:

$$\epsilon = |\mathcal{U}| \quad \text{where } \mathcal{U} = \{(i, o) \mid f_X(i) \neq o, \forall (i, o) \in \mathcal{T}\}$$

To compute ϵ in ASP, we proceed as follows. First, we spot for which input i of the truth table we have $f_X(i) = 1$. To do so, we generate a predicate `compatible(ℓ, c)` if the input at the ℓ -th line of the truth table satisfies the c -th conjunction of f_X :

```

1 compatible(L, C) :-
2     input(L, NodeID, Val) : conjTaken(C, NodeID, Val), Val != 0
3     ; inputID(I); conjIDTaken(C) .
4 nbCompatibleConj(I, Count) :-
5     Count = #count {ConjID : compatible(I, ConjID), inputID(I)}; inputID(I).
    
```

Then, we generate a predicate `pbDynamic(ℓ)` for each line ℓ of the truth table where the observed output and the prediction of the candidate transition function differ. That is, if the line ℓ falls in one of the two cases below:

```

1 % Unexplained activation:
2 pbDynamic(L) :-
3     ttout(L, x, 1)
4     ; nbCompatibleConj(L, Count)
5     ; Count == 0 .
6 % Unexplained inhibition:
7 pbDynamic(L) :-
8     ttout(L, x, -1)
9     ; nbCompatibleConj(L, Count)
10    ; Count > 0 .
    
```

Finally, the predicate `pred/1` stores the number of lines for which the predicate `pbDynamic/1` was generated:

```

1 error(EPSILON) :- EPSILON = #count{L : pbDynamic(L)}.
    
```

Computation of the Error ϵ Relatively to a Timeseries

When the dynamics is given as a timeseries, the procedure to compute ϵ is slightly more complicated. Indeed, we will weigh the unexplained transitions with “how far” the observations of \mathbf{X} are from the binarisation threshold θ_X . The procedure is the following. Given a binarised timeseries $\hat{\mathcal{T}}$, let d be the function that returns the set of timesteps t^\star that are the *consecutive duplicates* of a given timestep t .

$$d(t) = \{t^\star \mid \hat{\mathcal{T}}_t = \hat{\mathcal{T}}_{t^\star} \text{ and } \hat{\mathcal{T}}_t = \hat{\mathcal{T}}_{t'} \text{ for all } t' \in [t, t^\star] : \}$$

Remark that $t \in d(t)$ by construction.

Example 4.2.12. Consider the following sequence of configurations, indexed from $t = 0$ to $t = 6$: $000 \rightarrow 100 \rightarrow 100 \rightarrow 110 \rightarrow 110 \rightarrow 111 \rightarrow 000$. We have that $d(0) = \{0\}$, $d(1) = \{1, 2\}$, $d(2) = \{2\}$, $d(3) = \{3, 4\}$, $d(4) = \{4\}$, $d(5) = \{5\}$ and $d(6) = \{6\}$.

In ASP, we generate a predicate `consecutiveduplicate_tuple(t^\star, t)` for each pair (t^\star, t) where $t^\star \in d(t)$, but t is not itself a consecutive duplicate (hence $t^\star \notin s(t^\star - 1)$).

```

1 consecutiveduplicate(T) :-
2     obsBinarized(T-1, N, V) : obsBinarized(T, N, V); time(T).
3 consecutiveduplicate_tuple(T, Tfirst) :-
    
```

```

4      consecutiveduplicate(T)
5      ; time(T); time(Tfirst); Tfirst < T; not consecutiveduplicate(Tfirst)
6      ; consecutiveduplicate(Ta):time(Ta), Ta > Tfirst, Ta <= T.
7      consecutiveduplicate_tuple(T, T) :- time(T).

```

Let t be a timestep that is *not* a consecutive duplicate. If there is an unexplained activation or inhibition of X between $\hat{\mathcal{T}}_{t-1}$ and $\hat{\mathcal{T}}_t$, we generate a predicate `pbDynamic(t)` that stores the timestep on which an unexplained configuration starts. That is, if one of the two following cases applies:

```

1  % Case 1: unexplained activation
2  pbDynamic_unexplainedactivation(T) :-
3      obsBinarized(T, x, 1)
4      ; nbCompatibleCunjunction(T-1, Count)
5      ; Count == 0
6      ; not consecutiveduplicate(T).
7  pbDynamic(T) :- pbDynamic_unexplainedactivation(T).
8
9  % Case 2: unexplained inhibition
10 pbDynamic_unexplainedinhibition(T) :-
11     obsBinarized(T, c, -1)
12     ; nbCompatibleCunjunction(T-1, Count); Count > 0
13     ; not consecutiveduplicate(T).
14 pbDynamic(T) :- pbDynamic_unexplainedinhibition(T).
15
16 % Helpers predicates:
17 compatible(T, ConjID) :-
18     obsBinarized(T, NodeID, Val):conjTaken(ConjID, NodeID, Val), Val!=0, ConjID!=0
19     ; time(T)
20     ; ConjID(ConjID)
21     ; ConjID!= 0.
22 nbCompatibleConj(T, Count) :-
23     Count = #count {ConjID : compatible(T, ConjID), time(T)}
24     ; time(T).

```

We now describe how to compute the error ϵ of a candidate f_X towards the timeseries. Let \mathcal{U} be the set of all timesteps t such that there exists a predicate `pbDynamic(t)`. The error ϵ_t of f_X at time t is the average of “how far” the value of X is from its binarisation threshold θ_X on all the timesteps on which the binarised configuration starting at t spans. This corresponds to the mean absolute error.

$$\epsilon_t = \frac{\sum_{t^* \in s(t)} |X_{t^*} - \theta_X|}{|s(t)|}$$

Finally, the total error ϵ is

$$\epsilon = \sum_{t | (t, t^*) \in \mathcal{U}} \epsilon_t$$

The procedure translates as follows in ASP:


```

1  % in case of pdDynamic at time T, determine the error associated with all the
   ↪ timesteps on which the config at T spans
2  error(T,S/C) :-
3      S=#sum{|V|, Tother: consecutiveduplicate_tuple(Tother,T), obs(Tother, c, V)}
4      ; C=#count{Tother: consecutiveduplicate_tuple(Tother, T), obs(Tother, c, V)}
5      ; pbDynamic(T).
6  sumErrors(S) :- S = #sum{E, T : error(T, E)} .
7  mae(S/NbT) :- sumErrors(S); nbtime(NbT).

```

Here again, the error is 0 when the candidate transition function explains all transitions.

4.2.2.5 Minimality Constraint

What are the best functions regarding their size?

So far, we generated the search space of transition functions, and evaluated the candidates based on their compatibility with the given structure and dynamics. However, we have no guarantee that a candidate transition function is encoded by a *minimal* DNF expression ([definition 1.6.7](#) on page 32). Indeed, there were no such constraints when we generated the search space ([section 4.2.2.2](#) on page 109). This section aims to show how ASK&D-BN enforces minimality.

As a first step, we can discard each candidate whose DNF has locally adjacent conjunctions (see [definition 1.2.7](#) on page 16), or conjunctions subsuming one another (see [definition 1.2.9](#) on page 16). Indeed, as seen in [section 1.6.3](#) on page 32, a DNF with such conjunctions cannot be minimal. However, these hard constraints are still insufficient to guarantee the synthesis of *minimal* DNF. Additionally, we rely on the active minimisation of the size of the DNF. The minimisation constraints depend on whether we are interested in *subset-minimal* or *cardinal minimal DNFs*. This distinction makes sense in the context of Boolean network synthesis from biological data. Indeed, the given dynamics is usually partial in the sense that it does not explore all the possible behaviour of the system under study. Consequently, the truth tables we build from such partial are partial as well, for there are some inputs with unknown outputs. Recall from [section 1.6.3](#) that each possible completion of a partial truth table corresponds to one Boolean function which corresponds in turn to a (set of) subset-minimal DNF(s). Also, recall that among all the subset-minimal DNF compatible with a partial truth table, the smallest DNFs are called the cardinal-minimal DNFs [definition 1.6.10](#). Note that all cardinal-minimal DNF are subset-minimal, but the converse is not true.

Remove Candidates with Local Adjacent and Subsumptive conjunctions

The following ASP constraints enforce that no conjunctions picked by the choice rule (during the search space generation) are locally adjacent nor subsume one another.

```

1  :- pbpolarity(C1, C2); isvariablessubsetof(C1, C2); conjID(C1); conjID(C2); C1!=C2.
2
3  pbpolarity(C1, C2) :- diffcount(C1, C2, 1).
4
5  diffcount(C1, C2, N) :-
6      conjID(C1); conjID(C2); C1 != C2
7      ; N=#count{NodeID : isdiff(C1, C2, NodeID)}.

```

```

8
9  isdiff(C1,C2, NodeID) :-
10     conjID(C1); conjID(C2); C1!=C2; varID(NodeID)
11     ; cube(C2, NodeID, V2); cube(C1, NodeID, V1); V2!=V1; V2!=0; V1!=0.
12
13  isvariablessubsetof(C1, C2) :-
14     conjID(C1); conjID(C2)
15     ; NI=#count{NodeID : commonimportantvariables(C1, C2, NodeID)}
16     ; nbLiteral(C1, NI).
17
18  commonimportantvariables(C1, C2, NodeID) :-
19     conjID(C1); conjID(C2); C1 != C2; varID(NodeID)
20     ; cube(C1, NodeID, Val1); cube(C2, NodeID, Val2)
21     ; Val1 != 0; Val2 != 0.

```

Enforce the Synthesis of Transition Functions in Cardinal-Minimal DNF

Finding the cardinal-minimal DNFs is the most straightforward since we only need to minimise the number of literals used by the DNF of the candidate transition function. In ASP, we use the predicates `nbLiteral(C, N)` to store the number `N` of literals in each conjunction `C`.

```

1  % count the number of literals whose polarity is not 0 in the conjunction C
2  nbLiteral(C, S) :- conjIDTaken(C); S=#sum{|V|, N : cube(C, N, V)}.

```

The size of a candidate DNF is computed by summing the number of literals in each conjunction.

```

1  nbLiteral(S) :- S=#sum{N, C : nbLiteral(C, N), conjIDTaken(C)}.
2  % N literals in the conjunction C

```

Finally, we minimise the expression size using the following ASP rule.

```

1  #minimize{S@1 : nbLiteral(S)}.

```

The priority of this minimisation rule is 1. That is lower than the one priority given earlier to the minimisation of the error toward the dynamics (section 4.2.2.4). The size of the candidate will thus be optimised only afterward. Hence, among the candidates transition functions with the smallest error, ASK&D-BN selects the ones represented with the smallest DNF expressions.

Example 4.2.13. Let $f_B^1 := B \wedge \neg C$ and $f_B^2 := (B \wedge \neg C) \vee (\neg B \wedge C)$ be two candidate transition functions. Let us assume both have the same error with respect to the input dynamics. Here, ASK&D-BN will prefer f_B^1 over f_B^2 as it is smaller. Indeed, it has a size two instead of four.

Relax the Cardinal-Minimal Constraint to get Subset-Minimal DNFs

We want to relax the cardinality constraint to synthesise all the subset-minimal DNFs, not only those that are cardinal-minimal DNFs. We cannot simply remove the ASP minimisation rule

mentioned above, as ASK&D-BN could then generate non subset-minimal DNFs that the hard constraints we presented earlier (about local adjacency and subsumption) do not rule out.

We thus proceed with two sequential ASP programs. The first one deals with the partial truth table corresponding to the observations, from which it produces all the possible complete truth tables by guessing the unobserved lines. The second ASP program runs on each completed truth table and generates the corresponding cardinal-minimal DNFs.

Example 4.2.14. *To illustrate the procedure, we use the truth table from [example 1.6.9](#) on page 33 except that we now assume the output for the 7th line (input 111) is unknown. Hence, we guess what the output could be: either 0, or 1. We end up finding different answer sets depending on the initial guess.*

- *If we guess that 111 returns 0, then the solution is $A \vee B \vee C$ which has size 3.*
- *If we guess that 111 returns 1, then the solutions are $(\neg B \wedge C) \vee (A \wedge \neg C) \vee (\neg A \wedge C)$ and $(B \wedge \neg C) \vee (A \wedge \neg B) \vee (\neg A \wedge C)$ which have size 6.*

All these DNF are subset-minimal by construction and explain the same partial observation. Yet, applying the parsimony principle ([section 2.4.1.3](#)), one can consider the shortest one (obtained when the guess for 111 is 0) to be more parsimonious and thus better.

The procedure is also illustrated in [example 1.6.11](#) on page 34.

4.2.3 Step 3: Global Boolean Network Assembly

The goal of this step is to create all the possible Boolean networks from the sets \mathcal{F}_X of transition functions we synthesised for each species in $X \in \mathcal{S}$ in the previous step. To do so, ASK&D-BN generates the Cartesian product of the sets. The number of Boolean networks thus corresponds to the product of the number of functions synthesised for each species. However, in practice, if the mincard option is used, the local synthesis step usually finds one formula for each species, resulting in a unique assembly ([section 4.4](#)).

Example 4.2.15. *Let us consider the Boolean network synthesis problem for three species A, B and C yielded the following sets of transition functions: $\mathcal{F}_A = \{f_A^1, f_A^3, f_A^3\}$, $\mathcal{F}_B = \{f_B^1\}$, $\mathcal{F}_C = \{f_C^1, f_C^2\}$. The global Boolean network assembly will produce six Boolean networks, corresponding to all the possible combinations.*

4.2.4 Evaluation of the Synthesised Boolean Networks

In the last step of ASK&D-BN, we evaluate the quality of the synthesised Boolean networks. Several criteria are considered:

- **The number of Boolean network synthesised**, which should be small.
- **The compatibility of the synthesised Boolean networks with the given structure and dynamics.** Since the Boolean networks are compatible with the given structure (by construction), our quality check focuses on the compatibility with the given dynamics. The *coverage ratio* of a Boolean network is the proportion of transitions extracted from the data

that are in its general-asynchronous transition graph. We compute this coverage ratio for each Boolean network synthesised. Then, we aggregate the individual coverage ratios by computing their median and standard deviation. Ideally, the pipeline returns *only* Boolean networks with *maximal* coverage ratios i.e., with a median of 1 and a standard deviation of 0.

- **The monotony of the local transition functions.** Following the basic principle of parsimony (section 2.4.1.3), a biological species is usually assumed to have either an activation or an inhibition role towards another species [Son07]. As a result, non-monotone local transition functions (i.e., functions which contain a literal and its negation) are supposed to be quite unlikely. We thus count the number of monotone local transition functions synthesised. Note that only a non-monotone prior influence graph can result in the synthesis of non-monotone transition functions. Note that, by construction, a prior influence graph built from a reaction network using rules and events is non-monotone (section 3.2 on page 76).

4.3 Related Works

Boolean network synthesis is a long ongoing problem. It is an extension of Boolean function synthesis, which is not specific to biology and found some applications in electronic circuit synthesis, for example. As mentioned in chapter 2, various methods have been proposed to synthesise Boolean network models of biological systems from a given structure and dynamics. To date, Boolean networks are often built by hand, but there are some methods that aim at automating the process [LFS98; LSY03; Ost+16; AD22; Che+19b; BK18; Dor+16]. They all share common characteristics, in particular regarding the general notion of compatibility. Indeed, they all roughly consist in enforcing that the influence graph and the transition graph of the synthesised Boolean networks contain specific edges that correspond to what is known of the structure and dynamics of the underlying system. However, despite these commonalities, there are *several important keypoints* for which they use different strategies. I believe it is crucial to address them carefully because they have a significant impact on the quality of the synthesised Boolean networks.

In the following, we thus discuss some of these keypoints. We particularly focus on REVEAL [LFS98], Best-Fit [LSY03] and Caspo-TS [Ost+16], as they are the methods with which we compare ASK&D-BN in section 4.4. We also use this discussion section to take a step back and justify several modelisation choices made in ASK&D-BN.

Outline The section is organised as follows. Section 4.3.1 discusses the definition of non-redundant solution. Sections 4.3.2 and 4.3.3 review the way of fitting to the structure and to the dynamics, respectively. Section 4.3.4 compares the two main strategies of enumeration of the solutions: the partial enumeration of the good-enough solutions versus the exhaustive enumeration of the globally good solutions. Finally, section 4.3.5 deals with the notion of simplicity of the solutions, in particular in terms of monotony and size.

4.3.1 Non-redundant Boolean Networks

To the best of my knowledge, all the automatic Boolean network synthesis methods claim to synthesise non-redundant Boolean networks. However, they do not all use the exact same definition of non-redundancy. Since all the Boolean network synthesis methods I am aware of work species-wise, we will first define non-redundancy on transition functions. The definitions will then be extended to define non-redundant Boolean networks.

The notion of non-redundancy depends in fact on how the transition functions are represented (which data structure, in a sense). Recall from [section 1.6](#) on page 29 that a transition function is a Boolean function $\mathbb{B}^n \rightarrow \mathbb{B}$, where n is the number of inputs of the function. The Boolean mapping it corresponds to can be represented with several kinds of expressions, including —but not only— minimal propositional logic expressions in disjunctive normal form (minDNF). Some data structures used to represent Boolean functions are canonical, which means a data structure for which there is a unique representation of a given mapping. Truth tables are one of such canonical representations, for example. However, minDNF are not since several minDNF can represent the exact same mapping ([example 1.6.9](#) on page 33).

The choice of representation is often driven by the necessity of easing the synthesis, or reducing its computation cost. Once the representation has been fixed, it is possible to define what a redundant solution is. There are two co-existing ways of defining non-redundancy of transition functions (and thus of Boolean networks): the one based on the mapping of the transition functions, and the one based on the expressions representing the transition functions.

4.3.1.1 Redundancy of the Boolean Mapping

The first definition of redundancy deals with the Boolean mappings. It is particularly adapted for the methods in which the transition functions are represented as truth tables, or full DNF expressions. In REVEAL and Best-Fit, the transition function are encoded in minDNF, but the non-redundancy is defined from the mapping. When several minDNF are equivalent, only one is picked at random.

At the Boolean network level Non-redundant Boolean networks have to differ in at least one configuration in the mapping $\mathbb{B}^n \rightarrow \mathbb{B}^n$. This definition of non-redundancy is usually impractical with a large number of inputs because it requires comparing all the mappings.

4.3.1.2 Redundancy of the Expressions

The second definition of redundancy deals with the expressions themselves. This definition is usually applied on expressions in standard form, such as DNF, for example. Two expressions E and E' are said to be redundant if E' can be derived from E using a given set of syntactic transformation rules. These rules have to preserve the mapping represented by the expressions. The rules typically involve changing the order of the literals in a conjunction, or of the conjunctions in a disjunction. Caspo-TS is one of the Boolean network synthesis methods that use this notion of non-redundancy. It is thus able to return all the minDNF which equivalently encode a given mapping.

At the Boolean network level Non-equivalent Boolean networks can have the exact same transition graph, even though the expression of the transition functions they comprise are

non-redundant.

In ASK&D-BN

We synthesise transition functions expressed in minDNF. This choice makes the search space larger compared to if we had considered the Boolean mapping, since a given truth table corresponds to potentially several minDNF.

Example 4.3.1. *The expressions shown in [example 1.6.9](#) on page 33 are two minimal DNF expressions corresponding to the same Boolean mapping. Hence, they are redundant according to the first definition ([section 4.3.1.1](#)), but not according to the second definition ([section 4.3.1.2](#)). In particular, ASK&D-BN would thus consider both as different solutions.*

However, it is also easier to check the non-redundancy. In our case, the notion of redundancy is defined syntactically. Redundant expressions are ruled out from the generation of the search space, thanks to the way we represent the candidates in ASP. Last but not least, these expressions are easy to interpret from a biological point of view. Indeed, minDNF can be seen as the list of minimal necessary conditions for a species to be activated.

4.3.2 Fitting the Structure

All the automatic Boolean network synthesis methods that use a prior influence graph use it to *hard* constraint the synthesis. This means that the influence graph of a synthesised Boolean network is necessary a spanning subgraph of the given prior influence graph.

However, the methods differ in the kind of information they use. For example, **Caspo-TS** can take the influence signs from a prior influence graph, while **REVEAL** and **Best-Fit** cannot use them. Note that, a priori, this might not be too much of a problem. Indeed, it is reasonable to assume that the dynamics data will reflect the sign of the influence anyway.

In ASK&D-BN

We hard constrain the influence graph of the synthesised Boolean network to be a spanning subgraph of the given influence graph. This includes the signs of the influences. Since we represent the transition functions as minDNF expressions, we only have to use syntactic checks (see [section 1.6.3](#)).

4.3.3 Fitting the Dynamics

All the automatic Boolean networks synthesis methods claim to synthesise the best Boolean networks in regard of given dynamic data. They extract from the data a list transitions between Boolean configurations that one would like to see in the transition graph of the synthesised Boolean networks. The methods differ in how are the transitions retrieved, how is the fit between the dynamics and the transition graph of the Boolean networks measured, and what is the update scheme used to compute the transition graph.

4.3.3.1 Binarisation

At some point or another, the Boolean network synthesis procedures require the data to be binarised. As a matter of fact, there exist a lot of binarisation procedures. The choice of the binarisation procedure is crucial for the outcome and the choice is a difficult problem with no obvious best one. In particular, the procedure might heavily depend on the technology used to get the data (DNA chip in [Mar+07], RNA seq, trajectory from single cell data [Che+19a]).

Most of the methods either leave the binarisation burden to the user [AD22; LFS98; LSY03], or at least propose the user to provide the threshold themselves [Ost+16]. All these tools are thus agnostic of the technology used to produce the data.

The methods which propose to binarise the data themselves, tend to rely on quite simple binarisation procedures, such as the average threshold in [MDW04], or the midrange in [Vid+15; Ost+16]. Despite their simplicity, these procedures produce good results when applied in the context of Boolean network synthesis from biological data.

In ASK&D-BN

The user can provide the threshold for each species. However, by default, we use the midrange binarisation procedure. This is because it is intuitively less impacted than others (especially the average and median-based) by periods of time where the value of a species oscillates in a small range of values. The midrange procedure is however very sensitive to outliers. Yet, in the context of this thesis, we intend to use data obtained from the simulation of existing reaction networks interpreted with their differential semantics. It is thus reasonable to assume there is no such outliers.

4.3.3.2 Extraction of the Transitions from a Timeseries

When the data are given as times series, the method has to extract the transitions. Given a binarised timeseries $\hat{\mathcal{T}}$, some methods such as **REVEAL** and **Best-Fit** assume there is a *synchronous* transition at *each* timestep.

Example 4.3.2. We consider the binarised times series from *example 2.4.7* on page 61. **REVEAL** and **Best-Fit** try to explain $010 \rightarrow 010 \rightarrow 010 \rightarrow 010 \rightarrow 011 \rightarrow 011 \rightarrow 011 \rightarrow 100 \rightarrow \dots$

The problem with doing so is that the procedure is likely to result in considering inconsistent transitions i.e., transitions where a configuration is associated with distinct successor configurations. Indeed, biological systems often involve processes happening at different speeds, and the configurations corresponding to slower processes might thus span several timesteps. More generally, inconsistent transitions can also occur if the sampling rate is too high.

Example 4.3.3. We continue the *example 4.3.2*. We remark that some time elapses before the configurations to change. As a result, the transitions $010 \rightarrow 010$ and $010 \rightarrow 011$ are inconsistent.

Such inconsistencies cause the failure of **REVEAL** since it is not designed to handle them and dumbly attempts to find the functions that explain *all* the transitions (and thus have a *coverage proportion* of 1). Unlike **REVEAL**, **Best-Fit** can manage inconsistencies. To do so, it relaxes the definition of compatibility of a Boolean network with the dynamics. Indeed, it is capable of

returning Boolean networks that do not perfectly explain the measurements, as long as they explain the maximal number of timesteps. In other words, it weights each transition by the number of timesteps it was observed. But with this strategy, we now have the problem that **Best-Fit** focuses on explaining self-loop transitions, corresponding in particular to the slower processes, that span several timesteps.

Example 4.3.4. We continue [example 4.3.2](#). In the sequence shown there, $010 \rightarrow 010$ is observed three times and $010 \rightarrow 011$ only once, **Best-Fit** will focus on explaining the former instead of the latter.

The strategy of **Caspo-TS** [Ost+16] is quite different and makes much fewer assumptions regarding the dynamics of the underlying system. In particular, **Caspo-TS** focuses on explaining the asynchronous *reachability* of the configurations, instead of the transitions themselves. This feature is an asset in the case of missing time points.

Caspo-TS processes as follows. (1) First, it works on a deduplicated sequence of configurations. (2) Second, it includes wildcard configurations, that can be expanded to any asynchronous sequence of configurations.

Example 4.3.5. The following sequence of configurations $010 \rightarrow 011 \rightarrow 100 \rightarrow 001$ (taken from [example 2.4.7](#) on page 61) is transformed to $010 \rightarrow * \rightarrow 011 \rightarrow * \rightarrow 100 \rightarrow * \rightarrow 001$.

(3) Third, **Caspo-TS** attempts to find Boolean networks with asynchronous transition graph in which the sequence is a path. The third step is in fact divided into two steps. (3a) **Caspo-TS** uses an overapproximation of the dynamics using the so-called most-permissive semantics [Pau+20]. Because of this overapproximation, the result can contain many false positive Boolean networks, i.e., Boolean networks optimising the cost function used under the hood of **Caspo-TS**, while their asynchronous dynamics is not able to reproduce the configuration sequence of the multivariate timeseries. (3b) These false positive Boolean networks are subsequently ruled out using exact model checking. This filtering is *PSPACE*-hard, but thanks to the step (3a), a large set of Boolean networks has already been excluded.

One last thing to mention with the strategy of **REVEAL** and **Best-Fit**, it that since they do not use the quantitative data, they cannot mitigate the binarisation effect at all. **Caspo-TS** has a finer grain error than the simple coverage used by **REVEAL** and **Best-Fit**. Indeed, it takes the quantitative data into account to weight the errors by how far the value is from its binarisation threshold. This strategy thus relaxes a bit the effect of the binarisation. We can thus assume that **Caspo-TS** is less biased by the chosen binarisation procedure than **REVEAL** and **Best-Fit**.

In ASK&D-BN

We attempt to make as few assumptions as possible, in particular regarding the update scheme of the underlying system. This is why we used the general-asynchronous update scheme, as it is the most general one. Also, for a given species X , we only consider transitions where X has changed its value, without caring about whether the parents of X have changed their value. Thanks to this strategy, we kind of let the benefit of the doubt to the candidate transition functions: they only have to explain the transitions for which we are sure X has updated. This strategy assumes some kind of switch-like behaviour (as soon as X changed, we look at the value of its parents in the previous timestep), but this assumption sounds reasonable since this is often the case in biology.

Example 4.3.6. We consider the binarised times series from [example 2.4.7](#) on page 61. *ASK&D-BN* consider the following sequence of configurations:

$$010 \rightarrow 011 \rightarrow 100 \rightarrow 001.$$

Regarding the binarisation, we already justified the choice of midrange by default. But on top of this, we have a similar strategy to *Caspo-TS* in that we use the quantitative values of the input multivariate time-series to compute the error. The error for an unexplained transition where X was close to its threshold is smaller than for an unexplained transition where X is far from it. This relaxes a bit the effect of the binarisation.

Finally, let us mention the strongest assumption of *ASK&D-BN*. It assumes the given timeseries are complete i.e., there are no missing time points. Since we intend to use *ASK&D-BN* with data obtained by simulation, this assumption is fulfilled.

4.3.4 Partial Good-Enough versus Exhaustive Optimal Synthesis

Recall that the mapping of a Boolean function can only be uniquely identified if all the input/output pairs of its truth table are provided [AMK99]. Moreover, even a complete truth table corresponds potentially to several Boolean expressions. In the context of Boolean network synthesis from biological knowledge and data, the available dynamics data is usually very incomplete. In turn, the Boolean network synthesis problem is thus not very constrained. As a result, the search space is really large, and the number of equally good transition functions—and thus Boolean networks—vis-à-vis the constraints is often huge [Mar+07]. There are two strategies regarding this.

Some Boolean network synthesis methods are designed to return *all* these equally good Boolean networks, exhaustively. It is usually the case for the methods that rely on constraint satisfaction technics, such as logic programming [Che+20; Vid+15] or SAT-Modulo Theory (SMT). In both cases, the set of constraints is encoded and an exhaustive search can (virtually) be performed. When too many Boolean networks are synthesised, some methods post-filter them based on other criteria that were not used to hard constrain the synthesis. In [Mar+07] for example, the authors only keep the Boolean networks whose synchronous transition graph has some known attractors. Finally, note that if a unique Boolean network is required, it is always possible to extract one by randomly selecting one transition function for each species. This is the solution adopted by most of the methods [Mar+07]

In contrast, other methods are limited to the synthesis of only one Boolean network, the best they could find [AMK00]. It is mainly the case for methods based on stochastic optimisation technics (such as genetic algorithm, for example). As we discussed in [chapter 2](#), not only such methods cannot synthesise all the equally good solutions, but also cannot guarantee to eventually synthesise globally optimal Boolean networks. *CellNOptR* [Ter+12] (standing for Cellular Network OptimizerR), which implements in *R* the method introduced in [Sae+09], is one of such methods. It is based on a genetic algorithm that returns the best Boolean network it could derive during the run.

In ASK&D-BN

We opted for the first strategy and return all the equally good Boolean networks. For this, we rely on the clever exhaustive search performed by ASP. In the case, too many Boolean networks are synthesised, our philosophy would be to avoid post-filtering the Boolean networks based on new knowledge and data not used for the synthesis. We would rather encode these as new constraints. ASP is really flexible, so this strategy would really not be a problem.

4.3.5 Simplicity

All the Boolean network synthesis methods I am aware of claim to synthesise Boolean networks with the *simplest* transition functions possible. This is justified by the parsimony principle, as *among all the possible explanations, the best ones are the simplest*. The concept of simplicity usually involves the minimality of the transition functions and monotony of the influences among the species (section 2.4.1.3).

4.3.5.1 Minimality

The definitions of minimal transition function differ slightly depending on whether it is its mapping or its expression that is considered.

When focusing on the mapping, the goal is simply to eliminate useless inputs, i.e., an input whose status has no consequence on the species of interest. Each input evicted halves the number of lines in the truth table, or equivalently, the number of conjunctions in the corresponding full DNF.

When focussing on the expression, the measure of simplicity relates to the size of the expressions (section 1.6.3). Since minimisation is a hard problem, some approximative methods have been developed. Among them, we can mention the method Espresso [Bra+82] that is used in [MDW04]. It returns *one* DNF and there is no guarantee for it to really be minimal.

4.3.5.2 Monotony

As already mentioned in chapter 2, several methods [Ost+16; Che22] hard constraint the monotony of the influences used by the transition functions. Not only this drastically reduces the search space (see table 1.5 on page 38), but it also complies with the fact that in biology, a species is *usually* either an activator or an inhibitor (but not both).

Caspo-TS is one of such methods that can only generate Boolean networks with bipolar transition functions (definition 1.6.22 on page 36).

In ASK&D-BN

We return Boolean networks with transition functions expressed in minDNF. The strategy to obtain minDNF expressions is to actively minimise the number of influences used by the expression, directly in ASP. The minimisation is thus exact. Moreover, we return all the minDNF that are equally good regarding the given constraint. In particular, we do not take the monotony of the transition functions into account. That is because some authors contested the use of monotony as

a hard constraint [NRS11], and I personally share their point of view. However, I think parsimony is a good criterion in evaluating the synthesised functions.

4.4 Evaluation of ASK&D-BN

In this section, we evaluate ASK&D-BN by comparing it with three state-of-the-art methods, namely REVEAL [LFS98], Best-Fit [LSY03] and Caspo-TS [Ost+16].

4.4.1 Boolean Network Synthesis Methods for the Comparison

In this section, we summarise the key elements of each method with which we compare ASK&D-BN. Note that we chose these methods because they are very similar to ASK&D-BN in their inputs and in their goal. Indeed, as seen in section 4.3, they all use a prior influence graph and dynamic data to synthesise an *exhaustive* set of Boolean networks. However, despite this similarity, and despite they inspired our work, they differ on several key points that were discussed in section 4.3. table 4.1 summarises their differences. In particular, unlike REVEAL and Best-Fit, ASK&D-BN is capable of using the influence signs provided in the given prior influence graph and the raw values of the input multivariate timeseries. Unlike Caspo-TS, ASK&D-BN directly fits the behaviour of each species with the timeseries. Also, it is not limited to the synthesis of Boolean networks with bipolar transition functions.

Table 4.1: Comparison of ASK&D-BN with REVEAL, Best-Fit and Caspo-TS. The colours indicate how bad we assume the assumptions are in the specific context of Boolean network synthesis from structure and dynamics retrieved from a reaction network (red: very bad, orange: not so bad).

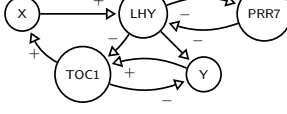
Method	PIG	Assumption on the TS and the config. seq.	Redundancy	Function Class
REVEAL	Unsigned	Sync. transition at each timestep and no inconsistency	Mapping	All
Best-Fit	Unsigned	Sync. transition at each timestep	Mapping	All
Caspo-TS	Signed	Asynchronous reachability	Expression	Bipolar
ASK&D-BN	Signed	Complete timeseries	Expression	All

4.4.2 Datasets

A dataset corresponds to a prior influence graph and a timeseries. Two of our datasets are real published datasets of two biological systems, while the others correspond to generated data for a given system (among six) in a given setting (among eight), with a given seed (among seven). We thus have a total of 338 datasets.

Real Datasets The real datasets are about the *yeast*'s cell cycle and *A. thaliana*'s circadian clock. The influence graph and experimental multivariate timeseries are taken from [Loc+06] and [Spe+98], and they involve four and five species respectively. Table 4.2 summarises these two datasets.

Table 4.2: Summary of the two real datasets used for the evaluation of ASK&D-BN.

System	Species	PIG	TS	Source
<i>Yeast</i> (cell cycle)	Fkh2, Swi5, Sic1, Clb1	Sic1 does not influence itself nor Fkh2	14 timesteps 6 transitions	[Spe+98]
<i>A. thaliana</i> (circadian clock)	LHY, PRR7, TOC1, X, Y		50 timesteps 11 transitions	[Loc+06]

Generated Datasets For the six other systems¹⁵, we generate the data from existing Boolean networks taken from the repository of Boolean networks of the package `PyBoolNet` [KSS16]. Except for one, these Boolean networks are published models of real biological systems. For these systems, the number of species ranges from three to ten.

The prior influence graph we consider is the influence graph of the associated Boolean network. As for the generation of the multivariate timeseries, three parameters are taken into consideration:

- the update scheme (either synchronous or asynchronous);
- the maximum number of repetitions introduced for each configuration (either 1 or 4);
- the standard deviation of the added noise (either 0 or 0.1).

We thus have eight combinations of these parameters, aka *settings*. In the following, we denote ARN the setting with the Asynchronous update scheme, random Repetitions of configurations (of 4 max) and Noise addition (with standard deviation of 0.1). This setting allows us to get close to real timeseries.

For each system and each setting, we produce seven timeseries. To do so, we use a procedure similar to the function `generateTimeSeries` of the R package `BoolNet` [MHK10]. The procedure is run with a different random seed each time.

1. Choose randomly a configuration c of the considered Boolean network
2. Apply the transition function(s) 20 times (starting from c) w.r.t the chosen update scheme.
3. If Repetitions: duplicate randomly each configuration in the obtained sequence (added in contrast to `generateTimeSeries`)
4. If Noise: add Gaussian noise with a standard deviation of N .

Now, we illustrate how to generate a multivariate timeseries in the ARN setting.

Example 4.4.1 (Generation of a multivariate timeseries in the ARN setting.). *We start from a random configuration of the Boolean network \mathcal{B}_1 (figure 1.8a on page 41). Let it be 010. Then we apply 20 times the transition functions of \mathcal{B}_1 with the asynchronous update scheme. This process*

¹⁵ Referred to as `raf`, `randomnet_n7k3`, `xiao_wnt5a`, `arellano_rootstem`, `davidich_yeast` and `faure_cellcycle` in `PyBoolNet`.

is not deterministic as any path from [figure 1.8e](#) starting from 010 and of length 20 is valid. Let us say we obtain a path starting with

$$010 \rightarrow 011 \rightarrow 010 \rightarrow 011 \rightarrow 111 \rightarrow 101 \rightarrow \dots$$

Now, we add a random number of duplications (in bold). The beginning of the sequence could for example look like

$$\begin{aligned} 010 \rightarrow 011 \rightarrow \mathbf{011} \rightarrow 010 \rightarrow 011 \rightarrow \mathbf{011} \rightarrow \mathbf{011} \\ \rightarrow 111 \rightarrow 101 \rightarrow \mathbf{101} \rightarrow \mathbf{101} \rightarrow \mathbf{101} \rightarrow \dots \end{aligned}$$

Finally, we add a random Gaussian noise with a standard deviation of 0.1. The generated multivariate timeseries could now start with

$$(0.02; 0.92; -0.16) \rightarrow (0.04; 0.8; 0.7) \rightarrow (-0.05; 1.06; 0.7) \rightarrow \dots$$

4.4.3 Evaluation Procedure

In the following, we use the word *experiment* to refer to a given Boolean network synthesis method applied to one of these datasets. As mentioned before, the unicity of the multivariate timeseries makes the problem under-constrained. This setting allows us to evaluate the performances of the different approaches in this specific context.

For REVEAL and Best-Fit, we use the implementation from the R package BoolNet [MHK10]. Caspo-TS is run with the option `mincard`, which asks for Boolean networks with functions minimising the number of influences. Note that this is also the option chosen for ASK&D-BN.

REVEAL, Best-Fit and ASK&D-BN need the binarised multivariate timeseries in their inputs. For the two systems with real timeseries, the binarisation threshold is determined species-wise, following the midrange binarisation procedure described in [section 4.2.1](#). For the six systems with the generated multivariate timeseries, we directly use a binarisation threshold of 0.5 because we know *a priori* that the values of all the components are between 0 and 1 (\pm the noise). This prevents from spurious transitions we might have had if computing threshold species-wise in noisy setting with constant species. Indeed, the fluctuations of a constant component would have been interpreted as a changes of configuration, that the Boolean network synthesis methods are not capable to detecting.

In order to have a fair comparison of the methods, and since Caspo-TS is making the binarisation itself and is not aware that the theoretical minimum and maximum of the components are 0 and 1 (\pm the noise), we correct the transition functions it returned *a posteriori*: the value of the constant is set to the binarised value that is the most present in the binarised timeseries of the component concerned. Also, since Caspo-TS does not return a function for the species without parents in the prior influence graph nor for the components that it found constant for all the timeseries (in the case where no noise is involved), we use the same technique to set the transition functions to their correct values. We also added a step to filter out the Boolean networks returned by REVEAL and Best-Fit that do not respect the polarities given in the prior influence graph of each dataset.

For all the Boolean networks returned by the four methods (and after the PIG-based filtering

for REVEAL and Best-Fit), we use PyBoolNet [KSS16] to compute the general-asynchronous transition graph of each Boolean network synthesised. Finally, we evaluate the results of each experiment according to two criteria:

1. The number of Boolean networks synthesised, which should be small.
2. The coverage ratio, i.e., the proportion of configuration transitions extracted from the input timeseries that are present in the general-asynchronous transition graph. The median of the coverage ratios should be as high as possible, but the standard deviation should be low.

All data and programs needed to reproduce the presented results are accessible at <https://gitlab.inria.fr/avaginay/OLA2021>.

4.4.4 Results on the Two Real Datasets

Yeast (figure 4.2 left). For this dataset, Caspo-TS synthesises 61 Boolean networks while both Best-Fit and ASK&D-BN synthesise 16. As for REVEAL, it does not return any Boolean network. This is because of inconsistent transitions in the sequence of configurations extracted from the timeseries, and that REVEAL is not tailored to handle. Concerning the coverage, on the seven transitions observed in the timeseries, the Boolean networks synthesised by Best-Fit recover four while those synthesised by ASK&D-BN recover five. The best coverage ratio (six retrieved transitions over seven) is obtained for eight Boolean networks synthesised by Caspo-TS (among the total of 61). Nevertheless, as the box plot shows, the Boolean networks synthesised by Caspo-TS present a large variance in their coverage.

A. thaliana (figure 4.2 right). For this dataset, REVEAL returns no Boolean network. This is again because of inconsistent transitions. The only Boolean network returned by Best-Fit has all the species set to 1 and recovers four transitions over the ten observed. ASK&D-BN also returns a single Boolean network with a perfect coverage since the Boolean network recovers all the ten transitions. As for the five Boolean networks synthesised by Caspo-TS, we can make the same observation as before: they present a variability in their coverage. The best coverage obtained by Caspo-TS are from two different Boolean networks including the one synthesised by ASK&D-BN.

To sum up, the results on these two real datasets show that:

- REVEAL constantly fails to return any Boolean network. At the opposite, Caspo-TS returns more Boolean networks than the other methods;
- the coverage of the Boolean networks returned by both ASK&D-BN and Caspo-TS are better than for Best-Fit;
- Caspo-TS presents worse variability in the coverage ratio of its Boolean networks compared to ASK&D-BN.

4.4.5 Results on the Generated Datasets

4.4.5.1 Number of Synthesised Boolean networks

The total number of Boolean networks returned on the generated datasets and the number of times the identification methods failed to synthesise any Boolean network are reported in table 4.3.

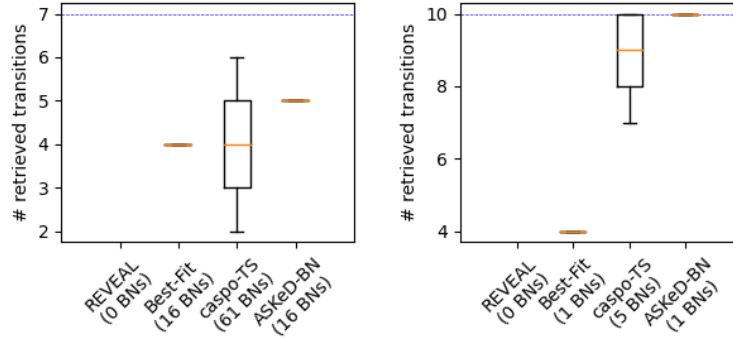


Figure 4.2: Coverage evaluation of the Boolean networks synthesised by **ASK&D-BN**, **REVEAL**, **Best-Fit** and **Caspo-TS** on two real datasets: *yeast* (left) and *A. thaliana* (right). The blue dashed line indicates the number of deduplicated transitions that were observed in the multivariate timeseries.

The table shows that surprisingly, a large proportion of Boolean networks generated by **REVEAL** and **Best-Fit** were not complying with the influence signs from the input influence graph. In the following reported results, we do not take these non-compliant Boolean networks into account. **REVEAL** returns the smallest number of Boolean networks, in particular in the ARN setting (Asynchronous update scheme, random Repetition of configurations, and Noise addition). This is due to the inconsistencies in the timeseries, which are frequent in the ARN setting (as in real timeseries), because of the repetitions we introduce. Conversely, **Caspo-TS** is the method that returned the largest number of Boolean networks. Moreover, when considering all experiments, there are 18 experiments for which **Caspo-TS** generated more than 100 Boolean networks. In these cases, we stopped the enumeration and analysed the 100 first Boolean networks **Caspo-TS** returned. Despite this limit, **Caspo-TS** returned between 5 and 7 times more Boolean networks than **ASK&D-BN**.

From here on, we focus on the results of the experiments corresponding to the ARN setting after having removed the Boolean networks from **REVEAL** and **Best-Fit** that do not respect the given FIG.

4.4.5.2 Coverage Ratio

To assess the coverage ratio criterion, instead of plotting the boxplots for the 42 experiments of the ARN setting (6 systems times 7 replicates), we summarised them in [figure 4.3](#). In the scatter plot, each experiment is represented by a point whose coordinates are the coverage ratio median of the synthesised Boolean networks and the associated standard deviation (std). The closer a point is to the top-right corner, the better the corresponding identification method is (i.e., it produces Boolean networks with a high coverage ratio and a low std). We can see that for the few experiments for which **REVEAL** was able to return Boolean networks, the median coverage is actually excellent. The median coverage of the Boolean networks returned by **Best-Fit** is almost uniform: **Best-Fit** lacks regularity in finding Boolean networks with good coverage. But the high peak around 0 on the plot of the std distribution shows that for a given experiment, the Boolean networks returned by **Best-Fit** have similar coverage rates. **Caspo-TS** and **ASK&D-BN** have a very

Table 4.3: Number of experiments for which each method failed to synthesise any Boolean network, number of Boolean networks synthesised over all 336 experiments with generated timeseries and number of Boolean networks returned over the 42 experiments with the ARN setting. The labels “Before” and “After” refer to the filtering step which rules out the Boolean networks not respecting the polarity of the influences of the given PIG (see [section 4.4.3](#)).

Setting	Measure	REVEAL		Best-Fit		Caspo-TS	ASK&D-BN
		Before	After	Before	After		
All	# failing experiments	230	240	0	64	20	0
All	# BNs returned	100 677 500	406	100 678 198	724	8481	1210
ARN	# BNs returned	3	3	51	35	720	85

similar distribution of median coverage. They are both good at finding Boolean networks with very good coverage. But here again, for a given experiment, the Boolean networks synthesised by **Caspo-TS** present a bigger variation of their coverage proportions than the ones synthesised by **ASK&D-BN**.

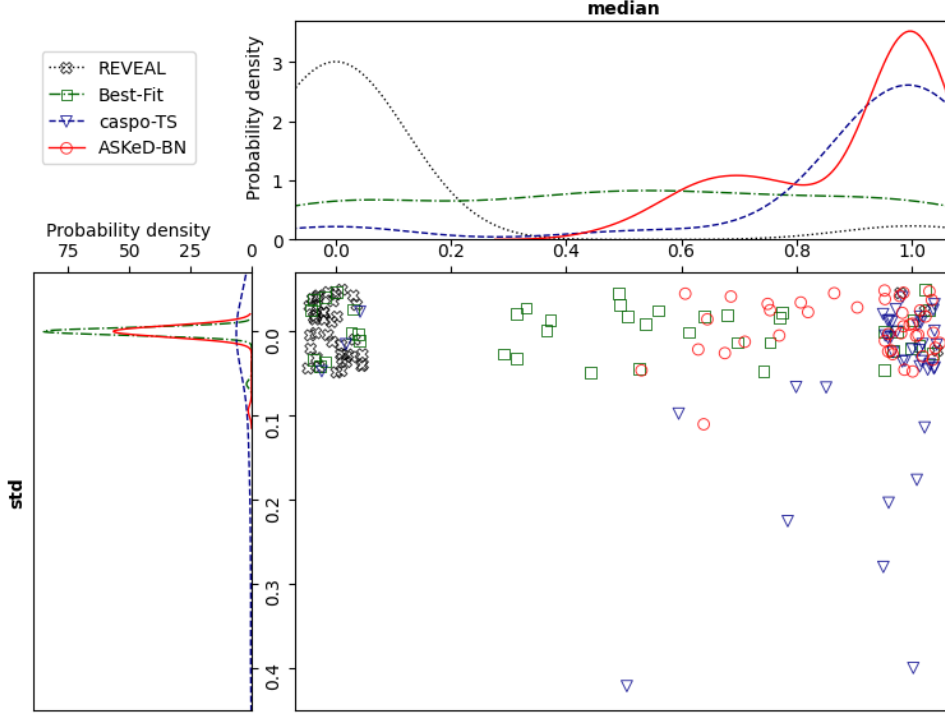


Figure 4.3: Coverage evaluation of the Boolean networks synthesised by **ASK&D-BN**, **REVEAL**, **Best-Fit** and **Caspo-TS** on the generated datasets in the ARN setting. On the scatter plot, each point represents the results of an experiment, for which a given method potentially returned several Boolean networks with different coverage ratios. The horizontal coordinate of the point is the median of these ratios. The vertical coordinate is their standard deviation (std). For better visualisation, the coordinates are jittered with a variance of 0.1 on both axes. The curves on the top (resp. on the left) of the scatter plot are the probability densities of the median (resp. the std) of the points in the scatter plots. The densities have been estimated from the non-jittered coordinates of the points with the Gaussian kernel density estimation method. The smoothing parameter of the estimator was determined automatically (with the Scott method). The areas under all these curves are 1, and the peaks show where the points are the most concentrated.

4.5 Wrap-Up

In this chapter, we presented **ASK&D-BN**, a novel method to synthesise Boolean networks *compatible* with a given structure (a prior influence graph) and a given dynamics (a truth table or a timeseries). The synthesised Boolean networks comply with the given structure and dynamics in that their influence graph is a spanning subgraph of the given prior and their general-asynchronous transition graph contains as many observed transitions as possible.

The reported results on 338 datasets (concerning eight systems) showed that **ASK&D-BN** has the

best trade-off on the evaluation criteria: it returns a small set of Boolean networks, all compatible with the given structure, and with a high coverage median and low variance. In particular, our results confirm that although **Caspo-TS** finds good Boolean networks, too many sub-optimal ones are retrieved. As we saw, this is because **Caspo-TS** consider the reachability property between the Boolean configurations, but a new version of **Caspo-TS** was recently proposed to tackle this problem, taking into account constraints on the transition themselves, as in **ASK&D-BN** [Che+20].

An interesting perspective would be to define formally what constraints would ensure finding the Boolean networks with the best coverage. Indeed, our results suggest that **ASK&D-BN** does not all the time return the best Boolean networks since **Caspo-TS** sometimes synthesises some better ones. There are already some studies about whether a Boolean network fitting given constraints exists, as well as how much needs to be known to identify the correct Boolean functions with a given probability [AMK99].

Moreover, **ASK&D-BN** uses constraints that are in fact tailored for the very specific application we developed it for, namely the synthesis of Boolean networks from structure and dynamics retrieved automatically from existing reaction networks ([chapter 3](#)). Because of this, **ASK&D-BN** makes assumptions that might not be verified when using real dynamics data. We now review these assumptions and hint how we could improve **ASK&D-BN** for it to handle better real dynamics data.

1. **ASK&D-BN** assumes the given times series are complete (hence without missing information). It is a valid assumption due to the way we retrieve the data ([chapter 3](#)), but this is generally not the case for real datasets, for which the sampling rate might be too small. In contrast, **Caspo-TS**, which works on the reachability of the configurations, is better suited to process real data.
2. Another limitation when using **ASK&D-BN** on real datasets is that they may contain outlier measurements which could mislead our binarisation procedure. It would thus be interesting to propose a better binarisation procedure with prior outliers detection, for instance.
3. Biologists often have several timeseries. They can correspond to perturbation experiments, where some species of the system under study are forced to stay either active or inactive. Exploiting such supplementary data gives more information about the behaviour of the studied system in specific conditions (e.g., pathological states). This knowledge constrains even more the space of solutions. **REVEAL**, **Best-Fit** and **Caspo-TS**, can handle multiple timeseries but **ASK&D-BN** cannot, even though it would be no problem to adapt the ASP encoding to handle this.

In the next chapter, we apply **ASK&D-BN** on structure and dynamics retrieved automatically from existing reaction networks encoded in the Systems Biology Markup Language (SBML).

Chapter 5

The SBML2BNET Pipeline and its Evaluation

Contents

5.1	Introduction	135
5.2	Input and Output File Formats	137
5.2.1	Reaction Networks in SBML	137
5.2.2	Boolean Networks in BNET	138
5.3	Implementation of the SBML2BNET Pipeline	138
5.4	Evaluation	141
5.4.1	Results on \mathcal{R}_{enz}	141
5.4.2	Results on SBML Models from BioModels	144
5.5	Wrap-up, Discussion, and Perspectives	149

5.1 Introduction

In this chapter, we describe and evaluate the SBML2BNET pipeline, which consists of an implementation of all the methods presented in the previous chapters. These methods concern how to retrieve a structure and a dynamics from a reaction network ([chapter 3](#)) and how to synthesise Boolean networks out of it ([chapter 4](#)). [Figure 5.1](#) summarises the different steps of the pipeline. The pipeline starts from a reaction network encoded in the Systems Biology Markup Language (SBML) and outputs a set of compatible Boolean networks in the BNET format.

Outline In [section 5.2](#), we briefly present the Systems Biology Markup Language and the BNET format, as well as some concepts related to the models we can process with the SBML2BNET pipeline. In [section 5.3](#), we present the implementation of the different steps of the pipeline. Then, in [section 5.4](#), we run different variants of the pipeline on the simple enzymatic reaction model as well as real-life models from the BioModels repository. Finally, in [section 5.5](#), we summarise our findings and close the chapter with some discussions and perspectives.

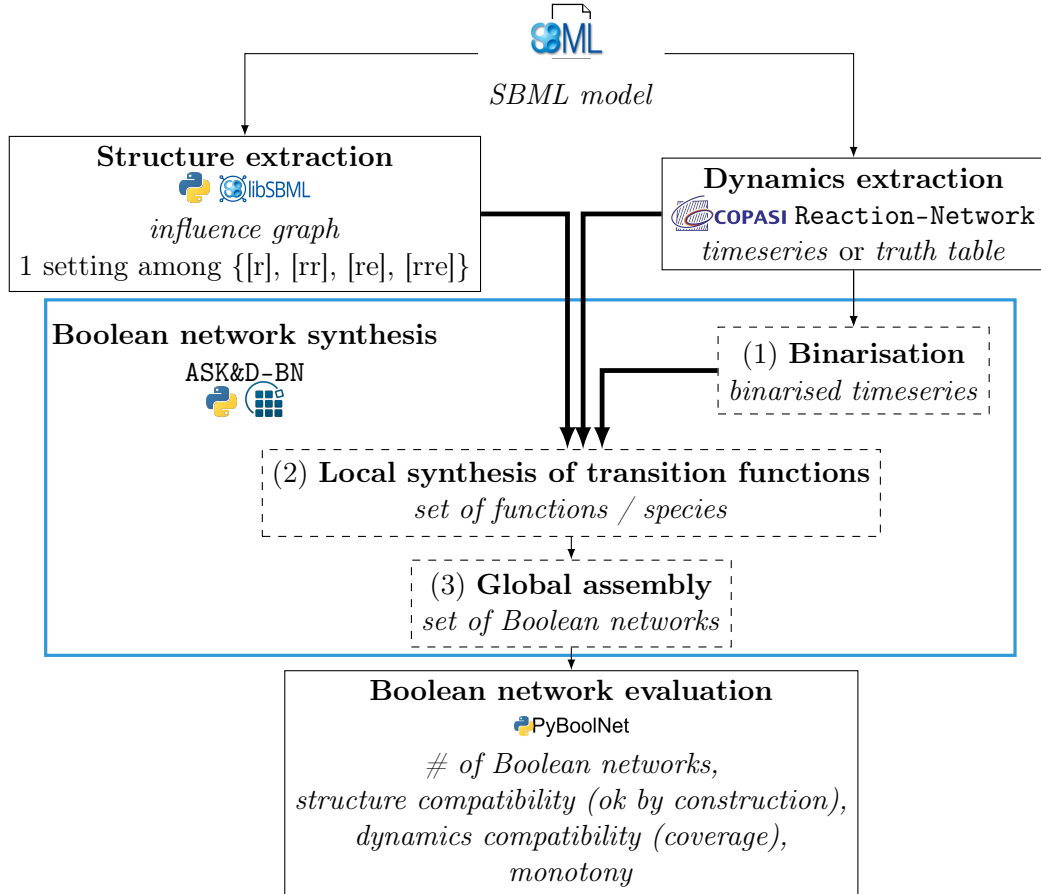


Figure 5.1: Workflow of the SBML2BNET pipeline. It starts with a reaction network encoded in SBML and produces Boolean networks in the BNET format. The Boolean network synthesis step (blue box) is composed of three steps among which the local synthesis of the transition functions which uses as input (thick arrows): a prior structure (influence graph) and a prior dynamics (a truth table, or a timeseries with its binarisation).

5.2 Input and Output File Formats

5.2.1 Reaction Networks in SBML

The Systems Biology Markup Language (SBML) is an XML markup language used to represent all kinds of biological models [Kea+20]. The SBML specifications¹⁶ define how the different elements are named and structured, as well as many features that make SBML very expressive. However, this thesis focuses on a subset of SBML models: those representing mechanism-based models (core reaction networks eventually extended with rules and events). Moreover, the encoded reaction networks must be simulable with the differential semantics. We do not give the BNF syntax as it is quite difficult to define and not very informative. However, we describe the content of the models we process. An SBML model involves elements referred to as **species**, **compartments**, **parameters**, **rules**, **reactions**, and **events**, that map the corresponding concepts in reaction networks.

An SBML **species** is located in a given **compartment**. A **compartment** may or may not represent actual physical structures. The **species** typically interact with other species from the same compartment, but that is not enforced by the SBML specifications. Moreover, the size of a **compartment** may change over time. This is important to track when dealing with species concentrations.

The initial value of every **species** (quantity, activity, or concentration) and **parameters** has to be provided. It can then either stay constant or change over time according to the **reactions** (only for the species), **rules** and **events**.

As mentioned in [definition 1.9.1](#), for a given reaction, a modifier can be either an activator or an inhibitor (or both) of the reaction. However, SBML does not distinguish activators from inhibitors. Still, in some SBML models, they are not specific annotations we can rely on to get the exact role of the modifiers. These annotations use the Systems Biology Ontology (SBO) [Cou+11]. We use the SBO terms `SBO:0000459`, `SBO:0000020` and `SBO:0000595`, that respectively stand for “activator only”, “inhibitor only” and “dual activity” modifier.

Example 5.2.1 (A reaction network in SBML). [Appendix A](#) on page 177 represent the reaction network of \mathcal{R}_{enz} ([example 1.9.4](#) on page 44) in SBML. It is, in fact, one possible representation of the model, since the SBML specifications are quite flexible.

We consider an SBML model to be *well-formed* when it respects the SBML specifications, and when each of its reactions respects the well-formedness criteria presented in [definition 1.9.6](#). Well-formness requirements that are often violated are for a reaction to be implicitly reversible, for species labelled as modifiers to have their stoichiometry changed, and for species appearing in a kinetic expression, but not listed as reactant or modifiers [FGS12].

Finally, the SBML2BNET pipeline can only process a given SBML model if it encodes a reaction network that is *complete* ([definition 1.9.5](#)), and whose associated ODE system is not under-determined or has multiple solutions. These are the requirements for us to retrieve a dynamics as discussed in [section 3.3](#) (either with concrete or with abstract simulation).

¹⁶ <http://sbml.org/Documents/Specifications>

5.2.2 Boolean Networks in BNET

We export the Boolean networks synthesised by the SBML2BNET pipeline in the BNET format. It is a simple text-based format, heavily used to represent Boolean networks [MHK10; KSS16]. We only use a subset of the BNET syntax¹⁷, as we only need to represent one DNF expression of each species. The BNF syntax we use is:

```

BoolNet ::= Header Newline
              {TransitionFunction Newline | Comment Newline}+
Header ::= targets , factors
TransitionFunction ::= SpeciesName , BoolExpr
Comment ::= # String
BoolExpr ::= 0 | 1
              | SpeciesName
              | (BoolExpr)
              | ! BoolExpr                                (negation)
              | BoolExpr & BoolExpr                       (conjunction)
              | BoolExpr | BoolExpr                        (disjunction)
SpeciesName ::= String
String ::= any sequence of characters (except a line break)
Newline ::= a line break character

```

Example 5.2.2 (A Boolean network in the BNET format). *The following BNET file encodes for the Boolean network shown in figure 1.8f on page 41:*

```

targets, factors
A, 0
B, (B&C) | (!B&C)
C, A

```

5.3 Implementation of the SBML2BNET Pipeline

Along my PhD, I have made a point of following the best code practice I could, supporting reproducibility, and facilitating the installation of the pipeline. In particular, all the tools developed or reused are open-source, well documented and freely available. They are installed and run in a virtual environment managed with **Conda**, which simplifies the management of library dependencies and version conflicts [Ana21]. Moreover, we use the popular workflow manager **Snakemake** to describe the pipeline [Möl+21]. In **Snakemake**, the steps are described along with their dependencies on one another. **Snakemake** can thus orchestrate which steps must

¹⁷ The BNET format is described in the Appendix of the BoolNet's R vignette http://cran.nexr.com/web/packages/BoolNet/vignettes/BoolNet_package_vignette.pdf

be performed sequentially, and which steps that can be parallelised. It also ensures the upstream steps have succeeded before running the following steps. This helps to track potential problems, and overall, it makes the pipeline easy to use, even for users with limited programming skills. Still, more advanced users can easily extract parts of the pipeline or expand them based on their own research needs, for example, to replace the tools used with some others.

We use **Biocham** to determine the well-formedness of the models, and improve it when applicable [CFS06; FGS12]. In particular, **Biocham** removes reactions of null kinetics, splits in two the reactions that are implicitly reversible, and adds correct annotations for modifiers. SBML manipulation (in particular the structure extraction) is done with the library **libSBML** [Bor+08].

The numerical simulation is done with **Copasi** [Hoo+06], which is a program dedicated to the simulation and analysis of reaction networks extended with various elements (in particular rules and events). The solver LSODAR was used with the default parameter values.

As for the abstract simulation, the tool chain has been implemented in collaboration with Joachim Niehren and Cristian Versari. All this chain is integrated and uses preexisting tools from the **Reaction-Network** suite, created and maintained by the team BioComputing (from the lab CRISTAL). This suite can analyse and draw reaction networks in BioComputing’s XML format. The abstract simulation algorithm itself is implemented in **Python** which calls the constraints solver **Minizinc** [Net+07] from a given initial abstract configuration. Since the number of configurations in the transition graph we construct from an FO-BNN is exponential in the number of its species, our algorithm computes the transition graph by repeated iterations of constraint solving (see [algorithm 2](#)). In each step, the values of the variables in \mathcal{S} are constrained to the Boolean configurations from which the subgraph is to be explored. More precisely, for any FO-BNN ϕ with species in \mathcal{S} and abstract initial configuration $\gamma_0 : V \cup V_{\text{next}} \rightarrow \mathbb{S}$, the abstract simulation represented by the set S_{reach} of all reachable configurations and the corresponding transition relation T_{reach} can be obtained by iterative computation of the sets of new transitions T_{new} and new reachable configurations S_{new} starting from the initial configurations γ_0 . The algorithm ends when no new reachable configurations are obtained from the available transitions. To compute the full transition graph, we can thus run our algorithm starting from all the abstract configurations. But when we are only interested in a subgraph, that saves a lot of computations since we do not compute the transitions to unreachable configurations. Thanks to this strategy, we could explore the qualitative dynamics of models up to 27 species, such as the model B448 of the BioModels database¹⁸. For this model, the total number of abstract configurations is 2^{27} . However, by starting from the abstraction of the initial concentrations in the SBML model (all species are present except IRins), the transition graph is reduced to two edges between two configurations ([figure 5.2](#)). One of these configurations is the initial abstract configuration mentioned above, and the other is the 1-only bit vector. The latter is an attractor, as its exiting edge is making a self-loop. This is consistent with the concrete simulation of the model and the steady-state computed numerically, but independent of the precise initial concentrations chosen in the SBML model.

All the preprocessing and postprocessing needed before and after the abstract simulation heavily rely on intermediate files (to represent the ODEs, the transition graphs, diverse input parameters...). These files are automatically processed using XSLT stylesheets. The transition

¹⁸ The model is available at <https://www.ebi.ac.uk/biomodels/BIOMD0000000448>.

Algorithm 2 Algorithm for the abstract simulation of an FO-BNN ϕ with variables in V from an abstract initial configuration γ_0 . It computes the set of abstract configuration transitions T_{reach} .

Require: an FO-BNN ϕ with variables in V , an abstract configuration $\gamma_0 : V \cup V_{\text{next}} \rightarrow \mathbb{S}$

```

1: function ABS_SIM( $\phi, V, \gamma_0$ )
2:    $T_{\text{reach}} \leftarrow \emptyset$ 
3:    $S_{\text{reach}} \leftarrow \{\gamma_0\}$ 
4:    $S_{\text{new}} \leftarrow S_{\text{reach}}$ 
5:   while  $S_{\text{new}} \neq \emptyset$  do
6:      $T_{\text{new}} \leftarrow \text{trans} \circ \text{sol}^{\mathbb{S}}(\phi \wedge \bigvee_{\gamma \in S_{\text{new}}} \bigwedge_{x \in V \cup V_{\text{next}}} x \stackrel{\circ}{=} \gamma(x))$ 
7:      $S_{\text{new}} \leftarrow \{\gamma_2 \mid (\gamma_1, \gamma_2) \in T_{\text{new}}\} \setminus S_{\text{reach}}$ 
8:      $T_{\text{reach}} \leftarrow T_{\text{reach}} \cup T_{\text{new}}$ 
9:      $S_{\text{reach}} \leftarrow S_{\text{reach}} \cup S_{\text{new}}$ 
10:  return  $T_{\text{reach}}$ 

```

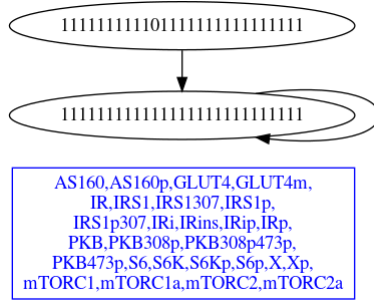


Figure 5.2: Abstract simulation of B448 starting from the abstraction of the initial configuration provided in the SBML model. In the Boolean vectors, the species are ordered like in the blue box.

graphs we reconstruct from the solution set of the constraints solved over the structure of signs are drawn via **Graphviz** [GN00]. Everything is orchestrated using **Make**.

For the Boolean network synthesis, we preprocess the data in **Python**. Then, as presented in [chapter 4](#), the core of the Boolean network synthesis uses Answer-Set Programming (ASP) with the system **clingo** [Geb+12]. ASP relies on constraints and logic to define the solution of a problem. Thanks to heuristics (inspired from SAT solvers), ASP performs a *clever* exhaustive search of all answer sets satisfying the given constraints. It is powerful and quite adapted to the local synthesis problem, as it is both a combinatorial and optimisation problem.

Finally, the library **PyBoolNet** [KSS16] is used to manipulate the synthesised Boolean networks and to compute their transition graphs, which we draw using **Graphviz** as well.

5.4 Evaluation of the SBML2BNET Pipeline

First, we show and discuss the Boolean networks that are synthesised for \mathcal{R}_{enz} ([example 1.9.4](#) on page 44), with the pipeline ran in different setting. In particular, we compare what is obtained when using the concrete simulation with different binarisation procedures, and what we get when using dynamical data retrieved with abstract simulation. Then, we present an extensive evaluation of the pipeline by processing 209 SBML models from the repository BioModels. This time, the dynamic data are exclusively retrieved from concrete simulations that reproduce the data used in the curation pipeline of BioModels. The data are then binarised using the midrange threshold. By this evaluation, we want to evaluate the impact of taking the rules and the events into account when constructing the prior influence graph, as well as whether the well-formedness of the models impacts the results.

The programs needed to reproduce the presented results are accessible at https://gitlab.inria.fr/avaginay/CNA2021_extension.

5.4.1 Results on \mathcal{R}_{enz}

We run the SBML2BNET pipeline with six different settings on the SBML file (see [Appendix A](#) on page 177) that models the enzymatic process ([example 1.9.4](#) on page 44). The settings are summarised in [table 5.1](#). When no prior influence graph is given, the structure of the synthesised

Table 5.1: Settings for the evaluation of the SBML2BNET pipeline on \mathcal{R}_{enz} . The influence graph \mathcal{P}_{enz} is depicted in [example 3.2.1](#) on page 77.

Setting	PIG \mathcal{P}_{enz}	Simulation	Binarisation	Error Constraint	minDNF Option
1a	✓	Concrete	Midrange	Soft	Mincard
1b	✓	Concrete	Median	Soft	Mincard
1c	✓	Concrete	Mean	Soft	Mincard
1d	✓	Concrete	Above 0	Soft	Mincard
2a	None	Abstract full	Above 0	Hard	All
2b	None	Abstract partial	Above 0	Hard	All

Boolean networks is not constrained at all. They can thus involve any influence between any species.

5.4.1.1 Settings 1a–d

We extract the influence graph of the reaction network with the routines presented in [section 3.2](#) on page 76. The dynamics is retrieved with a concrete simulation, and we compared several binarisation schemes: the default midrange-based, as well as with the median, mean, and “above 0” threshold.

[Table 5.2](#) summarises the results along with the sequences of Boolean configurations obtained with each binarisation scheme. We can see that the sequences are quite different from one another, and that they do not necessarily reflect the actual causation of the changes. That is, the fact that E and S have to be 1 for C to become 1 and then P in turn. This confirms our intuition from [chapter 3](#).

Table 5.2: Summary of the Boolean networks synthesised for \mathcal{R}_{enz} , using a concrete numerical simulation and different binarisation procedures to retrieve Boolean transitions.

Setting (Binarisation)	Boolean configuration sequence [S, E, C, P]	Coverage
1a (Midrange)	1100 \rightarrow 1000 \rightarrow 1010 \rightarrow 0010 \rightarrow 0011 \rightarrow 0101	0.8
1b (Median)	1100 \rightarrow 1010 \rightarrow 0011 \rightarrow 0101	0.6
1c (Mean)	1100 \rightarrow 1010 \rightarrow 1000 \rightarrow 0011 \rightarrow 0101	1
1d (Above 0)	1100 \rightarrow 1111 \rightarrow 1011 \rightarrow 1111 \rightarrow 0111	0.25
Expected	1100 \rightarrow * * 10 \rightarrow * * * 1	

Whatever the chosen binarisation, SBML2BNET synthesised a unique Boolean network. By construction, the synthesised Boolean networks respect the influence graph of the reaction network encoded by the given SBML file. Indeed, their respective influence graph is a spanning subgraph of the prior influence graph from [example 3.2.1](#) on page 77.

The Boolean network synthesised in setting 1a (midrange) is shown in [figure 5.3a](#). Its general-asynchronous transition graph (shown in [figure 5.3c](#)) covers 4 transitions out of the 5 extracted from the configuration sequence ([table 5.2](#)). The coverage ratio is thus 0.8, and the coverage median and standard deviation of this singleton of solutions are obviously 0.8 and 0 respectively, making SBML2BNET successful on this example. Using the synchronous update scheme, two fixed-point attractors are found for this Boolean network: [S:1, E:0, C:1, P:1] and [S:0, E:1, C:0, P:0]. They are consistent with what we would expect biologically. In particular as “nothing happens” if the dynamics starts with only E present while there is no S.

When using median, mean and “above 0” binarisation thresholds, the pipeline also returns a unique Boolean network with coverage of respectively 0.6, 1 and 0.25. The Boolean network obtained with the median-based binarisation is the same as the one we obtained when using the midrange-based binarisation ([figure 5.3a](#)). Yet, it has a smaller coverage because the sequence of configurations is slightly different. The Boolean network synthesised with the mean-based binarisation has the best coverage achievable ([table 5.2](#)). Still, we decided to stick with the midrange-based coverage for the rest of the experiments since it is the simplest binarisation and not influenced by periods of time where a species oscillates in a small range of values.

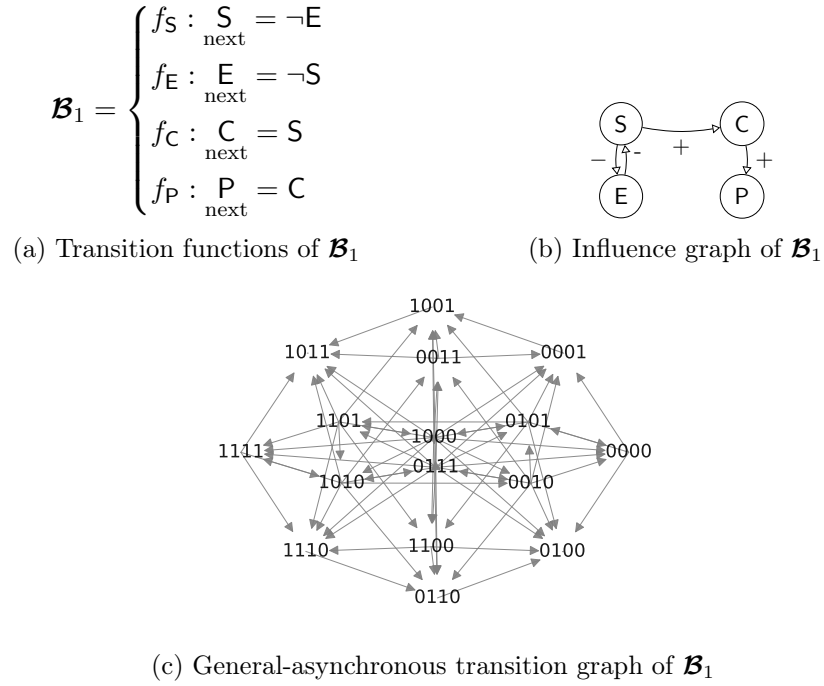


Figure 5.3: A Boolean network to model \mathcal{R}_{enz} (example 1.9.4). It was obtained automatically by running the SBML2BNET pipeline in settings 1a and 1b (see section 5.4.1.1 for more details).

5.4.1.2 Settings 2a–b

Hereafter, we only consider Boolean networks that explain the given dynamics *perfectly*. Indeed, the error constraint is hard, as shown in table 5.1. In the setting 2a, we apply the Boolean network synthesis step on the full transition graph computed by abstract simulation (figure 3.10). We obtain the following Boolean network:

$$\begin{cases} f_S : S_{\text{next}} = C \vee S \\ f_E : E_{\text{next}} = E \vee C \\ f_C : C_{\text{next}} = (E \wedge S) \vee C \\ f_P : P_{\text{next}} = C \vee P \end{cases}$$

This Boolean network is the only Boolean network retrieved, despite the option for minDNF set to “all”. Its transition functions are thus cardinal minimal. Moreover, it explains *perfectly* all the qualitative dynamics retrieved by abstract simulation. However, note that it is not compatible with the influence graph \mathcal{P}_{enz} (example 3.2.1 on page 77). Indeed, f_P involves P while $P \rightarrow P \notin \mathcal{P}$. We deduce from this experiment that it is not possible to get a Boolean network that is compatible with the full transition graph and that respect the prior influence graph. Even though the transition functions do not encode the direct causations among species, the synthesised Boolean network is actually very meaningful. It is quite close to the classic reasoning in biology which consists in unravelling the *contexts* in which species are present, instead of the direct causations of the changes.

In the setting 2b, we apply the Boolean network synthesis step on the transition graph restricted to the configurations accessible from the initial configuration [S:1, E:1, C:0, P:0]. This graph is shown in figure 3.11. The sequence of configurations is not very constraining, and without a prior influence graph \mathcal{P}_{enz} to constrain the synthesis, there are 37 564 918 660 497 000 possible Boolean networks. Among them, 14 respect the prior influence graph \mathcal{P}_{enz} and one has cardinal-minimal transition functions:

$$\begin{cases} f_S : S_{\text{next}} = 1 \\ f_C : C_{\text{next}} = 1 \\ f_P : P_{\text{next}} = C \\ f_E : E_{\text{next}} = 1 \end{cases}$$

We can see that this is not a very meaningful Boolean network as it sets most of the species to 1. This is because the given transition sequence does not visit a lot of the dynamic landscape.

5.4.2 Results on SBML Models from BioModels

5.4.2.1 Selection of the Models

In this evaluation, we apply the SBML2BNET pipeline on SBML models from BioModels. BioModels [Mal+20] is a repository of models of biological and biomedical systems, including metabolic networks, signalling networks, gene regulatory networks and infectious diseases. Each model stored in the `curated` branch of BioModels is encoded in SBML and has passed a manual curation process. The curation is two-fold: (1) the annotation of the elements of the model (for example, cross-referencing the species with other databases), (2) the assertion that the model reproduces the results from the paper in which it was originally published.

The latest available release of BioModels¹⁹ contains 640 curated SBML models, but only 369 are complete and can be processed by the SBML2BNET pipeline. Moreover, since the complexity of the Boolean network synthesis problem increases exponentially with the number of parents for each species, we only consider the 209 models for which all the species have less than ten incoming edges. The number of species in these models ranges from 1 to 61 (median = 8, std ~ 11), but models with more species would not have been a problem *per se* since ASK&D-BN is not directly impacted by the number of species. Among these 209 models, 38 models use rules and/or events (3 have both). Only 30 % (64) of these models are well-formed according to the tool Biocham.

5.4.2.2 Procedure and Evaluation Criteria

Given an SBML model, we want to evaluate the SBML2BNET pipeline according to the following criteria:

The runtime of the pipeline to attest that the pipeline scales to real-world SBML models.

¹⁹ release 31 <ftp://ftp.ebi.ac.uk/pub/databases/biomodels/releases/2017-06-26/>

The number of Boolean networks synthesised to assess that the problem is sufficiently constrained and that the pipeline does not return an overwhelming number of alternative Boolean networks, among which it would be difficult to choose.

The coverage ratio analysis to attest the compatibility of the synthesised Boolean networks with the corresponding timeseries by analysing the distribution of median and standard deviation summarising the coverage ratios

The constancy and monotony of the transition functions to check the parsimony of the transition functions of the synthesised Boolean networks.

Note that we do not evaluate the compliance of the synthesised Boolean network with the prior influence graph retrieved from the SBML model, because they are compliant by construction (chapter 4).

Moreover, we want to evaluate the impact on these criteria of the integration of the rules and events in the construction of the prior influence graph. Indeed, table 5.6 shows that adding rules and events increases the search space (it increases the number of parents of the species), as well as the number of species for which the prior structure is non-monotone. We thus run the pipeline in four distinct settings, summarised in table 5.3. Whatever the setting, we always retrieve the dynamics by a concrete simulation. The duration of the simulation corresponds to what is given in the curation reports of BioModels, and we use the default binarisation procedure (midrange).

Table 5.3: Settings for the evaluation of the SBML2BNET pipeline on SBML models from BioModels.

Setting	Prior influence graph built from. . .	Simulation	Binariesation	# models concerned
[r]	reactions only	Concrete	Midrange	209
[re]	reactions + events	Concrete	Midrange	3
[rr]	reactions + rules	Concrete	Midrange	38
[rre]	reactions + rules + events	Concrete	Midrange	3
Total # xp = 253				

5.4.2.3 Results

In this section, the term *experiment* refers to the run of the pipeline on a given SBML model and in a given setting (table 5.3).

Runtime

Table 5.4 and figure 5.4a show, for each setting, how many models were processed in less than 30 hours, as well as CPU time of the Boolean network synthesis step. Despite the complexity of the Boolean network synthesis step, about three fourth of the experiments (187/253) terminated in less than 30 hours. This corresponds to 155 SBML models. From figure 5.4a, we can see that the Boolean network synthesis step stopped processing an significant number of models after 10 hours. In the following, we report the results for the 187 experiments terminated in less than 30 hours.

To see how the addition of rules and/or events impacts the runtime of the local synthesis for a given model, we plot the runtimes for settings [re], [rr] and [rre] against the runtime obtained with the setting [r] (figure 5.4b). The dots are on the diagonal when there is no change, and above (resp. below) the diagonal when the addition of rules and/or events leads to bigger (resp. smaller) runtime. Quite surprisingly, despite the search space to be larger (table 5.6), adding rules and/or events does not necessarily increase the runtime.

Number of Boolean networks Synthesised

In each experiment, around 8 Boolean networks are synthesised in average. This number hides a strong disparity, since a single Boolean network was synthesised for almost 70% (126) of the experiments. table 5.5 shows the details for each setting. We can see that the choice of the setting does not have an impact on the median number of Boolean networks generated.

Compatibility of the Boolean networks with the Timeseries (Coverage Analysis)

To assess the coverage ratio criterion, we plot the median of the Boolean networks synthesised for each SBML model in figure 5.5. As said before, all the Boolean networks returned by the SBML2BNET pipeline for a given SBML model would ideally have a perfect coverage ratio, hence with a median of 1 and a standard deviation of 0. It is actually the case for almost 90% of the experiments in the setting [r] (139/155), and around 75% in all setting confounded (137/187). In the setting [r], the mean, median and standard deviation of the median coverage ratios of the Boolean networks synthesised are of 0.90, 1 and 0.19 respectively. There are only 4 experiments for which the standard deviation is not 0 (max = 0.22). Overall, the pipeline is efficient at finding Boolean networks with good coverage median and small standard deviation, whatever the considered setting. Nevertheless, there are experiments for which the coverage of the synthesised Boolean networks is not good. In particular, there is a significant loss of performance correlated to the number of nodes in the systems (Kendall's τ value of -0.19 , p-value of 0.001).

We are currently investigating possible reasons of this correlation, and reasons of poor coverage ratio in general. One reason could simply be that Boolean networks cannot explain all phenomena (section 1.8.2 on page 40): in some cases, the maximum achievable coverage ratio is smaller than 1, but our quality evaluation of the synthesised Boolean networks does not take this fact into account. We could use Boolean networks with the most-permissive semantics [Pau+20] to overcome this limitation. Unfortunately, no implementation is available for Boolean networks having non-monotone transition functions (such as the ones our pipeline might produce). We also speculated that another reason could be that the specifications of SBML leave open the possibility for a model to contain contradictory information. This speculation was taking its root from [FGS12], where it was shown that more than 60% of the SBML models tested in 2012 were not well-formed (definition 1.9.6 on page 45). For example, the model B44²⁰ has reactions with species used in the kinetics which are not listed as reactants nor modifiers. This has a bad impact on the construction of the PIG by our pipeline (section 3.2 on page 76), since potential parents of some species are not identified. For this particular model in setting [r], one Boolean network was generated, with a poor coverage of 0.55. Among the 187 experiments analysed (those that terminated in less than 30 hours), 131 concern not well-formed models and 56 concern well-formed models. However, the coverages obtained from well-formed models are not really different from the coverages obtained from not well-formed (means and median of 0.9 and 1 in

²⁰ <https://www.ebi.ac.uk/biomodels/B44>

Table 5.4: Number of SBML models processed and CPU time of the Boolean network synthesis step.

Setting	# to process	# done < 30 hours (%)	CPU time	
			median (minutes) \pm	std
[r]	209	155 (~ 75)	$\sim 33 \pm 5.6$	hours
[re]	3	2 (~ 66)	$\sim 28 \pm 14$	minutes
[rr]	38	29 (~ 75)	$\sim 7 \pm 6$	hours
[rre]	3	1 (~ 33)	~ 51	
Total	253	187 (~ 75)	$\sim 31 \pm 5.6$	hours

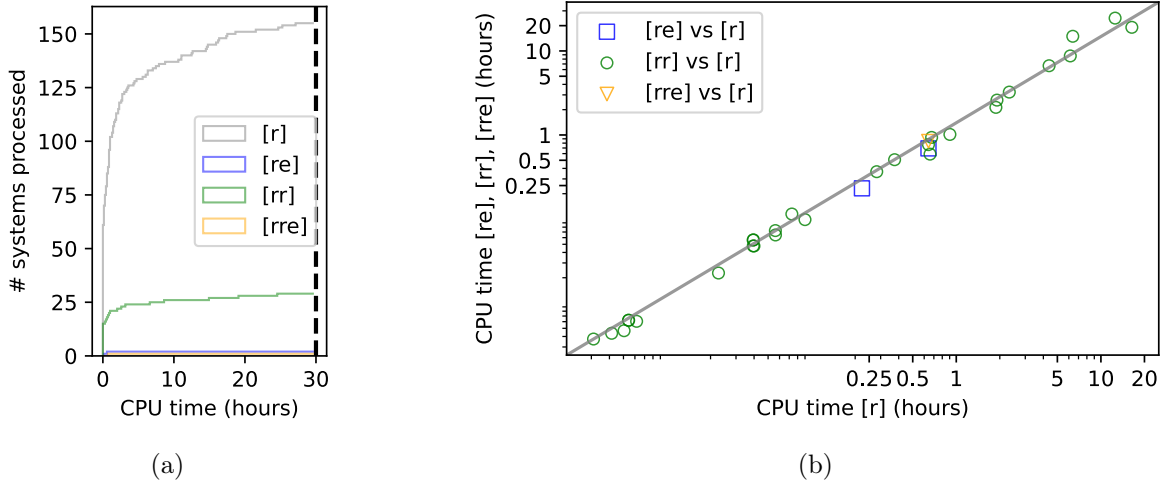


Figure 5.4: Runtime of the Boolean network synthesis step (a) in the four PIG settings (table 5.3) and (b) in comparison with the setting [r]. We only consider the 187 experiments that terminated in less than 30 hours. This corresponds to 155 SBML models.

Table 5.5: Number of Boolean networks synthesised by the SBML2BNET pipeline. The values are computed by taking into account the 187 experiments that terminated in less than 30 hours.

Setting	Median # BNs
[r]	1
[re]	1.5
[rr]	1
[rre]	1
All	1

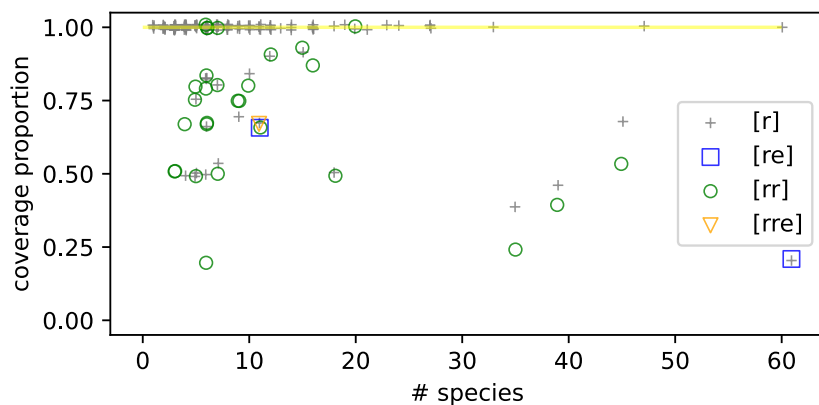
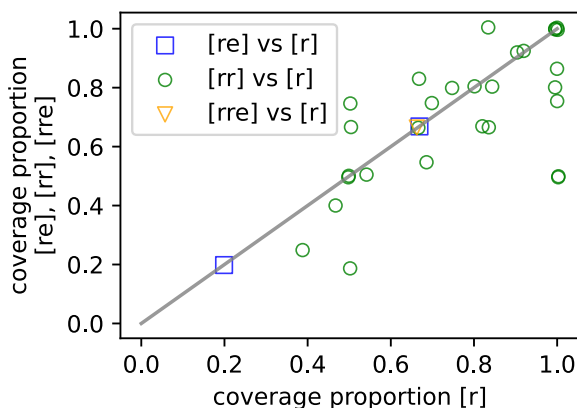


Figure 5.5: Coverage evaluation for the Boolean networks synthesised by SBML2BNET for 155 SBML models (corresponding to the 187 experiments that terminated in less than 30 hours). Each dot represents the set of Boolean networks returned for a given SBML model in a given setting. Its coordinates are the coverage ratio median (ordinate) and the number of species of the SBML model (abscissa). The yellow line shows where the dots are when the pipeline only returns Boolean networks with a perfect coverage. To ensure readability, the points are slightly jittered on x and y axes with a Gaussian noise of variance 0.2 and 0.02 respectively.



Setting	# xp	\nearrow	\searrow	=
[re]	2			2
[rr]	29	13	8	8
[rre]	1			1
Total	32	13	8	11

Figure 5.6: Evaluation of the impact on the coverage of constructing the PIG with rules and/or events. The points are jittered with a Gaussian noise of variance 0.01 on both axes to ensure readability.

both cases). We also hypothesised contradictions were most likely to occur in bigger models, but this is not the case either. Indeed, in the complete set of models we started with (209 models) the median size of the 64 well-formed models is of 9.5 versus 7 for the 145 not well-formed models.

Let us now consider specifically the impact of a PIG built with rules and/or reactions on the coverage results. For a given SBML model, we check how the coverages in setting [re], [rr] and [rre] differ from the ones obtained in setting [r] (figure 5.6). We can see that adding events does not impact the coverage of the synthesised Boolean networks. In fact, we actually obtain the exact same Boolean networks. Adding rules, however, has a mixed impact. There are 8 experiments for which it changed nothing, but 13 for which it improves the coverage and 8 for which it decreases the coverage. We are planning to investigate automatic ways to determine in advance which rules are worthy to be considered for the PIG construction.

Constancy and Monotony Analysis

table 5.6 reports the impact of the introduction of rules and events on the PIG and the synthesised functions in terms of parenthood and monotony. The analysis of constancy and monotony of the synthesised functions is done on the influence graph of a Boolean network obtained by merging the Boolean networks solutions.

The number of species without any potential parent in the PIG (for which the synthesised function is then the constant False by default) decreases when adding rules and/or events. As a result, 4 species in the setting [rr] led to synthesised functions which are not the constant function (the default one).

Concerning the monotony, the introduction of rules and events leads by construction to non-monotone influences in the PIG. Hence, in settings [re], [rr] and [rre], the number of species for which the PIG contains at least one non-monotone influence increases compared to what is observed for the setting [r]. However, the synthesised functions are all monotone.

5.5 Wrap-up, Discussion, and Perspectives

In this chapter, we presented the SBML2BNET pipeline. It takes as input a reaction network encoded in the Systems Biology Markup Language (SBML), and synthesises a set of Boolean networks encoded in the BNET format. The transformation is designed such that the synthesised Boolean networks are compatible with the structure and the dynamics of the input reaction network. Our evaluation demonstrates the effectiveness of the pipeline in generating valuable Boolean networks that comply with the input reaction network, according to our criteria. Thanks to the modularity of the pipeline, we could in fact explore the results of the pipeline ran with different settings. In particular, we discussed the limitations and difficulties faced when processing models from the literature, in particular, those that use features such as discontinuous events and rules. Moreover, our experiments with \mathcal{R}_{enz} really help to grasp several limitations of our approach.

First, when using concrete simulation, we only have one multivariate timeseries. It most likely doesn't visit a lot of the dynamic landscape. Moreover, the concrete simulation loses the causation of the changes, and the binarisation does not recover it either. Because of this, most of the species are associated with a constant transition function. Indeed, the species are not really varying, according to the extracted sequences of configurations. Also, with this variant of

the pipeline, the choice of the binarisation has a strong impact on the retrieved configuration sequences.

By using abstract simulation, we can explore more of the dynamic landscape. However, in this case, if we impose the influence graph to represent the direct influences among the species, it might be that the synthesised Boolean networks cannot explain all the dynamics. In contrast, if we relax the structure constrain, we might get really meaningful Boolean networks, explaining all the dynamics, but they rather encode the *contexts* of activation instead of the actual biological causation. In this case, the structure of the synthesised Boolean networks are not compatible with the prior influence graph we extracted from the input reaction network.

This led us to reconsider the role of Boolean networks in systems biology, and more globally, the use of a prior influence graph in the Boolean network synthesis tool. Indeed, Boolean networks are often seen as the ultimate abstraction of reaction networks. From this perspective, they are assumed to encode the biological mechanisms. Following this viewpoint, most of the existing Boolean network synthesis methods use a prior influence graph as a *hard* constraint for the influence graph of the synthesised Boolean networks. As we saw, such a hard constraint is ideal for reducing the search space of the Boolean network synthesis problem that is exponential by nature. However, since the influence graph of a synthesised Boolean network cannot use an influence that is not allowed by the prior influence graph, it is essential to find a good balance between reducing the search space and the freedom left for the synthesis. In other words, the prior influence graph needs to be relaxed enough to capture the appropriate solutions.

All in one, considering the structure constraint as a hard constraint might not necessarily be a good thing. In fact, it depends on whether we conceptualise the synthesised Boolean network as a mechanism-based model, or as an influence-based model. In conclusion, one needs to be really careful not to use a prior influence graph only for technical reasons (reducing the search space), and that the influences that are meaningful for the model they want to achieve are all considered.

We now point out several improvements we could make on the SBML2BNET pipeline. First, the pipeline runtime is quite long on some models. This is due to the Boolean network synthesis step. However, the runtime performance of ASK&D-BN is tightly related to the encoding in answer-set programming, and we might not have an optimal encoding. Second, we could use better evaluation metrics. So far, we use the coverage computed with the general-asynchronous update scheme. It might not be the best evaluation measure. Indeed, due to their Boolean nature, Boolean networks cannot represent all phenomena: it is proven that, in some cases, it is impossible to find a Boolean network in which a given sequence of configurations would be a walk in the general-asynchronous transition graph [Pau+20]. It follows that even the *most* compatible Boolean networks one can find necessarily have a coverage ratio smaller than 1. It would be interesting to be able to estimate the virtual maximum coverage ratio one can expect given the provided constraints, and to normalise our quality evaluation by this value. Alternatively, we could rely on the most permissive update scheme, as it has been proven to capture any possible quantitative behaviour. Unfortunately, no implementation currently exists to process Boolean networks with non-bipolar transition functions, that our pipeline may produce (even though we saw most of the synthesised functions are bipolar anyway).

Table 5.6: Impact of the setting on the search space and synthesised functions in terms of monotony and constancy.

Setting vs [r]	# models concerned	# species concerned	# species in the FIG, compared to [r]		# species from IG of merged BNs, compared to [r]	
			w/ potential parents	w/ only monotone incoming influence	Constant	Monotone
[re]	2	72	50 vs 46 → +4	25 vs 29 → -4	70 vs 70 → 0	72 vs 72 → 0
[rr]	29	333	309 vs 234 → +75	122 vs 197 → -77	306 vs 310 → -4	333 vs 333 → 0
[rre]	1	11	10 vs 7 → +3	2 vs 5 → -3	10 vs 10 → 0	11 vs 11 → 0

General Conclusion

We start this conclusion by recalling the questions raised in the general introduction. Then, we summarise the answers we formulated throughout all the thesis. We close the thesis with some limitations we identified for the proposed approach as well as some future work we deem of interest.

- Q1** What are the advantages of a Boolean network, compared to a reaction network? In which cases is a Boolean network preferable to a reaction network?
- Q2** What information can we use to synthesise Boolean networks?
- Q3** How can this information be retrieved from a reaction network?
- Q4** How can this information be used to define the notion of *compatibility* of a Boolean network with a reaction network?
- Q5** How to synthesise automatically such a compatible Boolean network?
- Q6** Does a compatible Boolean network always exist?

First, from the literature review presented in [chapter 2](#), we saw that reaction networks and Boolean networks are the respective epitome of two classes of models (namely mechanism-based and influence models), with very different philosophies. We addressed **Q1** by showing some cases in which Boolean networks are handier than reaction networks: in particular for tasks such as the model construction itself, as well as model simplification, aggregation and control. These findings are in line with the widespread opinion that the change of viewpoint, from a mechanism-based perspective to an influence-based perspective, brings several benefits and insights, and that models from both classes should be used in parallel to benefit from their respective strengths.

The answer for **Q2** is also from [chapter 2](#), where we found out that existing model construction methods (and in particular those to construct Boolean networks) classically consist in fitting the constructed models against knowledge and data about the structure (list of species and their influences on one another) and the dynamics (list of behavioural patterns of the species) of the system under study.

In [chapter 3](#), we thus focused on answering **Q3**. We defined the structure of a reaction network as a directed signed graph that summarises the direct influences among the species. We showed how to retrieve such a graph by simply parsing the input model. As for the dynamics, we explored two methods. Both are based on the simulation of ordinary differential equations (ODEs) reconstructed from the reaction network.

Concrete simulation The first simulation is a concrete simulation, such as classically implemented in the tools available to study the dynamics of reaction networks. Once the data has been binarised, we end up with sequences of Boolean configurations.

Abstract simulation The second simulation is referred to as abstract simulation, as it relies on the abstract interpretation of a first-order formula constructed from the ODEs, the so-called First-Order Boolean Network with Nondeterministic updates (FO-BNN). So far, such a formula was only developed for core reaction networks. It involves variables that represent the value of the species (which cannot be negative) along with variables that represent the derivatives (which can be negative). The interpretation of an FO-BNN formula using values representing the sign of the variables (hence only Booleans for the species variables), instead of the concrete values defines a reachability relation between the Boolean configurations of the system.

The first type of simulation may have seem naive at first glance, but we saw that the problem was overall not that simple. In particular because ODEs of biological systems exhibit so-called stiff behaviours (a mix of slow and rapid evolutions). We thus had to use a *clever* simulation algorithm that balances the trade-off between the accuracy and the efficiency of the simulation. Moreover, as it is more direct, the concrete simulation allows us to process reaction networks for which the abstract simulation is not yet applicable. In particular, if the reaction network is coupled with discontinuous events, which is the case for real-life models available in databases (such as those processed in [chapter 5](#)). Conversely, the abstract simulation gets its strength in that the resulting Boolean transition graph overapproximates the qualitative behaviours of the ODEs, relatively to Euler simulation, and that it thus preserves the causations of the changes.

Our proposition to answer **Q4** was presented in [chapter 4](#). The notion of compatibility was defined in terms of the structure (prior influence graph) and the dynamics (either a truth table or a timeseries) retrieved from the input reaction network. More precisely:

- A Boolean network is compatible with the prior structure if its influence graph is a spanning subgraph of the given prior.
- A Boolean network is compatible with the given dynamics if its general-asynchronous transition graph covers a good proportion (ideally 1) of the transitions between Boolean configurations retrieved from it.

The definition can in fact be extended to any structure (prior influence graph) and any dynamics (either a truth table or a timeseries), not necessarily extracted from a reaction network.

To answer **Q5**, we formalised the Boolean network synthesis problem as a constraint satisfaction and optimisation problem. The details on the proposed encoding, which uses the answer-set programming framework, can be found in [chapter 4](#). The SBML2BNET pipeline, which was presented in [chapter 5](#) puts together the implementation of all the methods mentioned so far, and focuses on the conversion of reaction networks, encoded in the System Biology Markup Language (SBML), into compatible Boolean networks, encoded in the BNET file format.

Finally, **Q6** has been answered mostly experimentally, in [chapter 4](#) but mostly in [chapter 5](#), where we applied the pipeline on simple reaction networks and more complex ones, retrieved from the BioModels repository. We observed that the SBML2BNET pipeline is always able to

retrieve a set of Boolean networks with the best compatibility, according to our criteria, but that most of the Boolean networks we synthesised do not have a coverage of 1. This observation is in line with more theoretical-grounded results from the literature, which state that because of their Boolean nature, Boolean networks are not able to represent every possible quantitative behaviour [Pau+20].

In summary, we proposed a pipeline to synthesise *automatically* a set of Boolean networks compatible with a given reaction network. As presented in [chapter 2](#), the transformation of biological models from one formalism to another has been investigated in several papers. Many of such transformations are not fully automatised and heavily rely on expert choices and knowledge. Since our approach automatises, whenever possible, the transformation of reaction networks into Boolean networks, we can safely assume it reduces the risk of errors and saves the effort and time of biologists. Moreover, the SBML2BNET pipeline is modular, so one can build upon it to explore alternative conversion methods. This is important since many choices had to be made when setting up the pipeline. Among these choices, we can mention for example the fact of having focused on the differential semantics to retrieve the dynamics, the choice of binarisation, or the choice of implementations. Now, let us mention some perspectives we consider interesting to explore.

So far, we haven't made any application of the Boolean network we synthesised. One thing we could start with concerns the aggregation of Boolean networks from several reaction network models when they concern distinct parts of the same biological system. We could also investigate how to take benefit from the *set* of BNs synthesised for a given SBML model by combining and simulating them together, as recently proposed in [Che+20]. More generally, we would like to be able to assess the relevance of the synthesised Boolean networks in regard to the underlying biological system, but this sounds like something that only experts of the considered system could do reliably. There is thus a need for such experts to get access to the SBML2BNET pipeline so they can experiment with their own variants of the pipeline. Making the pipeline accessible and easy to use would thus be a natural perspective. We could do that by the intermediate of the CoLoMoTo consortium, for example.

In parallel, it would be interesting to make the pipeline be able to process more models. To do so, we have to investigate performance issues when processing models involving species with many incoming influences (more than ten). We also would like to be able to perform abstract simulation on models with missing parameter values, as well as on models extended with additional features such as events. Unfortunately, for now, SBML lacks formal semantics we could rely on to extend our FO-BNN formalisation we are using for abstract simulation.

Bibliography

- [Abo+14] W. Abou-Jaoudé, P. T. Monteiro, A. Naldi, M. Grandclaude, V. Soumelis, C. Chaouiya, and D. Thieffry. “Model Checking to Assess T-helper Cell Plasticity”. In: *Frontiers in Bioengineering and Biotechnology* 2 (2014), p. 86. DOI: [10.3389/fbioe.2014.00086](https://doi.org/10.3389/fbioe.2014.00086) (cit. on p. 48).
- [Aga+12] A. Agarwal, R. Adams, G. C. Castellani, and H. Z. Shouval. “On the Precision of Quasi Steady State Assumptions in Stochastic Dynamics”. In: *The Journal of Chemical Physics* 137.4 (2012), p. 044105. DOI: [10.1063/1.4731754](https://doi.org/10.1063/1.4731754) (cit. on p. 69).
- [AAL93] R. P. Agarwal, R. P. Agarwal, and V. Lakshmikantham. *Uniqueness and Nonuniqueness Criteria for Ordinary Differential Equations*. World Scientific, 1993. 328 pp. ISBN: 978-981-02-1357-2 (cit. on p. 29).
- [AD22] S. S. Aghamiri and F. Delaplace. “TaBooN Boolean Network Synthesis Based on Tabu Search”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 19.4 (2022), pp. 2499–2511. DOI: [10.1109/TCBB.2021.3063817](https://doi.org/10.1109/TCBB.2021.3063817) (cit. on pp. 61, 119, 122).
- [Aka74] H. Akaike. “A New Look at the Statistical Model Identification”. In: *IEEE Transactions on Automatic Control* 19.6 (1974), pp. 716–723. DOI: [10.1109/TAC.1974.1100705](https://doi.org/10.1109/TAC.1974.1100705) (cit. on p. 64).
- [AMK99] T. Akutsu, S. Miyano, and S. Kuhara. “Identification of Genetic Networks from a Small Number of Gene Expression Patterns under the Boolean Network Model”. In: *Pacific Symposium on Biocomputing*. 1999, pp. 17–28. DOI: [10.1142/9789814447300_0003](https://doi.org/10.1142/9789814447300_0003) (cit. on pp. 124, 133).
- [AMK00] T. Akutsu, S. Miyano, and S. Kuhara. “Inferring Qualitative Relations in Genetic Networks and Metabolic Pathways”. In: *Bioinformatics* 16.8 (2000), pp. 727–734. DOI: [10.1093/bioinformatics/16.8.727](https://doi.org/10.1093/bioinformatics/16.8.727) (cit. on pp. 61, 124).
- [All21] E. Allart. “Abstractions de Différences Exactes de Réseaux de Réactions : Améliorer La Précision de Prédiction de Changements de Systèmes Biologiques”. PhD thesis. Université de Lille (2018-2021), 2021. URL: <https://www.theses.fr/2021LILUI013> (cit. on pp. 23, 78, 95).
- [ANV19] E. Allart, J. Niehren, and C. Versari. “Computing Difference Abstractions of Metabolic Networks Under Kinetic Constraints”. In: *Computational Methods in Systems Biology*. Vol. 11773. Lecture Notes in Computer Science. Springer, 2019, pp. 266–285. DOI: [10.1007/978-3-030-31304-3_14](https://doi.org/10.1007/978-3-030-31304-3_14) (cit. on p. 102).
- [ANV21] E. Allart, J. Niehren, and C. Versari. “Exact Boolean Abstraction of Linear Equation Systems”. In: *Computation* 9.11 (2021), p. 113. DOI: [10.3390/computation9110113](https://doi.org/10.3390/computation9110113) (cit. on p. 101).
- [Alo19] U. Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Second edition. Boca Raton, Fla: CRC Press, 2019. ISBN: 978-1-4398-3717-7 978-1-138-49011-6 (cit. on pp. 1, 63, 65).
- [Ana21] Anaconda Software Distribution. *Conda*. 2021. URL: <https://github.com/conda/conda> (cit. on p. 138).
- [AK11] D. F. Anderson and T. G. Kurtz. “Continuous Time Markov Chain Models for Chemical Reaction Networks”. In: *Design and Analysis of Biomolecular Circuits*. New York, NY: Springer New York, 2011, pp. 3–42. DOI: [10.1007/978-1-4419-6766-4_1](https://doi.org/10.1007/978-1-4419-6766-4_1) (cit. on p. 55).

- [AKM09] E. Antezana, M. Kuiper, and V. Mironov. “Biological Knowledge Management: The Emerging Role of the Semantic Web Technologies”. In: *Briefings in Bioinformatics* 10.4 (2009), pp. 392–407. DOI: [10.1093/bib/bbp024](#) (cit. on p. 3).
- [ADG04] J. Aracena, J. Demongeot, and E. Goles. “On Limit Cycles of Monotone Functions with Symmetric Connection Graph”. In: *Theoretical Computer Science* 322.2 (2004), pp. 237–244. DOI: [10.1016/j.tcs.2004.03.010](#) (cit. on p. 36).
- [AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. 1st ed. Cambridge University Press, 2009. DOI: [10.1017/CB09780511804090](#) (cit. on p. 13).
- [AFK93] W. Auzinger, R. Frank, and G. Kirlinger. “Modern Convergence Theory for Stiff Initial-Value Problems”. In: *Journal of Computational and Applied Mathematics* 45.1-2 (1993), pp. 5–16. DOI: [10.1016/0377-0427\(93\)90260-I](#) (cit. on p. 89).
- [Bac+60] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. V. Wijngaarden, and M. Woodger. “Report on the Algorithmic Language ALGOL 60”. In: *Communications of the ACM* 3.5 (1960) (cit. on p. 14).
- [Bak+18] R. E. Baker, J.-M. Peña, J. Jayamohan, and A. Jérusalem. “Mechanistic Models versus Machine Learning, a Fight Worth Fighting for the Biological Community?” In: *Biology Letters* 14.5 (2018), p. 20170660. DOI: [10.1098/rsbl.2017.0660](#) (cit. on pp. 2, 4).
- [BS20] M. Banwarth-Kuhn and S. Sindi. “How and Why to Build a Mathematical Model: A Case Study Using Prion Aggregation”. In: *Journal of Biological Chemistry* 295.15 (2020), pp. 5022–5035. DOI: [10.1074/jbc.REV119.009851](#) (cit. on p. 3).
- [BO04] A.-L. Barabási and Z. N. Oltvai. “Network Biology: Understanding the Cell’s Functional Organization”. In: *Nature Reviews Genetics* 5.2 (2004), pp. 101–113. DOI: [10.1038/nrg1272](#) (cit. on p. 63).
- [Bar+21] R. Barbuti, P. Bove, R. Gori, D. Gruska, F. Levi, and P. Milazzo. “Encoding Threshold Boolean Networks into Reaction Systems for the Analysis of Gene Regulatory Networks”. In: *Fundamenta Informaticae* 179.2 (2021), pp. 205–225. DOI: [10.3233/FI-2021-2021](#) (cit. on p. 71).
- [BGM21] R. Barbuti, R. Gori, and P. Milazzo. “Encoding Boolean Networks into Reaction Systems for Investigating Causal Dependencies in Gene Regulation”. In: *Theoretical Computer Science* 881 (2021), pp. 3–24. DOI: [10.1016/j.tcs.2020.07.031](#) (cit. on p. 71).
- [Bar+20] R. Barbuti, R. Gori, P. Milazzo, and L. Nasti. “A Survey of Gene Regulatory Networks Modelling Methods: From Differential Equations, to Boolean and Qualitative Bioinspired Models”. In: *Journal of Membrane Computing* 2.3 (2020), pp. 207–226. DOI: [10.1007/s41965-020-00046-y](#) (cit. on p. 50).
- [BK17] S. Barman and Y.-K. Kwon. “A Novel Mutual Information-Based Boolean Network Inference Method from Time-Series Gene Expression Data”. In: *PLoS ONE* 12.2 (2017), e0171097. DOI: [10.1371/journal.pone.0171097](#) (cit. on pp. 63, 64).
- [BK18] S. Barman and Y.-K. Kwon. “A Boolean Network Inference from Time-Series Gene Expression Data Using a Genetic Algorithm”. In: *Bioinformatics* 34.17 (2018), pp. i927–i933. DOI: [10.1093/bioinformatics/bty584](#) (cit. on pp. 66, 119).
- [BK20] S. Barman and Y.-K. Kwon. “A Neuro-Evolution Approach to Infer a Boolean Network from Time-Series Gene Expressions”. In: *Bioinformatics* 36 (2020), pp. i762–i769. DOI: [10.1093/bioinformatics/btaa840](#) (cit. on p. 64).
- [Bas+21] P. Basu, J. H. Kim, S. Saeed, and M. Martins-Green. “Using Systems Biology Approaches to Identify Signalling Pathways Activated during Chronic Wound Initiation”. In: *Wound Repair and Regeneration* 29.6 (2021), pp. 881–898. ISSN: 1524-475X (cit. on p. 1).
- [Bat+05] G. Batt, D. Ropers, H. De Jong, J. Geiselman, R. Mateescu, M. Page, and D. Schneider. “Validation of Qualitative Models of Genetic Regulatory Networks by Model Checking: Analysis of the Nutritional Stress Response in Escherichia Coli”. In: *Bioinformatics* 21 (Suppl 1 2005), pp. i19–i28. DOI: [10.1093/bioinformatics/bti1048](#) (cit. on p. 47).

- [BT04] D. Battogtokh and J. J. Tyson. “Bifurcation Analysis of a Model of the Budding Yeast Cell Cycle”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 14.3 (2004), pp. 653–661. DOI: [10.1063/1.1780011](https://doi.org/10.1063/1.1780011) (cit. on p. 29).
- [BS17] R. Baumann and H. Strass. “On the Number of Bipolar Boolean Functions”. In: *Journal of Logic and Computation* 27.8 (2017), pp. 2431–2449. DOI: [10.1093/logcom/exx025](https://doi.org/10.1093/logcom/exx025) (cit. on p. 36).
- [Bea+22] L.-B. Beaufort, P.-Y. Massé, A. Reboulet, and L. Oudre. “Network Reconstruction Problem for an Epidemic Reaction-Diffusion”. In: *Journal of Complex Networks* 10.6 (2022), cnac047. DOI: [10.1093/comnet/cnac047](https://doi.org/10.1093/comnet/cnac047) (cit. on p. 49).
- [Ben+22] N. Beneš, L. Brim, J. Kadlecak, S. Pastva, and D. Šafránek. “Exploring Attractor Bifurcations in Boolean Networks”. In: *BMC Bioinformatics* 23.1 (2022), p. 173. DOI: [10.1186/s12859-022-04708-9](https://doi.org/10.1186/s12859-022-04708-9) (cit. on p. 68).
- [BN13] N. Berestovsky and L. Nakhleh. “An Evaluation of Methods for Inferring Boolean Networks from Time-Series Data”. In: *PLoS ONE* 8.6 (2013), e66031. DOI: [10.1371/journal.pone.0066031](https://doi.org/10.1371/journal.pone.0066031) (cit. on p. 91).
- [Bia18] C. Biane. “Reprogrammation Comportementale : Modèles, Algorithmes et Application Aux Maladies Complexes”. PhD thesis. Université Paris-Saclay (ComUE), 2018. URL: <https://www.theses.fr/2018SACLE050> (cit. on p. 67).
- [BD17] C. Biane and F. Delaplace. “Abduction Based Drug Target Discovery Using Boolean Control Network”. In: *Computational Methods in Systems Biology. Lecture Notes in Computer Science*. Springer International Publishing, 2017, pp. 57–73. ISBN: 978-3-319-67471-1 (cit. on p. 49).
- [Bli88] W. D. Blizard. “Multiset Theory.” In: *Notre Dame Journal of Formal Logic* 30.1 (1988). DOI: [10.1305/ndjfl/1093634995](https://doi.org/10.1305/ndjfl/1093634995) (cit. on p. 12).
- [Blo+18] P. Bloomingdale, V. A. Nguyen, J. Niu, and D. E. Mager. “Boolean Network Modeling in Systems Pharmacology”. In: *Journal of Pharmacokinetics and Pharmacodynamics* 45.1 (2018), pp. 159–180. DOI: [10.1007/s10928-017-9567-4](https://doi.org/10.1007/s10928-017-9567-4) (cit. on p. 49).
- [Bor05] S. Bornholdt. “Less Is More in Modeling Large Genetic Networks”. In: *Science* 310.5747 (2005), pp. 449–451 (cit. on p. 4).
- [Bor+08] B. J. Bornstein, S. M. Keating, A. Jouraku, and M. Hucka. “LibSBML: An API Library for SBML”. In: *Bioinformatics* 24.6 (2008), pp. 880–881. DOI: [10.1093/bioinformatics/btn051](https://doi.org/10.1093/bioinformatics/btn051) (cit. on p. 139).
- [Bra+82] R. K. Brayton, G. D. Hachtel, L. A. Hemachandra, R. A. Newton, and A. L. Sangiovanni-Vincentelli. “A Comparison of Logic Minimization Strategies Using ESPRESSO: An APL Program Package for Partitioned Logic Minimalization.” In: *IEEE International Symposium on Circuits and Systems* 1982 (Pt.1 1982), pp. 42–48. ISSN: 0271-4302 (cit. on pp. 35, 63, 125).
- [Bre10] R. Breitling. “What Is Systems Biology?” In: *Frontiers in Physiology* 1 (2010). ISSN: 1664-042X (cit. on p. 1).
- [Cal+06] L. Calzone, N. Chabrier-Rivier, F. Fages, and S. Soliman. “Machine Learning Biochemical Networks from Temporal Logic Properties”. In: *Transactions on Computational Systems Biology VI*. Red. by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, and G. Weikum. Vol. 4220. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 68–94. DOI: [10.1007/11880646_4](https://doi.org/10.1007/11880646_4) (cit. on p. 66).
- [CFS06] L. Calzone, F. Fages, and S. Soliman. “BIOCHAM: An Environment for Modeling Biological Systems and Formalizing Experimental Knowledge”. In: *Bioinformatics* 22.14 (2006), pp. 1805–1807. DOI: [10.1093/bioinformatics/btl1172](https://doi.org/10.1093/bioinformatics/btl1172) (cit. on pp. 66, 72, 97, 139).
- [CMA05] N. Cannata, E. Merelli, and R. B. Altman. “Time to Organize the Bioinformatics Resourceome”. In: *PLoS Computational Biology* 1.7 (2005), e76. DOI: [10.1371/journal.pcbi.0010076](https://doi.org/10.1371/journal.pcbi.0010076) (cit. on p. 3).
- [CFS05] N. Chabrier-Rivier, F. Fages, and S. Soliman. “The BIOChemical Abstract Machine BIOCHAM”. In: *Computational Methods in Systems Biology*. Vol. 3082. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 172–191. DOI: [10.1007/978-3-540-25974-9_14](https://doi.org/10.1007/978-3-540-25974-9_14) (cit. on p. 72).

- [CRF20] C. Chase Huizar, I. Raphael, and T. G. Forsthuber. “Genomic, Proteomic, and Systems Biology Approaches in Biomarker Discovery for Multiple Sclerosis”. In: *Cellular Immunology* 358 (2020), p. 104219. DOI: [10.1016/j.cellimm.2020.104219](https://doi.org/10.1016/j.cellimm.2020.104219) (cit. on p. 1).
- [Che+19a] H. Chen, L. Albergante, J. Y. Hsu, C. A. Lareau, G. Lo Bosco, J. Guan, S. Zhou, A. N. Gorban, D. E. Bauer, M. J. Aryee, D. M. Langenau, A. Zinovyev, J. D. Buenrostro, G.-C. Yuan, and L. Pinello. “Single-Cell Trajectories Reconstruction, Exploration and Mapping of Omics Data with STREAM”. In: *Nature Communications* 10.1 (2019), p. 1903. DOI: [10.1038/s41467-019-09670-4](https://doi.org/10.1038/s41467-019-09670-4) (cit. on pp. 61, 122).
- [Che+14] S. Chen, A. Ansari, W. Sterrett, K. Hurley, J. Kemball, J. C. Weddell, and P. I. Imoukhuede. “Current State-of-The-Art and Future Directions in Systems Biology”. In: *Progress and Communication in Sciences* 1.1 (1 2014), pp. 12–26. DOI: [10.21535/pcs.v1i1.148](https://doi.org/10.21535/pcs.v1i1.148) (cit. on p. 1).
- [Che22] S. Chevalier. “Inférence logique de réseaux booléens à partir de connaissances et d’observations de processus de différenciation cellulaire”. PhD thesis. Université Paris-Saclay, 2022. URL: <https://theses.hal.science/tel-03917566> (cit. on pp. 60, 62, 63, 125).
- [Che+19b] S. Chevalier, C. Froidevaux, L. Paulevé, and A. Zinovyev. “Synthesis of Boolean Networks from Biological Dynamical Constraints Using Answer-Set Programming”. In: *International Conference on Tools with Artificial Intelligence*. IEEE, 2019, pp. 34–41. DOI: [10.1109/ICTAI.2019.00014](https://doi.org/10.1109/ICTAI.2019.00014) (cit. on pp. 48, 119).
- [Che+20] S. Chevalier, V. Noël, L. Calzone, A. Zinovyev, and L. Paulevé. “Synthesis and Simulation of Ensembles of Boolean Networks for Cell Fate Decision”. In: *Computational Methods in Systems Biology*. Vol. 12314. Cham: Springer International Publishing, 2020, pp. 193–209. DOI: [10.1007/978-3-030-60327-4_11](https://doi.org/10.1007/978-3-030-60327-4_11) (cit. on pp. 104, 124, 133, 155).
- [Chi+15] H. J. K. Chiang, F. Fages, J. H. R. Jiang, and S. Soliman. “Hybrid Simulations of Heterogeneous Biochemical Models in SBML”. In: *ACM Transactions on Modeling and Computer Simulation* 25.2 (2015), pp. 1–22. DOI: [10.1145/2742545](https://doi.org/10.1145/2742545) (cit. on p. 71).
- [Cho+08] J. Choi, K.-w. Yang, T.-y. Lee, and S. Y. Lee. “New Time-Scale Criteria for Model Simplification of Bio-Reaction Systems”. In: *BMC Bioinformatics* 9.1 (2008), p. 338. DOI: [10.1186/1471-2105-9-338](https://doi.org/10.1186/1471-2105-9-338) (cit. on p. 68).
- [CL12] T.-H. Chueh and H. H.-S. Lu. “Inference of Biological Pathway from Gene Expression Profiles by Time Delay Boolean Networks”. In: *PLoS ONE* 7.8 (2012), e42095. DOI: [10.1371/journal.pone.0042095](https://doi.org/10.1371/journal.pone.0042095) (cit. on p. 64).
- [Cla+05] E. Clarke, A. Fehnker, S. K. Jha, and H. Veith. “Temporal Logic Model Checking”. In: *Handbook of Networked and Embedded Control Systems*. Boston, MA: Birkhäuser Boston, 2005, pp. 539–558. DOI: [10.1007/0-8176-4404-0_23](https://doi.org/10.1007/0-8176-4404-0_23) (cit. on p. 66).
- [Cla+20] R. Clarke, P. Kraikivski, B. C. Jones, C. M. Seignny, S. Sengupta, and Y. Wang. “A Systems Biology Approach to Discovering Pathway Signaling Dysregulation in Metastasis”. In: *Cancer and Metastasis Reviews* 39.3 (2020), pp. 903–918. DOI: [10.1007/s10555-020-09921-7](https://doi.org/10.1007/s10555-020-09921-7) (cit. on p. 1).
- [Coh+15] D. P. A. Cohen, L. Martignetti, S. Robine, E. Barillot, A. Zinovyev, and L. Calzone. “Mathematical Modelling of Molecular Pathways Enabling Tumour Cell Invasion and Migration”. In: *PLoS Computational Biology* 11.11 (2015), e1004571. DOI: [10.1371/journal.pcbi.1004571](https://doi.org/10.1371/journal.pcbi.1004571) (cit. on p. 49).
- [Col+17] S. Collombet, C. van Oevelen, J. L. S. Ortega, W. Abou-Jaoudé, B. Di Stefano, M. Thomas-Chollier, T. Graf, and D. Thieffry. “Logical Modeling of Lymphoid and Myeloid Cell Specification and Transdifferentiation”. In: *Proceedings of the National Academy of Sciences of the United States of America* 114.23 (2017), pp. 5792–5799. ISSN: 0027-8424 (cit. on p. 48).
- [CL03a] R. Cori and D. Lascar. *Logique mathématique 1 - Calcul propositionnel, algèbre de Boole, calcul des prédicats*. réédition (avec corrections) de l’édition Masson 1993. Vol. 1. 2 vols. Paris, France: Dunod, 2003. 408 pp. ISBN: 978-2-10-005452-7 (cit. on p. 14).

- [CL03b] R. Cori and D. Lascar. *Logique mathématique 2 - Fonctions récursives, théorème de Gödel, théorie des ensembles, théorie des modèles*. réédition (avec corrections) de l'édition Masson 1993. Vol. 2. 2 vols. Paris, France: Dunod, 2003. 368 pp. ISBN: 978-2-10-005453-4 (cit. on p. 14).
- [Cou+11] M. Courtot, N. Juty, C. Knüpfer, D. Waltemath, A. Zhukova, A. Dräger, M. Dumontier, A. Finney, M. Golebiewski, J. Hastings, S. Hoops, S. Keating, D. B. Kell, S. Kerrien, J. Lawson, A. Lister, J. Lu, R. Machne, P. Mendes, M. Pocock, N. Rodriguez, A. Villeger, D. J. Wilkinson, S. Wimalaratne, C. Laibe, M. Hucka, and Le Novère, Nicolas. "Controlled Vocabularies and Semantics in Systems Biology". In: *Molecular Systems Biology* 7.1 (2011), p. 543. DOI: [10.1038/msb.2011.77](https://doi.org/10.1038/msb.2011.77) (cit. on p. 137).
- [CC77] P. Cousot and R. Cousot. "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction Or Approximation of Fixpoints". In: *ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 1977, pp. 238–252. DOI: [10.1145/512950.512973](https://doi.org/10.1145/512950.512973) (cit. on p. 72).
- [CP08] G. Craciun and C. Pantea. "Identifiability of Chemical Reaction Networks". In: *Journal of Mathematical Chemistry* 44.1 (2008), pp. 244–259. DOI: [10.1007/s10910-007-9307-x](https://doi.org/10.1007/s10910-007-9307-x) (cit. on p. 99).
- [CH11] Y. Crama and P. L. Hammer. *Boolean Functions: Theory, Algorithms, and Applications*. 1st ed. Cambridge University Press, 2011. DOI: [10.1017/CBO9780511852008](https://doi.org/10.1017/CBO9780511852008) (cit. on p. 35).
- [Csi+06] A. Csikász-Nagy, D. Battogtokh, K. C. Chen, B. Novák, and J. J. Tyson. "Analysis of a Generic Model of Eukaryotic Cell-Cycle Regulation". In: *Biophysical Journal* 90.12 (2006), pp. 4361–4379. DOI: [10.1529/biophysj.106.081240](https://doi.org/10.1529/biophysj.106.081240) (cit. on p. 29).
- [CMC19] J. E. R. Cury, P. T. Monteiro, and C. Chaouiya. "Partial Order on the Set of Boolean Regulatory Functions". 2019. DOI: [10.48550/arXiv.1901.07623](https://doi.org/10.48550/arXiv.1901.07623). preprint (cit. on p. 36).
- [Dha06] P. D'haeseleer. "What Are DNA Sequence Motifs?" In: *Nature Biotechnology* 24.4 (2006), pp. 423–425. DOI: [10.1038/nbt0406-423](https://doi.org/10.1038/nbt0406-423) (cit. on p. 2).
- [Dai+10] B. J. Daigle, B. S. Srinivasan, J. A. Flannick, A. F. Novak, and S. Batzoglou. "Current Progress in Static and Dynamic Modeling of Biological Networks". In: *Systems Biology for Signaling Networks*. Red. by S. Choi. New York, NY: Springer New York, 2010, pp. 13–73. DOI: [10.1007/978-1-4419-5797-9_2](https://doi.org/10.1007/978-1-4419-5797-9_2) (cit. on p. 2).
- [Dan+07] V. Danos, J. Feret, W. Fontana, and J. Krivine. "Scalable Simulation of Cellular Signaling Networks". In: *Programming Languages and Systems*. Red. by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, and G. Weikum. Vol. 4807. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 139–157. DOI: [10.1007/978-3-540-76637-7_10](https://doi.org/10.1007/978-3-540-76637-7_10) (cit. on p. 53).
- [DN08] R. Darling and J. Norris. "Differential Equation Approximations for Markov Chains". In: *Probability Surveys* 5 (2008). DOI: [10.1214/07-PS121](https://doi.org/10.1214/07-PS121) (cit. on p. 72).
- [DB08a] M. Davidich and S. Bornholdt. "The Transition from Differential Equations to Boolean Networks: A Case Study in Simplifying a Regulatory Network Model". In: *Journal of Theoretical Biology* 255.3 (2008), pp. 269–277. DOI: [10.1016/j.jtbi.2008.07.020](https://doi.org/10.1016/j.jtbi.2008.07.020) (cit. on pp. 4, 72).
- [DB08b] M. I. Davidich and S. Bornholdt. "Boolean Network Model Predicts Cell Cycle Sequence of Fission Yeast". In: *PLoS ONE* 3.2 (2008), e1672. DOI: [10.1371/journal.pone.0001672](https://doi.org/10.1371/journal.pone.0001672) (cit. on pp. 48, 60).
- [DR06] H. De Jong and D. Ropers. "Strategies for Dealing with Incomplete Information in the Modeling of Molecular Interaction Networks". In: *Briefings in Bioinformatics* 7.4 (2006), pp. 354–363. DOI: [10.1093/bib/bb1034](https://doi.org/10.1093/bib/bb1034) (cit. on p. 68).
- [Den+15a] A. Dennunzio, E. Formenti, L. Manzoni, and A. E. Porreca. "Ancestors, Descendants, and Gardens of Eden in Reaction Systems". In: *Theoretical Computer Science* 608 (2015), pp. 16–26. DOI: [10.1016/j.tcs.2015.05.046](https://doi.org/10.1016/j.tcs.2015.05.046) (cit. on p. 72).
- [Den+15b] A. Dennunzio, E. Formenti, L. Manzoni, and A. E. Porreca. "Preimage Problems for Reaction Systems". In: *Language and Automata Theory and Applications*. Vol. 8977. Cham: Springer International Publishing, 2015, pp. 537–548. DOI: [10.1007/978-3-319-15579-1_42](https://doi.org/10.1007/978-3-319-15579-1_42) (cit. on p. 72).

- [Den+19] A. Dennunzio, E. Formenti, L. Manzoni, and A. E. Porreca. “Complexity of the Dynamics of Reaction Systems”. In: *Information and Computation* 267 (2019), pp. 96–109. DOI: [10.1016/j.ic.2019.03.006](https://doi.org/10.1016/j.ic.2019.03.006) (cit. on p. 72).
- [DR15] P. Deuffhard and S. Röblitz. “Numerical Simulation of ODE Models”. In: *A Guide to Numerical Modelling in Systems Biology*. Vol. 12. Cham: Springer International Publishing, 2015, pp. 33–87. DOI: [10.1007/978-3-319-20059-0_2](https://doi.org/10.1007/978-3-319-20059-0_2) (cit. on p. 29).
- [DD07] P. Dittrich and P. S. Di Fenizio. “Chemical Organisation Theory”. In: *Bulletin of Mathematical Biology* 69.4 (2007), pp. 1199–1231. DOI: [10.1007/s11538-006-9130-8](https://doi.org/10.1007/s11538-006-9130-8) (cit. on p. 67).
- [DW05] P. Dittrich and L. Winter. “Reaction Networks as a Formal Mechanism to Explain Social Phenomena”. In: *International Workshop on Agent-based Approaches in Economics and Social Complex Systems*. 2005, p. 15 (cit. on p. 49).
- [Don+07] G. Q. Dong, L. Jakobowski, M. A. J. Iafolla, and D. R. McMillen. “Simplification of Stochastic Chemical Reaction Models with Fast and Slow Dynamics”. In: *Journal of Biological Physics* 33.1 (2007), pp. 67–95. DOI: [10.1007/s10867-007-9043-2](https://doi.org/10.1007/s10867-007-9043-2) (cit. on p. 69).
- [Dor+16] J. Dorier, I. Crespo, A. Niknejad, R. Liechti, M. Ebeling, and I. Xenarios. “Boolean Regulatory Network Reconstruction Using Literature Based Knowledge with a Genetic Algorithm Optimization Method”. In: *BMC Bioinformatics* 17.1 (2016), p. 410. DOI: [10.1186/s12859-016-1287-z](https://doi.org/10.1186/s12859-016-1287-z) (cit. on p. 119).
- [EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer Monographs in Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995. DOI: [10.1007/3-540-28788-4](https://doi.org/10.1007/3-540-28788-4) (cit. on p. 23).
- [ER05] A. Ehrenfeucht and G. Rozenberg. “Basic Notions of Reaction Systems”. In: *Developments in Language Theory*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 27–29. DOI: [10.1007/978-3-540-30550-7_3](https://doi.org/10.1007/978-3-540-30550-7_3) (cit. on p. 51).
- [EMR11] A. Ehrenfeucht, M. Main, and G. Rozenberg. “Functions Defined by Reaction Systems”. In: *International Journal of Foundations of Computer Science* 22.01 (2011), pp. 167–178. DOI: [10.1142/S0129054111007927](https://doi.org/10.1142/S0129054111007927) (cit. on p. 67).
- [EMR10] A. Ehrenfeucht, M. G. Main, and G. Rozenberg. “Combinatorics of Life and Death for Reaction Systems”. In: *International Journal of Foundations of Computer Science* 21.3 (2010), pp. 345–356. DOI: [10.1142/S0129054110007295](https://doi.org/10.1142/S0129054110007295) (cit. on p. 68).
- [FGS12] F. Fages, S. Gay, and S. Soliman. *Automatic Curation of SBML Models Based on Their ODE Semantics*. Research Report RR-8014. INRIA, 2012. URL: <https://hal.inria.fr/hal-00723554> (cit. on pp. 45, 137, 139, 146).
- [FGS15] F. Fages, S. Gay, and S. Soliman. “Inferring Reaction Systems from Ordinary Differential Equations”. In: *Theoretical Computer Science* 599 (2015), pp. 64–78. DOI: [10.1016/j.tcs.2014.07.032](https://doi.org/10.1016/j.tcs.2014.07.032) (cit. on p. 99).
- [Fag+16] F. Fages, T. Martinez, D. A. Rosenblueth, and S. Soliman. “Influence Systems vs Reaction Systems”. In: *Computational Methods in Systems Biology*. Vol. 9859. Cham: Springer International Publishing, 2016, pp. 98–115. DOI: [10.1007/978-3-319-45177-0_7](https://doi.org/10.1007/978-3-319-45177-0_7) (cit. on pp. 2, 71).
- [FS08a] F. Fages and S. Soliman. “Abstract Interpretation and Types for Systems Biology”. In: *Theoretical Computer Science* 403.1 (2008), pp. 52–70. DOI: [10.1016/j.tcs.2008.04.024](https://doi.org/10.1016/j.tcs.2008.04.024) (cit. on pp. 4, 71, 72, 97).
- [FS08b] F. Fages and S. Soliman. “From Reaction Models to Influence Graphs and Back: A Theorem”. In: *Formal Methods in Systems Biology*. Vol. 5054. Lecture Notes in Computer Science. Springer, 2008, pp. 90–102. DOI: [10.1007/978-3-540-68413-8_7](https://doi.org/10.1007/978-3-540-68413-8_7) (cit. on p. 77).
- [Far21] K. D. Farnsworth. “An Organisational Systems-Biology View of Viruses Explains Why They Are Not Alive”. In: *Biosystems* 200 (2021), p. 104324. DOI: [10.1016/j.biosystems.2020.104324](https://doi.org/10.1016/j.biosystems.2020.104324) (cit. on p. 1).

- [Fei+09] A. M. Feist, M. J. Herrgård, I. Thiele, J. L. Reed, and B. Ø. Palsson. “Reconstruction of Biochemical Networks in Microorganisms”. In: *Nature Reviews Microbiology* 7.2 (2009), pp. 129–143. DOI: [10.1038/nrmicro1949](https://doi.org/10.1038/nrmicro1949) (cit. on p. 60).
- [Fer+19] C. Ferretti, A. Leporati, L. Manzoni, and A. E. Porreca. “The Many Roads to the Simulation of Reaction Systems”. In: *Fundamenta Informaticae* 171.1-4 (2019), pp. 175–188. DOI: [10.3233/FI-2020-1878](https://doi.org/10.3233/FI-2020-1878) (cit. on p. 67).
- [FH07] J. Fisher and T. A. Henzinger. “Executable Cell Biology”. In: *Nature Biotechnology* 25.11 (2007), pp. 1239–1249. DOI: [10.1038/nbt1356](https://doi.org/10.1038/nbt1356) (cit. on p. 2).
- [FMP14] E. Formenti, L. Manzoni, and A. E. Porreca. “Fixed Points and Attractors of Reaction Systems”. In: *Language, Life, Limits*. Red. by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, A. Kobsa, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, D. Terzopoulos, D. Tygar, and G. Weikum. Vol. 8493. Cham: Springer International Publishing, 2014, pp. 194–203. DOI: [10.1007/978-3-319-08019-2_20](https://doi.org/10.1007/978-3-319-08019-2_20) (cit. on p. 72).
- [Fuj+17] M. Fujii, K. Ohashi, Y. Karasawa, M. Hikichi, and S. Kuroda. “Small-Volume Effect Enables Robust, Sensitive, and Efficient Information Transfer in the Spine”. In: *Biophysical Journal* 112.4 (2017), pp. 813–826. DOI: [10.1016/j.bpj.2016.12.043](https://doi.org/10.1016/j.bpj.2016.12.043) (cit. on p. 98).
- [GSA18] J. G.T. Zañudo, S. N. Steinway, and R. Albert. “Discrete Dynamic Network Modeling of Oncogenic Signaling: Mechanistic Insights for Personalized Treatment of Cancer”. In: *Current Opinion in Systems Biology* 9 (2018), pp. 1–10. DOI: [10.1016/j.coisb.2018.02.002](https://doi.org/10.1016/j.coisb.2018.02.002) (cit. on p. 49).
- [GN00] E. Gansner and S. North. “An Open Graph Visualization System and Its Applications to Software Engineering”. In: *Software—Practice & Experience* 30.11 (2000), pp. 1203–1233. URL: <https://dl.acm.org/doi/abs/10.5555/358668.358697> (cit. on p. 141).
- [Gao+22] S. Gao, C. Sun, C. Xiang, K. Qin, and T. H. Lee. “Learning Asynchronous Boolean Networks From Single-Cell Data Using Multiobjective Cooperative Genetic Programming”. In: *IEEE Transactions on Cybernetics* 52.5 (2022), pp. 2916–2930. DOI: [10.1109/TCYB.2020.3022430](https://doi.org/10.1109/TCYB.2020.3022430) (cit. on p. 66).
- [Gar+19] L. Garcia-Alonso, C. H. Holland, M. M. Ibrahim, D. Turei, and J. Saez-Rodriguez. “Benchmark and Integration of Resources for the Estimation of Human Transcription Factor Activities”. In: *Genome Research* 29.8 (2019), pp. 1363–1375. DOI: [10.1101/gr.240663.118](https://doi.org/10.1101/gr.240663.118) (cit. on p. 60).
- [Geb+12] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012. DOI: [10.2200/S00457ED1V01Y201211AIM019](https://doi.org/10.2200/S00457ED1V01Y201211AIM019) (cit. on p. 141).
- [Gil76] D. T. Gillespie. “A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions”. In: *Journal of Computational Physics* 22.4 (1976), pp. 403–434. DOI: [10.1016/0021-9991\(76\)90041-3](https://doi.org/10.1016/0021-9991(76)90041-3) (cit. on p. 55).
- [Gil+22] M. Gillespie, B. Jassal, R. Stephan, M. Milacic, K. Rothfels, A. Senff-Ribeiro, J. Griss, C. Sevilla, L. Matthews, C. Gong, C. Deng, T. Varusai, E. Ragueneau, Y. Haider, B. May, V. Shamovsky, J. Weiser, T. Brunson, N. Sanati, L. Beckman, X. Shao, A. Fabregat, K. Sidiropoulos, J. Murillo, G. Viteri, J. Cook, S. Shorser, G. Bader, E. Demir, C. Sander, R. Haw, G. Wu, L. Stein, H. Hermjakob, and P. D’Eustachio. “The Reactome Pathway Knowledgebase 2022”. In: *Nucleic Acids Research* 50.D1 (2022), pp. D687–D692. DOI: [10.1093/nar/gkab1028](https://doi.org/10.1093/nar/gkab1028) (cit. on p. 60).
- [GK73] L. Glass and S. A. Kauffman. “The Logical Analysis of Continuous, Non-Linear Biochemical Control Networks”. In: *Journal of Theoretical Biology* 39.1 (1973), pp. 103–129. DOI: [10.1016/0022-5193\(73\)90208-7](https://doi.org/10.1016/0022-5193(73)90208-7) (cit. on p. 97).
- [Goo12] W. Goodwin. “Mechanisms and Chemical Reaction”. In: *Philosophy of Chemistry* (2012), pp. 309–327. DOI: [10.1016/B978-0-444-51675-6.50023-2](https://doi.org/10.1016/B978-0-444-51675-6.50023-2) (cit. on p. 2).
- [GW82] B. A. Gottwald and G. Wanner. “Comparison of Numerical Methods for Stiff Differential Equations in Biology and Chemistry”. In: *Simulation* 38.2 (1982), pp. 61–66. DOI: [10.1177/003754978203800206](https://doi.org/10.1177/003754978203800206) (cit. on p. 89).

- [Gui+13] G. Guillén-Gosálbez, A. Miró, R. Alves, A. Sorribas, and L. Jiménez. “Identification of Regulatory Structure and Kinetic Parameters of Biochemical Networks via Mixed-Integer Dynamic Optimization”. In: *BMC Systems Biology* 7.1 (2013), p. 113. doi: [10.1186/1752-0509-7-113](https://doi.org/10.1186/1752-0509-7-113) (cit. on p. 64).
- [Gun14a] J. Gunawardena. “Beware the Tail That Wags the Dog: Informal and Formal Models in Biology”. In: *Molecular Biology of the Cell* 25.22 (2014), pp. 3441–3444. doi: [10.1091/mbc.e14-02-0717](https://doi.org/10.1091/mbc.e14-02-0717) (cit. on p. 2).
- [Gun14b] J. Gunawardena. “Models in Biology: ‘Accurate Descriptions of Our Pathetic Thinking’”. In: *BMC Biology* 12.1 (2014), p. 29. doi: [10.1186/1741-7007-12-29](https://doi.org/10.1186/1741-7007-12-29) (cit. on p. 2).
- [Gun14c] J. Gunawardena. “Time-Scale Separation - Michaelis and Menten’s Old Idea, Still Bearing Fruit”. In: *FEBS Journal* 281.2 (2014), pp. 473–488. doi: [10.1111/febs.12532](https://doi.org/10.1111/febs.12532) (cit. on p. 69).
- [Han+14] S. Han, R. K. W. Wong, T. C. M. Lee, L. Shen, S.-Y. R. Li, and X. Fan. “A Full Bayesian Approach for Boolean Genetic Network Inference”. In: *PLoS ONE* 9.12 (2014), e115806. doi: [10.1371/journal.pone.0115806](https://doi.org/10.1371/journal.pone.0115806) (cit. on pp. 63, 67).
- [HT79] V. Hars and J. Tóth. “On the Inverse Problem of Reaction Kinetics”. In: *Qualitative Theory of Differential Equations* 30 (1979) (cit. on p. 99).
- [HGD08] M. Heiner, D. Gilbert, and R. Donaldson. “Petri Nets for Systems and Synthetic Biology”. In: *Formal Methods for Computational Systems Biology*. Vol. 5016. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 215–264. doi: [10.1007/978-3-540-68894-5_7](https://doi.org/10.1007/978-3-540-68894-5_7) (cit. on p. 4).
- [Hem+22] A. A. Hemedan, A. Niarakis, R. Schneider, and M. Ostaszewski. “Boolean Modelling as a Logic-Based Dynamic Approach in Systems Medicine”. In: *Computational and Structural Biotechnology Journal* 20 (2022), pp. 3161–3172. doi: [10.1016/j.csbj.2022.06.035](https://doi.org/10.1016/j.csbj.2022.06.035) (cit. on p. 58).
- [Hil15] R. A. Hillmer. “Systems Biology for Biologists”. In: *PLoS Pathogens* 11.5 (2015), e1004786. doi: [10.1371/journal.ppat.1004786](https://doi.org/10.1371/journal.ppat.1004786) (cit. on p. 1).
- [Hir+21] A. Hirtz, N. Lebourdais, F. Rech, Y. Bailly, A. Vaginay, M. Smaïl-Tabbone, H. Dubois-Pot-Schneider, and H. Dumond. “GPER Agonist G-1 Disrupts Tubulin Dynamics and Potentiates Temozolomide to Impair Glioblastoma Cell Proliferation”. In: *Cells* 10.12 (2021), p. 3438. doi: [10.3390/cells10123438](https://doi.org/10.3390/cells10123438) (cit. on p. 8).
- [Hoo+06] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. “COPASI—a COMplex Pathway SIMulator”. In: *Bioinformatics* 22.24 (2006), pp. 3067–3074. doi: [10.1093/bioinformatics/btl485](https://doi.org/10.1093/bioinformatics/btl485) (cit. on pp. 102, 139).
- [Hua+03] E. Huang, S. H. Cheng, H. Dressman, J. Pittman, M. H. Tsou, C. F. Horng, A. Bild, E. S. Iversen, M. Liao, C. M. Chen, M. West, J. R. Nevins, and A. T. Huang. “Gene Expression Predictors of Breast Cancer Outcomes”. In: *The Lancet* 361.9369 (2003), pp. 1590–1596. doi: [10.1016/S0140-6736\(03\)13308-9](https://doi.org/10.1016/S0140-6736(03)13308-9) (cit. on p. 60).
- [Hun+08a] C. A. Hunt, G. E. P. Ropella, S. Park, and J. Engelberg. “Dichotomies between Computational and Mathematical Models”. In: *Nature Biotechnology* 26.7 (2008), pp. 737–738. doi: [10.1038/nbt0708-737](https://doi.org/10.1038/nbt0708-737) (cit. on p. 2).
- [Hun+08b] C. A. Hunt, G. E. P. Ropella, S. Park, and J. Engelberg. “Reply to Dichotomies between Computational and Mathematical Models”. In: *Nature Biotechnology* 26.7 (7 2008), pp. 738–739. doi: [10.1038/nbt0708-738](https://doi.org/10.1038/nbt0708-738) (cit. on p. 2).
- [HN60] O. F. Hutter and D. Noble. “Rectifying Properties of Heart Muscle”. In: *Nature* 188.4749 (1960), pp. 495–495. doi: [10.1038/188495a0](https://doi.org/10.1038/188495a0) (cit. on p. 49).
- [ISB15] ISBE Infrastructure for Systems Biology Europe. *Success Stories in Systems Biology - Case Studies*. 2015. URL: <https://zenodo.org/record/21393> (cit. on p. 49).
- [Jäk23] C. Jäkel. “A Computation of the Ninth Dedekind Number”. 2023. doi: [10.48550/arXiv.2304.00895](https://doi.org/10.48550/arXiv.2304.00895). preprint (cit. on p. 36).

- [JSB12] S. Jamshidi, H. Siebert, and A. Bockmayr. “Comparing Discrete and Piecewise Affine Differential Equation Models of Gene Regulatory Networks”. In: *Information Processign in Cells and Tissues*. Vol. 7223. Lecture Notes in Computer Science. Springer, 2012, pp. 17–24. DOI: [10.1007/978-3-642-28792-3_3](#) (cit. on p. 73).
- [JSB13] S. Jamshidi, H. Siebert, and A. Bockmayr. “Preservation of Dynamic Properties in Qualitative Modeling Frameworks for Gene Regulatory Networks”. In: *Biosystems* 112.2 (2013), pp. 171–179. DOI: [10.1016/j.biosystems.2013.03.001](#) (cit. on p. 97).
- [Jia+22] R. Jiang, P. Singh, F. Wrede, A. Hellander, and L. Petzold. “Identification of Dynamic Mass-Action Biochemical Reaction Networks Using Sparse Bayesian Methods”. In: *PLoS Computational Biology* 18.1 (2022), e1009830. DOI: [10.1371/journal.pcbi.1009830](#) (cit. on p. 67).
- [JNN13] M. John, M. Nebut, and J. Niehren. “Knockout Prediction for Reaction Networks with Partial Kinetic Information”. In: *Verification, Model Checking, and Abstract Interpretation*. Red. by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, and G. Weikum. Vol. 7737. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 355–374. DOI: [10.1007/978-3-642-35873-9_22](#) (cit. on p. 23).
- [KRD09] C. Kaleta, S. Richter, and P. Dittrich. “Using Chemical Organization Theory for Model Checking”. In: *Bioinformatics* 25.15 (2009), pp. 1915–1922. DOI: [10.1093/bioinformatics/btp332](#) (cit. on p. 67).
- [Kan00] M. Kanehisa. “KEGG: Kyoto Encyclopedia of Genes and Genomes”. In: *Nucleic Acids Research* 28.1 (2000), pp. 27–30. DOI: [10.1093/nar/28.1.27](#) (cit. on p. 60).
- [Kar+12] J. R. Karr, J. C. Sanghvi, D. N. Macklin, M. V. Gutschow, J. M. Jacobs, B. Bolival, N. Assad-Garcia, J. I. Glass, and M. W. Covert. “A Whole-Cell Computational Model Predicts Phenotype from Genotype”. In: *Cell* 150.2 (2012), pp. 389–401. DOI: [10.1016/j.cell.2012.05.044](#) (cit. on pp. 49, 60, 71).
- [Kau69] S. Kauffman. “Metabolic Stability and Epigenesis in Randomly Constructed Genetic Nets”. In: *Journal of Theoretical Biology* 22.3 (1969), pp. 437–467. DOI: [10.1016/0022-5193\(69\)90015-0](#) (cit. on pp. 42, 57).
- [Kau+03] S. Kauffman, C. Peterson, B. Samuelsson, and C. Troein. “Random Boolean Network Models and the Yeast Transcriptional Network”. In: *Proceedings of the National Academy of Sciences* 100.25 (2003), pp. 14796–14799. DOI: [10.1073/pnas.2036429100](#) (cit. on p. 48).
- [Kea+20] S. M. Keating et al. “SBML Level 3: An Extensible Format for the Exchange and Reuse of Biological Models”. In: *Molecular Systems Biology* 16.8 (2020). DOI: [10.15252/msb.20199110](#) (cit. on pp. 3, 137).
- [Kha21] A. Khare. “Experimental Systems Biology Approaches Reveal Interaction Mechanisms in Model Multispecies Communities”. In: *Trends in Microbiology* 29.12 (2021), pp. 1083–1094. DOI: [10.1016/j.tim.2021.03.012](#) (cit. on p. 1).
- [KJB14] J. K. Kim, K. Josić, and M. R. Bennett. “The Validity of Quasi-Steady-State Approximations in Discrete Stochastic Simulations”. In: *Biophysical Journal* 107.3 (2014), pp. 783–793. DOI: [10.1016/j.bpj.2014.06.012](#) (cit. on p. 69).
- [KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. “Optimization by Simulated Annealing”. In: *Science* 220.4598 (1983), pp. 671–680. DOI: [10.1126/science.220.4598.671](#) (cit. on p. 66).
- [Kit02] H. Kitano. “Systems Biology: A Brief Overview”. In: *Science* 295.5560 (2002), pp. 1662–1664. DOI: [10.1126/science.1069492](#) (cit. on p. 3).
- [Kit04] H. Kitano. “Biological Robustness”. In: *Nature Reviews Genetics* 5.11 (2004), pp. 826–837. DOI: [10.1038/nrg1471](#) (cit. on p. 1).
- [KSS16] H. Klarner, A. Streck, and H. Siebert. “PyBoolNet: A Python Package for the Generation, Analysis and Visualization of Boolean Networks”. In: *Bioinformatics* (2016), btw682. DOI: [10.1093/bioinformatics/btw682](#) (cit. on pp. 127, 129, 138, 141).

- [Knu64] D. E. Knuth. “Backus Normal Form vs. Backus Naur Form”. In: *Communications of the ACM* 7.12 (1964), pp. 735–736. DOI: [10.1145/355588.365140](https://doi.org/10.1145/355588.365140) (cit. on p. 14).
- [Koh+10] P. Kohl, E. J. Crampin, T. A. Quinn, and D. Noble. “Systems Biology: An Approach”. In: *Clinical Pharmacology & Therapeutics* 88.1 (2010), pp. 25–33. DOI: [10.1038/clpt.2010.92](https://doi.org/10.1038/clpt.2010.92) (cit. on pp. 2, 99).
- [Kru+11] J. Krumsiek, C. Marr, T. Schroeder, and F. J. Theis. “Hierarchical Differentiation of Myeloid Progenitors Is Encoded in the Transcription Factor Network”. In: *PLoS ONE* 6.8 (2011), e22649. DOI: [10.1371/journal.pone.0022649](https://doi.org/10.1371/journal.pone.0022649) (cit. on p. 60).
- [Kru+10] J. Krumsiek, S. Pölsterl, D. M. Wittmann, and F. J. Theis. “Odefy - From Discrete to Continuous Models”. In: *BMC Bioinformatics* 11.1 (2010), p. 233. DOI: [10.1186/1471-2105-11-233](https://doi.org/10.1186/1471-2105-11-233) (cit. on p. 71).
- [Kur70] T. G. Kurtz. “Solutions of Ordinary Differential Equations as Limits of Pure Jump Markov Processes”. In: *Journal of Applied Probability* 7.1 (1970), pp. 49–58. DOI: [10.2307/3212147](https://doi.org/10.2307/3212147) (cit. on pp. 72, 98).
- [LA87] P. J. M. Van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Dordrecht: Springer Netherlands, 1987. DOI: [10.1007/978-94-015-7744-1](https://doi.org/10.1007/978-94-015-7744-1) (cit. on p. 66).
- [Läh+06] H. Lähdesmäki, S. Hautaniemi, I. Shmulevich, and O. Yli-Harja. “Relationships between Probabilistic Boolean Networks and Dynamic Bayesian Networks as Models of Gene Regulatory Networks”. In: *Signal processing* 86.4 (2006), pp. 814–834. DOI: [10.1016/j.sigpro.2005.06.008](https://doi.org/10.1016/j.sigpro.2005.06.008) (cit. on p. 71).
- [LSY03] H. Lähdesmäki, I. Shmulevich, and O. Yli-Harja. “On Learning Gene Regulatory Networks under the Boolean Network Model”. In: *Machine Learning* 52.1 (2003), pp. 147–167. DOI: [10.1023/A:1023905711304](https://doi.org/10.1023/A:1023905711304) (cit. on pp. 63, 119, 122, 126).
- [Lav89] A.-L. Lavoisier. *Traité élémentaire de chimie*. 1789. URL: <https://gallica.bnf.fr/ark:/12148/btv1b8615746s> (cit. on p. 1).
- [Laz02] Y. Lazebnik. “Can a Biologist Fix a Radio?—Or, What I Learned While Studying Apoptosis”. In: *Cancer Cell* 2.3 (2002), pp. 179–182. DOI: [10.1016/S1535-6108\(02\)00133-2](https://doi.org/10.1016/S1535-6108(02)00133-2) (cit. on p. 1).
- [LZ05] E. A. Lee and H. Zheng. “Operational Semantics of Hybrid Systems”. In: *Hybrid Systems: Computation and Control*. Red. by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, and G. Weikum. Vol. 3414. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 25–53. DOI: [10.1007/978-3-540-31954-2_2](https://doi.org/10.1007/978-3-540-31954-2_2) (cit. on p. 55).
- [LA21] A. Lezaja and M. Altmeyer. “Dealing with DNA Lesions: When One Cell Cycle Is Not Enough”. In: *Current Opinion in Cell Biology* 70 (2021), pp. 27–36. DOI: [10.1016/j.ceb.2020.11.001](https://doi.org/10.1016/j.ceb.2020.11.001) (cit. on p. 1).
- [Li+04] F. Li, T. Long, Y. Lu, Q. Ouyang, and C. Tang. “The Yeast Cell-Cycle Network Is Robustly Designed”. In: *Proceedings of the National Academy of Sciences* 101.14 (2004), pp. 4781–4786. DOI: [10.1073/pnas.0305937101](https://doi.org/10.1073/pnas.0305937101) (cit. on pp. 1, 62).
- [Li+07] P. Li, C. Zhang, E. J. Perkins, P. Gong, and Y. Deng. “Comparison of Probabilistic Boolean Network and Dynamic Bayesian Network Approaches for Inferring Gene Regulatory Networks”. In: *BMC Bioinformatics* 8.S7 (2007), S13. DOI: [10.1186/1471-2105-8-S7-S13](https://doi.org/10.1186/1471-2105-8-S7-S13) (cit. on p. 71).
- [LH12] J. Liang and J. Han. “Stochastic Boolean Networks: An Efficient Approach to Modeling Gene Regulatory Networks”. In: *BMC Systems Biology* 6.1 (2012), p. 113. DOI: [10.1186/1752-0509-6-113](https://doi.org/10.1186/1752-0509-6-113) (cit. on p. 98).
- [LFS98] S. Liang, S. Fuhrman, and R. Somogyi. “REVEAL, a General Reverse Engineering Algorithm for Inference of Genetic Network Architectures”. In: *Pacific Symposium on Biocomputing*. (1998), pp. 18–29. ISSN: 2335-6928 (cit. on pp. 63, 64, 119, 122, 126).
- [Lis11] A. L. Lister. “Enhancing Systems Biology Models through Semantic Data Integration”. PhD thesis. Newcastle University, 2011, p. 220 (cit. on p. 3).

- [Liu+21] X. Liu, Y. Wang, N. Shi, Z. Ji, and S. He. “GAPORE: Boolean Network Inference Using a Genetic Algorithm with Novel Polynomial Representation and Encoding Scheme”. In: *Knowledge-Based Systems* 228 (2021), p. 107277. DOI: [10.1016/j.knosys.2021.107277](https://doi.org/10.1016/j.knosys.2021.107277) (cit. on p. 66).
- [LMY12] Z. Liu, B. Malone, and C. Yuan. “Empirical Evaluation of Scoring Functions for Bayesian Network Model Selection”. In: *BMC Bioinformatics* 13.S15 (2012), S14. DOI: [10.1186/1471-2105-13-S15-S14](https://doi.org/10.1186/1471-2105-13-S15-S14) (cit. on p. 64).
- [Loc+06] J. C. W. Locke, L. Kozma-Bognár, P. D. Gould, B. Fehér, É. Kevei, F. Nagy, M. S. Turner, A. Hall, and A. J. Millar. “Experimental Validation of a Predicted Feedback Loop in the Multi-oscillator Clock of *Arabidopsis Thaliana*”. In: *Molecular Systems Biology* 2.1 (2006), p. 59. DOI: [10.1038/msb4100102](https://doi.org/10.1038/msb4100102) (cit. on pp. 126, 127).
- [LAM19] P. Loskot, K. Atitey, and L. Mihaylova. “Comprehensive Review of Models and Methods for Inferences in Bio-Chemical Reaction Networks”. In: *Frontiers in Genetics* 10 (2019). DOI: [10.3389/fgene.2019.00549](https://doi.org/10.3389/fgene.2019.00549) (cit. on p. 58).
- [Lot25] A. Lotka. “Elements of Physical Biology”. In: *Nature* 116.2917 (1917 1925), pp. 461–461. DOI: [10.1038/116461b0](https://doi.org/10.1038/116461b0) (cit. on p. 49).
- [Mac+11] D. Machado, R. S. Costa, M. Rocha, E. C. Ferreira, B. Tidor, and I. Rocha. “Modeling Formalisms in Systems Biology”. In: *AMB Express* 1.1 (2011), p. 45. DOI: [10.1186/2191-0855-1-45](https://doi.org/10.1186/2191-0855-1-45) (cit. on p. 2).
- [Mac03] D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003. 694 pp. ISBN: 978-0-521-64298-9 (cit. on p. 64).
- [Mad+16] G. Madelaine, C. Lhoussaine, J. Niehren, and E. Tonello. “Structural Simplification of Chemical Reaction Networks in Partial Steady States”. In: *Biosystems* 149 (2016), pp. 34–49. DOI: [10.1016/j.biosystems.2016.08.003](https://doi.org/10.1016/j.biosystems.2016.08.003) (cit. on p. 68).
- [MG20] J. C. Magee and C. Grienberger. “Synaptic Plasticity Forms and Functions”. In: *Annual Review of Neuroscience* 43.1 (2020), pp. 95–117. DOI: [10.1146/annurev-neuro-090919-022842](https://doi.org/10.1146/annurev-neuro-090919-022842) (cit. on p. 98).
- [Mal+20] R. S. Malik-Sheriff, M. Glont, T. V. N. Nguyen, K. Tiwari, M. G. Roberts, A. Xavier, M. T. Vu, J. Men, M. Maire, S. Kananathan, E. L. Fairbanks, J. P. Meyer, C. Arankalle, T. M. Varusai, V. Knight-Schrijver, L. Li, C. Dueñas-Roca, G. Dass, S. M. Keating, Y. M. Park, N. Buso, N. Rodriguez, M. Hucka, and H. Hermjakob. “BioModels—15 Years of Sharing Computational Models in Life Science”. In: *Nucleic Acids Research* 48.D1 (2020), pp. D407–D415. DOI: [10.1093/nar/gkz1055](https://doi.org/10.1093/nar/gkz1055) (cit. on pp. 3, 49, 144).
- [Mar+06] A. A. Margolin, I. Nemenman, K. Basso, C. Wiggins, G. Stolovitzky, R. D. Favera, and A. Califano. “ARACNE: An Algorithm for the Reconstruction of Gene Regulatory Networks in a Mammalian Cellular Context”. In: *BMC Bioinformatics* 7.S1 (2006), S7. DOI: [10.1186/1471-2105-7-S1-S7](https://doi.org/10.1186/1471-2105-7-S1-S7) (cit. on p. 64).
- [Mar82] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Cambridge, Mass: MIT Press, 1982. 403 pp. ISBN: 978-0-262-51462-0 (cit. on p. 1).
- [Mar+21] M. Martens, A. Ammar, A. Riutta, A. Waagmeester, D. N. Slenter, K. Hanspers, R. A. Miller, D. Digles, E. N. Lopes, F. Ehrhart, L. J. Dupuis, L. A. Winckers, S. L. Coort, E. L. Willighagen, C. T. Evelo, A. R. Pico, and M. Kutmon. “WikiPathways: Connecting Communities”. In: *Nucleic Acids Research* 49.D1 (2021), pp. D613–D621. DOI: [10.1093/nar/gkaa1024](https://doi.org/10.1093/nar/gkaa1024) (cit. on p. 60).
- [MDW04] S. Martin, G. Davidson, and M. Werner-Washburne. “Inferring Genetic Networks from Microarray Data”. In: *IEEE Computational Systems Bioinformatics Conference*. Stanford, CA, USA: IEEE, 2004, pp. 539–542. DOI: [10.1109/CSB.2004.1332498](https://doi.org/10.1109/CSB.2004.1332498) (cit. on pp. 60, 63, 122, 125).
- [Mar+07] S. Martin, Z. Zhang, A. Martino, and J.-L. Faulon. “Boolean Dynamics of Genetic Regulatory Networks Inferred from Microarray Time Series Data”. In: *Bioinformatics* 23.7 (2007), pp. 866–874. DOI: [10.1093/bioinformatics/btm021](https://doi.org/10.1093/bioinformatics/btm021) (cit. on pp. 60, 63, 122, 124).

- [Mau+11] M. Maucher, B. Kracher, M. Köhl, and H. A. Kestler. “Inferring Boolean Network Structure via Correlation”. In: *Bioinformatics* 27.11 (2011), pp. 1529–1536. DOI: [10.1093/bioinformatics/btr166](https://doi.org/10.1093/bioinformatics/btr166) (cit. on p. 65).
- [McC56] E. J. McCluskey. “Minimization of Boolean Functions*”. In: *Bell System Technical Journal* 35.6 (1956), pp. 1417–1444. DOI: [10.1002/j.1538-7305.1956.tb03835.x](https://doi.org/10.1002/j.1538-7305.1956.tb03835.x) (cit. on p. 35).
- [Mel13] T. Melham. “Modelling, Abstraction, and Computation in Systems Biology: A View from Computer Science”. In: *Progress in Biophysics and Molecular Biology* 111.2-3 (2013), pp. 129–136. DOI: [10.1016/j.pbiomolbio.2012.08.015](https://doi.org/10.1016/j.pbiomolbio.2012.08.015) (cit. on p. 4).
- [Mel+16] T. Melliti, D. Regnault, A. Richard, and S. Sené. “Asynchronous Simulation of Boolean Networks by Monotone Boolean Networks”. In: *International Conference on Cellular Automata for Research and Industry*. Vol. 9863. Lecture Notes in Computer Science. Springer, 2016, pp. 182–191. DOI: [10.1007/978-3-319-44365-2_18](https://doi.org/10.1007/978-3-319-44365-2_18) (cit. on p. 57).
- [MTA99] L. Mendoza, D. Thieffry, and E. R. Alvarez-Buylla. “Genetic Control of Flower Morphogenesis in Arabidopsis Thaliana: A Logical Analysis.” In: *Bioinformatics* 15.7 (1999), pp. 593–606. DOI: [10.1093/bioinformatics/15.7.593](https://doi.org/10.1093/bioinformatics/15.7.593) (cit. on p. 49).
- [Men06] L. Mendoza. “A Network Model for the Control of the Differentiation Process in Th Cells”. In: *Biosystems* 84.2 (2006), pp. 101–114. DOI: [10.1016/j.biosystems.2005.10.004](https://doi.org/10.1016/j.biosystems.2005.10.004) (cit. on p. 48).
- [MÁ98] L. Menzoda and E. R. Álvarez-Buylla. “Dynamics of the Genetic Regulatory Network for Arabidopsis Thaliana Flower Morphogenesis”. In: *Journal of Theoretical Biology* (1998). DOI: [10.1006/jtbi.1998.0701](https://doi.org/10.1006/jtbi.1998.0701) (cit. on p. 49).
- [MM13a] L. Michaelis and M. M. L. Menten. “The Kinetics of Invertin Action: Translated by T.R.C. Boyde Submitted 4 February 1913”. In: *FEBS Letters* 587.17 (2013), pp. 2712–2720. DOI: [10.1016/j.febslet.2013.07.015](https://doi.org/10.1016/j.febslet.2013.07.015) (cit. on p. 70).
- [MM13b] L. Michaelis and M. L. Menten. “Die Kinetik Der Invertinwirkung”. In: *Biochemische Zeitschrift* 49.333-369 (1913), p. 352 (cit. on p. 70).
- [Mic+11] L. Michaelis, M. L. Menten, K. A. Johnson, and R. S. Goody. “The Original Michaelis Constant: Translation of the 1913 Michaelis-Menten Paper”. In: *Biochemistry* 50.39 (2011), pp. 8264–8269. DOI: [10.1021/bi201284u](https://doi.org/10.1021/bi201284u) (cit. on p. 70).
- [Möl+21] F. Mölder, K. P. Jablonski, B. Letcher, M. B. Hall, C. H. Tomkins-Tinch, V. Sochat, J. Forster, S. Lee, S. O. Twardziok, A. Kanitz, A. Wilm, M. Holtgrewe, S. Rahmann, S. Nahnsen, and J. Köster. “Sustainable Data Analysis with Snakemake”. In: *F1000Research* 10 (2021), p. 33. DOI: [10.12688/f1000research.29032.2](https://doi.org/10.12688/f1000research.29032.2) (cit. on p. 138).
- [Mov+21] S. Mover, A. Cimatti, A. Griggio, A. Irfan, and S. Tonetta. “Implicit Semi-Algebraic Abstraction for Polynomial Dynamical Systems”. In: *Computer Aided Verification*. Vol. 12759. Cham: Springer International Publishing, 2021, pp. 529–551. DOI: [10.1007/978-3-030-81685-8_25](https://doi.org/10.1007/978-3-030-81685-8_25) (cit. on p. 96).
- [MFS22] S. Müller, C. Flamm, and P. F. Stadler. “What Makes a Reaction Network “Chemical”?” In: *Journal of Cheminformatics* 14.1 (2022), p. 63. DOI: [10.1186/s13321-022-00621-8](https://doi.org/10.1186/s13321-022-00621-8) (cit. on p. 49).
- [Mun+09] P. Munz, I. Hudea, J. Imad, and R. J. Smith. “When Zombies Attack!: Mathematical Modelling of an Outbreak of Zombie Infection”. In: *nfectious Disease Modelling Research Progress* (2009), pp. 133–150. preprint (cit. on p. 49).
- [MM06] K. Murphy and S. Mian. *Modelling Gene Expression Data Using Dynamic Bayesian Networks*. Technical Report. University of California, Berkeley, CA, 2006, p. 12 (cit. on p. 65).
- [MHK10] C. Müssel, M. Hopfensitz, and H. A. Kestler. “BoolNet—an R Package for Generation, Reconstruction and Analysis of Boolean Networks”. In: *Bioinformatics* 26.10 (2010), pp. 1378–1380. DOI: [10.1093/bioinformatics/btq124](https://doi.org/10.1093/bioinformatics/btq124) (cit. on pp. 127, 128, 138).
- [NSS12] R. K. Nagle, E. B. Saff, and A. D. Snider. *Fundamentals of Differential Equations*. 8th ed. Boston: Pearson Education, 2012. 644 pp. ISBN: 978-0-321-74773-0 (cit. on p. 29).

- [Nal+10] A. Naldi, J. Carneiro, C. Chaouiya, and D. Thieffry. “Diversity and Plasticity of Th Cell Types Predicted from Regulatory Network Modelling”. In: *PLoS Computational Biology* 6.9 (2010), e1000912. DOI: [10.1371/journal.pcbi.1000912](https://doi.org/10.1371/journal.pcbi.1000912) (cit. on p. 48).
- [Nal+11] A. Naldi, E. Remy, D. Thieffry, and C. Chaouiya. “Dynamically Consistent Reduction of Logical Regulatory Graphs”. In: *Theoretical Computer Science* 412.21 (2011), pp. 2207–2218. DOI: [10.1016/j.tcs.2010.10.021](https://doi.org/10.1016/j.tcs.2010.10.021) (cit. on p. 68).
- [Net+07] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. “MiniZinc: Towards a Standard CP Modelling Language”. In: *Principles and Practice of Constraint Programming*. Vol. 4741. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 529–543. DOI: [10.1007/978-3-540-74970-7_38](https://doi.org/10.1007/978-3-540-74970-7_38) (cit. on p. 139).
- [NVV22] J. Niehren, A. Vaginay, and C. Versari. “Abstract Simulation of Reaction Networks via Boolean Networks”. In: *Computational Methods in Systems Biology*. Vol. 13447. Lecture Notes in Computer Science. Springer, 2022, pp. 21–40. DOI: [10.1007/978-3-031-15034-0_2](https://doi.org/10.1007/978-3-031-15034-0_2) (cit. on pp. 7, 72, 95, 100, 101).
- [Nie+16] J. Niehren, C. Versari, M. John, F. Coutte, and P. Jacques. “Predicting Changes of Reaction Networks with Partial Kinetic Information”. In: *BioSystems*. Special Issue of CMSB 2015 149 (2016), pp. 113–124. DOI: [10.1016/j.biosystems.2016.09.003](https://doi.org/10.1016/j.biosystems.2016.09.003) (cit. on pp. 23, 102).
- [Nob60] D. Noble. “Cardiac Action and Pacemaker Potentials Based on the Hodgkin-Huxley Equations”. In: *Nature* 188.4749 (1960), pp. 495–497. DOI: [10.1038/188495b0](https://doi.org/10.1038/188495b0) (cit. on p. 49).
- [Nor98] D. B. Northrop. “On the Meaning of Km and V/K in Enzyme Kinetics”. In: *Journal of Chemical Education* 75.9 (1998), p. 1153. DOI: [10.1021/ed075p1153](https://doi.org/10.1021/ed075p1153) (cit. on p. 70).
- [NRS11] M. Noual, D. Regnault, and S. Sené. “Non-Monotony and Boolean Automata Networks”. 2011. URL: <http://arxiv.org/abs/1111.4552>. preprint (cit. on pp. 64, 126).
- [Nov+01] B. Novak, Z. Pataki, A. Ciliberto, and J. J. Tyson. “Mathematical Model of the Cell Division Cycle of Fission Yeast”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 11.1 (2001), p. 277. DOI: [10.1063/1.1345725](https://doi.org/10.1063/1.1345725) (cit. on pp. 53, 60).
- [Nov06] N. Le Novère. “Model Storage, Exchange and Integration”. In: *BMC Neuroscience* 7.S1 (2006), S11. DOI: [10.1186/1471-2202-7-S1-S11](https://doi.org/10.1186/1471-2202-7-S1-S11) (cit. on p. 3).
- [Nov13] N. Le Novère. *Modelling Success Stories in Systems Biology*. Blog: Phosphenes. 2013. URL: <https://nlenov.wordpress.com/2013/02/19/modelling-success-stories-in-systems-biology/> (cit. on p. 49).
- [Nov15] N. Le Novère. “Quantitative and Logic Modelling of Molecular and Gene Networks”. In: *Nature Reviews Genetics* 16.3 (2015), pp. 146–158. DOI: [10.1038/nrg3885](https://doi.org/10.1038/nrg3885) (cit. on pp. 2, 4).
- [Nur+22] S. Nur et al. “The Complete Sequence of a Human Genome”. In: *Science* 376.6588 (2022), pp. 44–53. DOI: [10.1126/science.abj6987](https://doi.org/10.1126/science.abj6987) (cit. on p. 60).
- [OM98] M. S. Okino and M. L. Mavrouniotis. “Simplification of Mathematical Models of Chemical Reaction Systems”. In: *Chemical Reviews* 98.2 (1998), pp. 391–408. DOI: [10.1021/cr9502231](https://doi.org/10.1021/cr9502231) (cit. on p. 68).
- [Ost+20] M. Ostaszewski, A. Mazein, M. E. Gillespie, I. Kuperstein, A. Niarakis, H. Hermjakob, A. R. Pico, E. L. Willighagen, C. T. Evelo, J. Hasenauer, F. Schreiber, A. Dräger, E. Demir, O. Wolkenhauer, L. I. Furlong, E. Barillot, J. Dopazo, A. Orta-Resendiz, F. Messina, A. Valencia, A. Funahashi, H. Kitano, C. Auffray, R. Balling, and R. Schneider. “COVID-19 Disease Map, Building a Computational Repository of SARS-CoV-2 Virus-Host Interaction Mechanisms”. In: *Scientific Data* 7.1 (2020), p. 136. DOI: [10.1038/s41597-020-0477-8](https://doi.org/10.1038/s41597-020-0477-8) (cit. on p. 49).
- [Ost+16] M. Ostrowski, L. Paulevé, T. Schaub, A. Siegel, and C. Guziolowski. “Boolean Network Identification from Perturbation Time Series Data Combining Dynamics Abstraction and Logic Programming”. In: *Biosystems* 149 (2016), pp. 139–153. DOI: [10.1016/j.biosystems.2016.07.009](https://doi.org/10.1016/j.biosystems.2016.07.009) (cit. on pp. 59, 61–63, 65, 91, 119, 122, 123, 125, 126).
- [Pap94] C. H. Papadimitriou. *Computational complexity*. 1994. ISBN: 0201530821 (cit. on p. 13).

- [Pau+20] L. Paulevé, J. Kolčák, T. Chatain, and S. Haar. “Reconciling Qualitative, Abstract, and Scalable Modeling of Biological Networks”. In: *Nature Communications* 11.1 (2020), p. 4256. DOI: [10.1038/s41467-020-18112-5](https://doi.org/10.1038/s41467-020-18112-5) (cit. on pp. [42](#), [123](#), [146](#), [150](#), [155](#)).
- [PS21] L. Paulevé and S. Sené. “Non-Deterministic Updates of Boolean Networks”. In: *International Workshop on Cellular Automata and Discrete Complex Systems*. Vol. 90. OASICS. Marseille, France: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 10:1–10:16. DOI: [10.4230/OASICS.AUTOMATA.2021.10](https://doi.org/10.4230/OASICS.AUTOMATA.2021.10) (cit. on p. [92](#)).
- [PS22] L. Paulevé and S. Sené. “Boolean Networks and Their Dynamics: The Impact of Updates”. In: *Systems Biology Modelling and Analysis*. John Wiley & Sons, Ltd, 2022. Chap. 6, pp. 173–250. DOI: [10.1002/9781119716600.ch6](https://doi.org/10.1002/9781119716600.ch6) (cit. on p. [42](#)).
- [Pău00] G. Păun. “Computing with Membranes”. In: *Journal of Computer and System Sciences* 61.1 (2000), pp. 108–143. DOI: [10.1006/jcss.1999.1693](https://doi.org/10.1006/jcss.1999.1693) (cit. on p. [53](#)).
- [Pel+21] A. Pellissier-Tanon, G. Morgado, A. Lemarchand, and L. Jullien. “Quasi-Steady-State and Partial-Equilibrium Approximations in Chemical Kinetics: One Stage beyond the Elimination of a Fast Variable”. 2021. DOI: [10.26434/chemrxiv.14198768.v2](https://doi.org/10.26434/chemrxiv.14198768.v2). preprint (cit. on p. [69](#)).
- [PV22] M. Pelz and M. T. Velcsov. “Entropy Analysis of Boolean Network Reduction According to the Determinative Power of Nodes”. In: *Physica A: Statistical Mechanics and its Applications* 589.C (2022). DOI: [10.1016/j.physa.2021.1266](https://doi.org/10.1016/j.physa.2021.1266) (cit. on p. [68](#)).
- [Pet62] C. A. Petri. “Kommunikation mit Automaten”. PhD thesis. Universität Hamburg, 1962. URL: <http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/> (cit. on p. [55](#)).
- [PK06] S. Philippi and J. Köhler. “Addressing the Problems with Life-Science Databases for Traditional Uses and Systems Biology”. In: *Nature Reviews Genetics* 7.6 (2006), pp. 482–488. DOI: [10.1038/nrg1872](https://doi.org/10.1038/nrg1872) (cit. on p. [3](#)).
- [Pla18] A. Platzer. *Logical Foundations of Cyber-Physical Systems*. Cham: Springer International Publishing, 2018. DOI: [10.1007/978-3-319-63588-0](https://doi.org/10.1007/978-3-319-63588-0) (cit. on pp. [79](#), [81](#)).
- [PG51] R. L. Plunkett and C. L. Gemmill. “The Kinetics of Invertase Action”. In: *The Bulletin of Mathematical Biophysics* 13.4 (1951), pp. 303–312. DOI: [10.1007/BF02477924](https://doi.org/10.1007/BF02477924) (cit. on p. [70](#)).
- [PHD09] A. Polynikis, S. Hogan, and M. Di Bernardo. “Comparing Different ODE Modelling Approaches for Gene Regulatory Networks”. In: *Journal of Theoretical Biology* 261.4 (2009), pp. 511–530. DOI: [10.1016/j.jtbi.2009.07.040](https://doi.org/10.1016/j.jtbi.2009.07.040) (cit. on p. [97](#)).
- [Poo+22] W. Poole, A. Pandey, A. Shur, Z. A. Tuza, and R. M. Murray. “BioCRNpyler: Compiling Chemical Reaction Networks from Biomolecular Parts in Diverse Contexts”. In: *PLoS Computational Biology* 18.4 (2022), e1009987. DOI: [10.1371/journal.pcbi.1009987](https://doi.org/10.1371/journal.pcbi.1009987) (cit. on p. [71](#)).
- [PM14] N. Pullen and R. J. Morris. “Bayesian Model Comparison and Parameter Inference in Systems Biology Using Nested Sampling”. In: *PLoS ONE* 9.2 (2014), e88419. DOI: [10.1371/journal.pone.0088419](https://doi.org/10.1371/journal.pone.0088419) (cit. on p. [64](#)).
- [Qui52] W. V. Quine. “The Problem of Simplifying Truth Functions”. In: *The American Mathematical Monthly* 59.8 (1952), pp. 521–531. DOI: [10.1080/00029890.1952.11988183](https://doi.org/10.1080/00029890.1952.11988183) (cit. on pp. [34](#), [35](#)).
- [RH93] K. Radhakrishnan and A. C. Hindmarsh. *Description and Use of LSODE, the Livemore Solver for Ordinary Differential Equations*. Technical report UCRL-ID-113855, 15013302. NASA, 1993, UCRL-ID-113855, 15013302. DOI: [10.2172/15013302](https://doi.org/10.2172/15013302) (cit. on p. [89](#)).
- [Rad+12] O. Radulescu, A. N. Gorban, A. Zinovyev, and V. Noel. “Reduction of Dynamical Biochemical Reactions Networks in Computational Biology”. In: *Frontiers in Genetics* 3 (2012). DOI: [10.3389/fgene.2012.00131](https://doi.org/10.3389/fgene.2012.00131) (cit. on p. [68](#)).
- [RA03] C. V. Rao and A. P. Arkin. “Stochastic Chemical Kinetics and the Quasi-Steady-State Assumption: Application to the Gillespie Algorithm”. In: *The Journal of Chemical Physics* 118.11 (2003), pp. 4999–5010. DOI: [10.1063/1.1545446](https://doi.org/10.1063/1.1545446) (cit. on p. [69](#)).

- [Ras+14] J. Raspopovic, L. Marcon, L. Russo, and J. Sharpe. “Modeling Digits. Digit Patterning Is Controlled by a Bmp-Sox9-Wnt Turing Network Modulated by Morphogen Gradients”. In: *Science (New York, N.Y.)* 345.6196 (2014), pp. 566–570. DOI: [10.1126/science.1252960](https://doi.org/10.1126/science.1252960) (cit. on p. 49).
- [RML94] V. N. Reddy, M. L. Mavrovouniotis, and M. N. Liebman. “Modeling Biological Pathways”. In: *Molecular Modeling*. Vol. 576. ACS Symposium Series 576. American Chemical Society, 1994. Chap. 14, pp. 221–234. DOI: [10.1021/bk-1994-0576.ch014](https://doi.org/10.1021/bk-1994-0576.ch014) (cit. on p. 55).
- [RS02] A. Regev and E. Shapiro. “Cellular Abstractions: Cells as Computation”. In: *Nature* 419.6905 (2002), pp. 343–343. DOI: [10.1038/419343a](https://doi.org/10.1038/419343a) (cit. on p. 3).
- [Ric19] A. Richard. “Positive and Negative Cycles in Boolean Networks”. In: *Journal of Theoretical Biology* 463 (2019), pp. 67–76. DOI: [10.1016/j.jtbi.2018.11.028](https://doi.org/10.1016/j.jtbi.2018.11.028) (cit. on p. 67).
- [Ric05] K. A. Richardson. “Simplifying Boolean Networks”. In: *Advances in Complex Systems* 08.04 (2005), pp. 365–381. DOI: [10.1142/S0219525905000518](https://doi.org/10.1142/S0219525905000518) (cit. on p. 68).
- [Rod+20] I. Rodchenkov, O. Babur, A. Luna, B. A. Aksoy, J. V. Wong, D. Fong, M. Franz, M. C. Siper, M. Cheung, M. Wrana, H. Mistry, L. Mosier, J. Dlin, Q. Wen, C. O’Callaghan, W. Li, G. Elder, P. T. Smith, C. Dallago, E. Cerami, B. Gross, U. Dogrusoz, E. Demir, G. D. Bader, and C. Sander. “Pathway Commons 2019 Update: Integration, Analysis and Exploration of Pathway Data”. In: *Nucleic Acids Research* 48.D1 (2020), pp. D489–D497. DOI: [10.1093/nar/gkz946](https://doi.org/10.1093/nar/gkz946) (cit. on p. 60).
- [Ros08] B. Rossman. “Homomorphism Preservation Theorems”. In: *Journal of the ACM* 55.3 (2008), pp. 1–53. DOI: [10.1145/1379759.1379763](https://doi.org/10.1145/1379759.1379763) (cit. on p. 23).
- [SAR13] A. Saadatpour, R. Albert, and T. C. Reluga. “A Reduction Method for Boolean Network Models Proven to Conserve Attractors”. In: *SIAM Journal on Applied Dynamical Systems* 12.4 (2013), pp. 1997–2011. DOI: [10.1137/13090537X](https://doi.org/10.1137/13090537X) (cit. on p. 68).
- [Sae+09] J. Saez-Rodriguez, L. G. Alexopoulos, J. Epperlein, R. Samaga, D. A. Lauffenburger, S. Klamt, and P. K. Sorger. “Discrete Logic Modelling as a Means to Link Protein Signalling Networks with Functional Analysis of Mammalian Signal Transduction”. In: *Molecular Systems Biology* 5.1 (2009), p. 331. DOI: [10.1038/msb.2009.87](https://doi.org/10.1038/msb.2009.87) (cit. on p. 124).
- [Sal13a] A. Salomaa. “Functional Constructions Between Reaction Systems and Propositional Logic”. In: *International Journal of Foundations of Computer Science* 24.1 (2013), pp. 147–160. DOI: [10.1142/S0129054113500044](https://doi.org/10.1142/S0129054113500044) (cit. on p. 67).
- [Sal13b] A. Salomaa. “Minimal and Almost Minimal Reaction Systems”. In: *Natural Computing* 12.3 (2013), pp. 369–376. DOI: [10.1007/s11047-013-9372-y](https://doi.org/10.1007/s11047-013-9372-y) (cit. on p. 67).
- [SLP00] C. H. Schilling, D. Letscher, and B. Ø. Palsson. “Theory for the Systemic Definition of Metabolic Pathways and Their Use in Interpreting Metabolic Function from a Pathway-Oriented Perspective”. In: *Journal of Theoretical Biology* 203.3 (2000), pp. 229–248. DOI: [10.1006/jtbi.2000.1073](https://doi.org/10.1006/jtbi.2000.1073) (cit. on p. 67).
- [Sch99] S. Schuster. “Detection of Elementary Flux Modes in Biochemical Networks: A Promising Tool for Pathway Analysis and Metabolic Engineering”. In: *Trends in Biotechnology* 17.2 (1999), pp. 53–60. DOI: [10.1016/S0167-7799\(98\)01290-6](https://doi.org/10.1016/S0167-7799(98)01290-6) (cit. on p. 67).
- [Sch+20] J. D. Schwab, S. D. Kühlwein, N. Ikonomi, M. Köhl, and H. A. Kestler. “Concepts in Boolean Network Modeling: What Do They All Mean?” In: *Computational and Structural Biotechnology Journal* 18 (2020), pp. 571–582. DOI: [10.1016/j.csbj.2020.03.001](https://doi.org/10.1016/j.csbj.2020.03.001) (cit. on p. 57).
- [Sch78] G. Schwarz. “Estimating the Dimension of a Model”. In: *The Annals of Statistics* 6.2 (1978). DOI: [10.1214/aos/1176344136](https://doi.org/10.1214/aos/1176344136) (cit. on p. 65).
- [SG79] L. F. Shampine and C. W. Gear. “A User’s View of Solving Stiff Ordinary Differential Equations”. In: *SIAM Review* 21.1 (1979), pp. 1–17. DOI: [10.1137/1021001](https://doi.org/10.1137/1021001) (cit. on p. 89).
- [Sha48] C. E. Shannon. “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: [10.1002/j.1538-7305.1948.tb01338.x](https://doi.org/10.1002/j.1538-7305.1948.tb01338.x) (cit. on p. 64).

- [Shi+20] N. Shi, Z. Zhu, K. Tang, D. Parker, and S. He. “ATEN: And/Or Tree Ensemble for Inferring Accurate Boolean Network Topology and Dynamics”. In: *Bioinformatics* 36.2 (2020), pp. 578–585. DOI: [10.1093/bioinformatics/btz563](https://doi.org/10.1093/bioinformatics/btz563) (cit. on p. 66).
- [Sno89] E. H. Snoussi. “Qualitative Dynamics of Piecewise-Linear Differential Equations: A Discrete Mapping Approach”. In: *Dynamics and Stability of Systems* 4.3-4 (1989), pp. 565–583. DOI: [10.1080/02681118908806072](https://doi.org/10.1080/02681118908806072) (cit. on pp. 73, 97).
- [SH10] S. Soliman and M. Heiner. “A Unique Transformation from Ordinary Differential Equations to Reaction Networks”. In: *PLoS ONE* 5.12 (2010), e14284. DOI: [10.1371/journal.pone.0014284](https://doi.org/10.1371/journal.pone.0014284) (cit. on p. 99).
- [Son07] E. D. Sontag. “Monotone and Near-Monotone Biochemical Networks”. In: *Systems and Synthetic Biology* 1.2 (2007), pp. 59–87. DOI: [10.1007/s11693-007-9005-9](https://doi.org/10.1007/s11693-007-9005-9) (cit. on p. 119).
- [Spe+98] P. T. Spellman, G. Sherlock, M. Q. Zhang, V. R. Iyer, K. Anders, M. B. Eisen, P. O. Brown, D. Botstein, and B. Futcher. “Comprehensive Identification of Cell Cycle-Regulated Genes of the Yeast *Saccharomyces Cerevisiae* by Microarray Hybridization”. In: *Molecular Biology of the Cell* 9.12 (1998), pp. 3273–3297. DOI: [10.1091/mbc.9.12.3273](https://doi.org/10.1091/mbc.9.12.3273) (cit. on pp. 60, 126, 127).
- [Spi04] A. Spivey. “Systems Biology: The Big Picture”. In: *Environmental Health Perspectives* 112.16 (2004), A938–A943. ISSN: 0091-6765 (cit. on p. 1).
- [SP94] M. Srinivas and L. Patnaik. “Genetic Algorithms: A Survey”. In: *Computer* 27.6 (1994), pp. 17–26. DOI: [10.1109/2.294849](https://doi.org/10.1109/2.294849) (cit. on p. 66).
- [Sri22] B. Srinivasan. “A Guide to the Michaelis–Menten Equation: Steady State and Beyond”. In: *The FEBS Journal* 289.20 (2022), pp. 6086–6098. DOI: [10.1111/febs.16124](https://doi.org/10.1111/febs.16124) (cit. on p. 51).
- [Stä+21] P. Städter, Y. Schälte, L. Schmiester, J. Hasenauer, and P. L. Stapor. “Benchmarking of Numerical Integration Methods for ODE Models of Biological Systems”. In: *Scientific Reports* 11.1 (2021), p. 2696. DOI: [10.1038/s41598-021-82196-2](https://doi.org/10.1038/s41598-021-82196-2) (cit. on p. 29).
- [Str09] S. Strogatz. *Guest Column: Loves Me, Loves Me Not (Do the Math)*. Opinionator. 2009. URL: <https://nyti.ms/2EHozQE> (cit. on p. 27).
- [SMS22] A. Subbaroyan, O. C. Martin, and A. Samal. “Minimum Complexity Drives Regulatory Logic in Boolean Models of Living Systems”. In: *PNAS Nexus* 1.1 (2022), pgac017. DOI: [10.1093/pnasnexus/pgac017](https://doi.org/10.1093/pnasnexus/pgac017) (cit. on p. 36).
- [Sub+17] A. Subramanian, R. Narayan, S. M. Corsello, D. D. Peck, T. E. Natoli, X. Lu, J. Gould, J. F. Davis, A. A. Tubelli, J. K. Asiedu, D. L. Lahr, J. E. Hirschman, Z. Liu, M. Donahue, B. Julian, M. Khan, D. Wadden, I. C. Smith, D. Lam, A. Liberzon, C. Toder, M. Bagul, M. Orzechowski, O. M. Enache, F. Piccioni, S. A. Johnson, N. J. Lyons, A. H. Berger, A. F. Shamji, A. N. Brooks, A. Vrcic, C. Flynn, J. Rosains, D. Y. Takeda, R. Hu, D. Davison, J. Lamb, K. Ardlie, L. Hogstrom, P. Greenside, N. S. Gray, P. A. Clemons, S. Silver, X. Wu, W.-N. Zhao, W. Read-Button, X. Wu, S. J. Haggarty, L. V. Ronco, J. S. Boehm, S. L. Schreiber, J. G. Doench, J. A. Bittker, D. E. Root, B. Wong, and T. R. Golub. “A Next Generation Connectivity Map: L1000 Platform and the First 1,000,000 Profiles”. In: *Cell* 171.6 (2017), 1437–1452.e17. DOI: [10.1016/j.cell.2017.10.049](https://doi.org/10.1016/j.cell.2017.10.049) (cit. on p. 60).
- [SLG06] J. R. Swedlow, S. E. Lewis, and I. G. Goldberg. “Modelling Data across Labs, Genomes, Space and Time”. In: *Nature Cell Biology* 8.11 (2006), pp. 1190–1194. DOI: [10.1038/ncb1496](https://doi.org/10.1038/ncb1496) (cit. on p. 3).
- [Szi+18] B. Szigeti, Y. D. Roth, J. A. Sekar, A. P. Goldberg, S. C. Pochiraju, and J. R. Karr. “A Blueprint for Human Whole-Cell Modeling”. In: *Current Opinion in Systems Biology* 7 (2018), pp. 8–15. DOI: [10.1016/j.coisb.2017.10.005](https://doi.org/10.1016/j.coisb.2017.10.005) (cit. on p. 49).
- [Ter+12] C. Terfve, T. Cokelaer, D. Henriques, A. MacNamara, E. Goncalves, M. K. Morris, M. V. Iersel, D. A. Lauffenburger, and J. Saez-Rodriguez. “CellNOptR: A Flexible Toolkit to Train Protein Signaling Networks to Data Using Multiple Logic Formalisms”. In: *BMC Systems Biology* 6.1 (2012), p. 133. DOI: [10.1186/1752-0509-6-133](https://doi.org/10.1186/1752-0509-6-133) (cit. on p. 124).

- [Ter+19] G. Terje Lines, Ł. Paszkowski, L. Schmiester, D. Weindl, P. Stapor, and J. Hasenauer. “Efficient Computation of Steady States in Large-Scale ODE Models of Biochemical Reaction Networks”. In: *IFAC-PapersOnLine* 52.26 (2019). 8th Conference on Foundations of Systems Biology in Engineering FOSBE 2019, pp. 32–37. DOI: [10.1016/j.ifacol.2019.12.232](https://doi.org/10.1016/j.ifacol.2019.12.232) (cit. on p. 29).
- [Tho81] R. Thomas. “On the Relation Between the Logical Structure of Systems and Their Ability to Generate Multiple Steady States or Sustained Oscillations”. In: *Numerical Methods in the Study of Critical Phenomena*. Vol. 9. Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, pp. 180–193. DOI: [10.1007/978-3-642-81703-8_24](https://doi.org/10.1007/978-3-642-81703-8_24) (cit. on p. 67).
- [Tho73] R. Thomas. “Boolean Formalization of Genetic Control Circuits”. In: *Journal of Theoretical Biology* (1973), p. 23. DOI: [10.1016/0022-5193\(73\)90247-6](https://doi.org/10.1016/0022-5193(73)90247-6) (cit. on p. 97).
- [Tho91] R. Thomas. “Regulatory Networks Seen as Asynchronous Automata: A Logical Description”. In: *Journal of Theoretical Biology* 153.1 (1991), pp. 1–23. DOI: [10.1016/S0022-5193\(05\)80350-9](https://doi.org/10.1016/S0022-5193(05)80350-9) (cit. on pp. 42, 57).
- [TK21] H.-C. Trinh and Y.-K. Kwon. “A Novel Constrained Genetic Algorithm-Based Boolean Network Inference Method from Steady-State Gene Expression Data”. In: *Bioinformatics* 37 (2021), pp. i383–i391. DOI: [10.1093/bioinformatics/btab295](https://doi.org/10.1093/bioinformatics/btab295) (cit. on p. 66).
- [Tur+21] B. Turanli, O. Altay, J. Borén, H. Turkez, J. Nielsen, M. Uhlen, K. Y. Arga, and A. Mardinoglu. “Systems Biology Based Drug Repositioning for Development of Cancer Therapy”. In: *Seminars in Cancer Biology* 68 (2021), pp. 47–58. DOI: [10.1016/j.semcancer.2019.09.020](https://doi.org/10.1016/j.semcancer.2019.09.020) (cit. on p. 1).
- [Tur52] A. Turing. “The Chemical Basis of Morphogenesis”. In: *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 237.641 (1952), pp. 37–72. DOI: [10.1098/rstb.1952.0012](https://doi.org/10.1098/rstb.1952.0012) (cit. on p. 49).
- [TCN01] J. J. Tyson, K. Chen, and B. Novak. “Network Dynamics and Cell Physiology”. In: *Nature Reviews Molecular Cell Biology* 2.12 (2001), pp. 908–916. DOI: [10.1038/35103078](https://doi.org/10.1038/35103078) (cit. on p. 29).
- [Tys+96] J. J. Tyson, B. Novak, G. M. Odell, K. Chen, and C. Dennis Thron. “Chemical Kinetic Theory: Understanding Cell-Cycle Regulation”. In: *Trends in Biochemical Sciences* 21.3 (1996), pp. 89–96. DOI: [10.1016/S0968-0004\(96\)10011-6](https://doi.org/10.1016/S0968-0004(96)10011-6) (cit. on p. 91).
- [Uth21] A. Uthamacumaran. “A Review of Dynamical Systems Approaches for the Detection of Chaotic Attractors in Cancer Networks”. In: *Patterns* 2.4 (2021), p. 100226. DOI: [10.1016/j.patter.2021.100226](https://doi.org/10.1016/j.patter.2021.100226) (cit. on p. 68).
- [VBS21a] A. Vaginay, T. Boukhobza, and M. Smaïl-Tabbone. “Automatic Synthesis of Boolean Networks from Biological Knowledge and Data”. In: *Optimization and Learning*. Communications in Computer and Information Science. Cham: Springer International Publishing, 2021, pp. 156–170. DOI: [10.1007/978-3-030-85672-4_12](https://doi.org/10.1007/978-3-030-85672-4_12) (cit. on pp. 8, 104).
- [VBS21b] A. Vaginay, T. Boukhobza, and M. Smaïl-Tabbone. “From Quantitative SBML Models to Boolean Networks”. In: *Complex Networks & Their Applications*. Vol. 1016. Studies in Computational Intelligence. Springer, 2021, pp. 676–687. DOI: [10.1007/978-3-030-93413-2_56](https://doi.org/10.1007/978-3-030-93413-2_56) (cit. on p. 7).
- [VBS22] A. Vaginay, T. Boukhobza, and M. Smaïl-Tabbone. “From quantitative SBML models to Boolean networks”. In: *Applied Network Science* 7.1 (2022), p. 73. DOI: [10.1007/s41109-022-00505-8](https://doi.org/10.1007/s41109-022-00505-8) (cit. on p. 7).
- [Val+06] M. Valorani, F. Creta, D. A. Goussis, J. C. Lee, and H. N. Najm. “An Automatic Procedure for the Simplification of Chemical Kinetic Mechanisms Based on CSP”. In: *Combustion and Flame* 146.1-2 (2006), pp. 29–51. DOI: [10.1016/j.combustflame.2006.03.011](https://doi.org/10.1016/j.combustflame.2006.03.011) (cit. on p. 68).
- [Van+23] L. Van Hirtum, P. De Causmaecker, J. Goemaere, T. Kenter, H. Riebler, M. Lass, and C. Plessl. “A Computation of D(9) Using FPGA Supercomputing”. 2023. DOI: [10.48550/arXiv.2304.03039](https://doi.org/10.48550/arXiv.2304.03039). preprint (cit. on p. 36).
- [VP94] A. Varma and B. O. Palsson. “Metabolic Flux Balancing: Basic Concepts, Scientific and Practical Use”. In: *Bio/Technology* 12.10 (1994), pp. 994–998. DOI: [10.1038/nbt1094-994](https://doi.org/10.1038/nbt1094-994) (cit. on p. 67).

- [Vel11] A. Veliz-Cuba. “Reduction of Boolean Network Models”. In: *Journal of Theoretical Biology* (2011). DOI: [10.1016/j.jtbi.2011.08.042](https://doi.org/10.1016/j.jtbi.2011.08.042) (cit. on p. 68).
- [Vel+14] A. Veliz-Cuba, B. Aguilar, F. Hinkelmann, and R. Laubenbacher. “Steady State Analysis of Boolean Molecular Network Models via Model Reduction and Computational Algebra”. In: *BMC Bioinformatics* 15.1 (2014), p. 221. DOI: [10.1186/1471-2105-15-221](https://doi.org/10.1186/1471-2105-15-221) (cit. on p. 68).
- [Vid+15] S. Videla, C. Guziolowski, F. Eduati, S. Thiele, M. Gebser, J. Nicolas, J. Saez-Rodriguez, T. Schaub, and A. Siegel. “Learning Boolean Logic Models of Signaling Networks with ASP”. In: *Theoretical Computer Science* 599 (2015), pp. 79–101. DOI: [10.1016/j.tcs.2014.06.022](https://doi.org/10.1016/j.tcs.2014.06.022) (cit. on pp. 91, 122, 124).
- [Vod23] Y. Vodovotz. “Towards Systems Immunology of Critical Illness at Scale: From Single Cell ‘omics to Digital Twins”. In: *Trends in Immunology* 44.5 (2023), pp. 345–355. DOI: [10.1016/j.it.2023.03.004](https://doi.org/10.1016/j.it.2023.03.004) (cit. on p. 2).
- [Vol26] V. Volterra. “Fluctuations in the Abundance of a Species Considered Mathematically1”. In: *Nature* 118.2972 (1926), pp. 558–560. DOI: [10.1038/118558a0](https://doi.org/10.1038/118558a0) (cit. on p. 49).
- [WG64] P. Waage and C. M. Gulberg. “Studies Concerning Affinity”. In: *Journal of Chemical Education* (1864). DOI: [10.1021/ed063p1044](https://doi.org/10.1021/ed063p1044) (cit. on pp. 45, 51).
- [Wal98] W. Walter. *Ordinary Differential Equations*. Vol. 182. Graduate Texts in Mathematics. New York, NY: Springer New York, 1998. DOI: [10.1007/978-1-4612-0601-9](https://doi.org/10.1007/978-1-4612-0601-9) (cit. on p. 29).
- [WBS19] D. J. Warne, R. E. Baker, and M. J. Simpson. “Simulation and Inference Algorithms for Stochastic Biochemical Reaction Networks: From Basic Concepts to State-of-the-Art”. In: *Journal of The Royal Society Interface* 16.151 (2019), p. 20180943. DOI: [10.1098/rsif.2018.0943](https://doi.org/10.1098/rsif.2018.0943) (cit. on p. 67).
- [Whi12] J. M. Whitacre. “Biological Robustness: Paradigms, Mechanisms, and Systems Principles”. In: *Frontiers in Genetics* 3 (2012). DOI: [10.3389/fgene.2012.00067](https://doi.org/10.3389/fgene.2012.00067) (cit. on p. 1).
- [Wil07] D. J. Wilkinson. “Bayesian Methods in Bioinformatics and Computational Systems Biology”. In: *Briefings in Bioinformatics* 8.2 (2007), pp. 109–116. DOI: [10.1093/bib/bbm007](https://doi.org/10.1093/bib/bbm007) (cit. on p. 67).
- [Wil09] R. J. Wilson. *Introduction to Graph Theory*. 4. ed., [Nachdr.] Harlow Munich: Prentice Hall, 2009. 171 pp. ISBN: 978-0-582-24993-6 (cit. on p. 12).
- [Wix01] J. Wixon. “Pathway Databases”. In: *Comparative and Functional Genomics* 2.6 (2001), pp. 391–397. DOI: [10.1002/cfg.123](https://doi.org/10.1002/cfg.123) (cit. on p. 60).
- [WL05] J. C. Wooley and H. S. Lin. *Computational Modeling and Simulation as Enablers for Biological Discovery*. National Academies Press (US), 2005. URL: <https://www.ncbi.nlm.nih.gov/books/NBK25466/> (cit. on p. 1).
- [Woo+21] D. J. Wooten, J. G. T. Zañudo, D. Murrugarra, A. M. Perry, A. Dongari-Bagtzoglou, R. Laubenbacher, C. J. Nobile, and R. Albert. “Mathematical Modeling of the Candida Albicans Yeast to Hyphal Transition Reveals Novel Control Strategies”. In: *PLoS Computational Biology* 17.3 (2021), e1008690. DOI: [10.1371/journal.pcbi.1008690](https://doi.org/10.1371/journal.pcbi.1008690) (cit. on p. 48).
- [YYC19] Y. Yan, J. Yue, and Z. Chen. “Algebraic Method of Simplifying Boolean Networks Using Semi-tensor Product of Matrices”. In: *Asian Journal of Control* 21.6 (2019), pp. 2569–2577. DOI: [10.1002/asjc.2125](https://doi.org/10.1002/asjc.2125) (cit. on p. 68).
- [ZA13] J. G. T. Zañudo and R. Albert. “An Effective Network Reduction Approach to Find the Dynamical Repertoire of Discrete Dynamic Networks”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 23.2 (2013), p. 025111. DOI: [10.1063/1.4809777](https://doi.org/10.1063/1.4809777) (cit. on p. 68).
- [Zha+20] F. Zhang, L. P. Smith, M. L. Blinov, J. Faeder, W. S. Hlavacek, J. Juan Tapia, S. M. Keating, N. Rodriguez, A. Dräger, L. A. Harris, A. Finney, B. Hu, M. Hucka, and M. Meier-Schellersheim. “Systems Biology Markup Language (SBML) Level 3 Package: Multistate, Multicomponent and Multicompartment Species, Version 1, Release 2”. In: *Journal of Integrative Bioinformatics* 17.2-3 (2020), p. 20200015. DOI: [10.1515/jib-2020-0015](https://doi.org/10.1515/jib-2020-0015) (cit. on p. 53).

- [ZBH20] A. Zupanic, H. C. Bernstein, and I. Heiland. “Systems Biology: Current Status and Challenges”. In: *Cellular and Molecular Life Sciences* 77.3 (2020), pp. 379–380. DOI: [10.1007/s00018-019-03410-z](https://doi.org/10.1007/s00018-019-03410-z) (cit. on p. 1).

BIBLIOGRAPHY

Appendix A

SBML file for \mathcal{R}_{enz}

The following encodes for [Example 1.9.4](#) on page 44.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml level="2" version="3" xmlns="http://www.sbml.org/sbml/level2/version3">
  <model name="EnzymaticReaction">
    <listOfUnitDefinitions>
      <unitDefinition id="per_second">
        <listOfUnits>
          <unit kind="second" exponent="-1"/>
        </listOfUnits>
      </unitDefinition>
      <unitDefinition id="litre_per_mole_per_second">
        <listOfUnits>
          <unit kind="mole" exponent="-1"/>
          <unit kind="litre" exponent="1"/>
          <unit kind="second" exponent="-1"/>
        </listOfUnits>
      </unitDefinition>
    </listOfUnitDefinitions>
    <listOfCompartments>
      <compartment id="cytosol" size="1e-14"/>
    </listOfCompartments>
    <listOfSpecies>
      <species compartment="cytosol" id="ES" initialAmount="0" name="ES"/>
      <species compartment="cytosol" id="P" initialAmount="0" name="P"/>
      <species compartment="cytosol" id="S" initialAmount="1e-20" name="S"/>
      <species compartment="cytosol" id="E" initialAmount="5e-21" name="E"/>
    </listOfSpecies>
    <listOfReactions>
      <reaction id="veq">
        <listOfReactants>
          <speciesReference species="E" stoichiometry="1"/>
          <speciesReference species="S" stoichiometry="1"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="ES" stoichiometry="1"/>
        </listOfProducts>
        <kineticLaw>
```

```

<math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
    <times/>
    <ci>cytosol</ci>
    <apply>
      <minus/>
      <apply>
        <times/>
        <ci>kon</ci><ci>E</ci> <ci>S</ci>
      </apply>
    </apply>
    <apply>
      <times/>
      <ci>koff</ci><ci>ES</ci>
    </apply>
  </apply>
</math>
<listOfParameters>
  <parameter id="kon" value="1000000" units="litre_per_mole_per_second"/>
  <parameter id="koff" value="0.2" units="per_second"/>
</listOfParameters>
</kineticLaw>
</reaction>
<reaction id="vcat" reversible="false">
  <listOfReactants>
    <speciesReference species="ES" stoichiometry="1"/>
  </listOfReactants>
  <listOfProducts>
    <speciesReference species="E" stoichiometry="1"/>
    <speciesReference species="P" stoichiometry="2"/>
  </listOfProducts>
  <kineticLaw>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <times/>
        <ci>cytosol</ci> <ci>kcat</ci> <ci>ES</ci>
      </apply>
    </math>
    <listOfParameters>
      <parameter id="kcat" value="0.1" units="per_second"/>
    </listOfParameters>
  </kineticLaw>
</reaction>
</listOfReactions>
</model>
</sbml>

```

Appendix B

Analytical Solution of Equation (3.1)

We show how to obtain the analytical solution of [equation \(3.1\)](#). Let a , v and x denote $a(t)$, $v(t)$ and $x(t)$ respectively.

(1) We start with the system in [equation \(3.1\)](#) except that we rewrite the differential with the $\frac{dx}{dt}$ notation instead of \dot{x} .

$$\begin{aligned}\frac{da}{dt} &= 0 \\ \frac{dv}{dt} &= a \\ \frac{dx}{dt} &= v\end{aligned}$$

(2) We multiply both sides of each equation by dt :

$$\begin{aligned}da &= 0 \, dt \\ dv &= a \, dt \\ dx &= v \, dt\end{aligned}$$

(3) Now, we integrate both sides. Let's start with the equation of a :

$$\begin{aligned}\int da &= \int 0 \, dt \\ a &= C_1 \quad (\text{where } C_1 \text{ is an integration constant})\end{aligned}$$

For v , we have:

$$\begin{aligned}\int dv &= \int a \, dt \\ \int dv &= \int C_1 \, dt \quad (\text{we use the value we just found for } a) \\ v &= C_1 t + C_2 \quad (\text{where } C_2 \text{ is an integration constant})\end{aligned}$$

Finally, for x :

$$\begin{aligned}\int dx &= \int v \, dt \\ \int dx &= \int (C_1 t + C_2) \, dt \quad (\text{we use the value we just found for } v) \\ x &= C_1 \frac{t^2}{2} + C_2 t + C_3 \quad (\text{where } C_3 \text{ is an integration constant})\end{aligned}$$

In summary, the analytical solution of equation (3.1) is

$$\begin{aligned}a &= C_1 \\ v &= C_1 t + C_2 \\ x &= C_1 \frac{t^2}{2} + C_2 t + C_3\end{aligned}\tag{B.1}$$

with C_1, C_2, C_3 being constants.

Appendix C

Extended abstracts (en/fr)

Extended Abstract

Life and other processes at its periphery are made possible by intertwined (bio)-chemical transformations that form so-called biological *systems*. Biological systems are often described as *complex*, because their dynamics are not easily derived from the static knowledge of their constituents in isolation [Spi04].

The comprehensive and systematic understanding of the dynamics of these systems and of the underlying mechanisms is the subject of *systems biology* [ZBH20; Bre10]. As for other fields, the core tool of systems biology are *models* [Laz02; WL05]. In the broadest sense, a model is an abstract representation (abbreviated and convenient) of the reality (more complex and detailed) [Koh+10]. A model relies on some assumptions, and it can be used to make predictions one can trust if the model is correct [Gun14]. As a matter of fact, the correctness of a model is hard to evaluate, but it is estimated through the following points:

1. the validity of its assumptions,
2. how well its predictions fit the existing knowledge and experimental data.

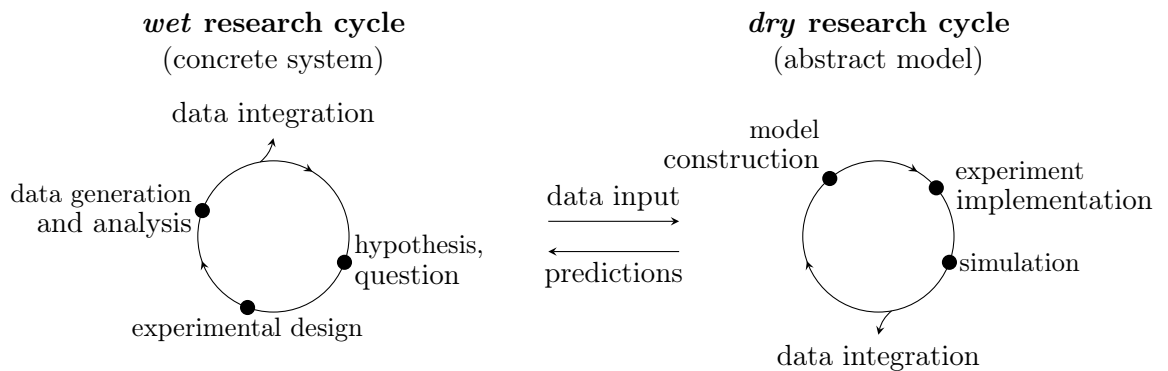
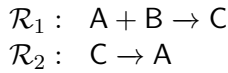


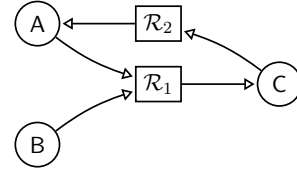
Figure C.1: Current consensus of the systems biology cycle adapted from [BS20, Figure 1]. The “wet” and “dry” research cycles consist basically of the same steps, and can provide insights in parallel to one another.

The assumptions and predictions of these models typically concern the *structure* of the system under study as well as its *dynamics* [RS02]. The former is about the components involved and how they influence each other, while the latter is about patterns in the dynamics of the system, such as the successive and stable configurations, or how it reacts to some perturbations. To achieve sufficient correctness, models and wet experiments are typically refined over repeated iterations of hypothesis formulation, modelling, prediction, and experimentation. This is the main idea of the so-called cycle of systems biology. The current consensus about this cycle is summarised in [figure C.1](#).

The cycle produces models in various formalisms. These formalisms are often categorised into two classes: the mechanism-based models and the influence-based models. Models from these two classes have very different philosophies. Indeed, models from the first category mimic the underlying processes using the concept of reaction, while models from the second category make sense of the observations by encoding the relationships between the components of the system. Reaction networks and Boolean networks are the respective epitome of mechanism-based models and influence-based models. We show an example of a reaction network and of a Boolean network in [figures C.2](#) and [C.3](#) respectively. In the case of Boolean networks, the relationships are encoded by n Boolean functions (one per component).



(a)

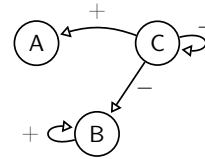


(b)

Figure C.2: A reaction network (a) and its graphical representation (b). The reaction \mathcal{R}_1 transforms one pair formed of one instance of A and one instance of B into one instance of C and the reaction \mathcal{R}_2 transforms one C into one A.

$$\mathcal{B}_1 = \begin{cases} f_A : A_{\text{next}} = C \\ f_B : B_{\text{next}} = B \wedge \neg C \\ f_C : C_{\text{next}} = \neg C \end{cases}$$

(a)



(b)

Figure C.3: A Boolean network (a) and its influence graph (b).

Each formalism has its own strengths and weaknesses [Nov15]. The choice of which to use is guided by the question at hand: the best one is the simplest one which is sufficient to answer the question [Bak+18; Bor05]. In the literature, most of the models correspond to detailed reaction networks usually studied using the formalisms of ordinary differential equations. However, and while counter-intuitive, it is sometimes interesting to downgrade (abstract) to a “simpler” (coarser, less granular) formalism [DB08].

In this thesis, we challenged ourselves to improve the model development cycle. The novelty

is in that we do so by up-cycling knowledge and data retrieved from existing models, instead of producing new ones through wet experiments. More specifically, we develop an approach to *automatically* synthesise Boolean networks from *existing* reaction networks. We think the conversion from one to the other is relevant, at least for the following three reasons:

1. From an application standpoint, it would be an *ad hoc* solution to ease some analyses that are hard to run on reaction network models. Such analyses are related to the control, or the exploration of the dynamic properties of the system.
2. From a theoretical standpoint, having access to such a conversion would certainly help to understand the relationship between reaction networks and Boolean networks better.
3. In turn, we think exploring reaction network to Boolean network conversion is useful to improve the Boolean network synthesis methods that use real biological data. Indeed, due to the proximity of reaction networks with their underlying biological systems, one can consider every Boolean network synthesis study to correspond indirectly to an implicit conversion from a reaction network to a Boolean network. Hence, any finding in a “controlled” synthesis (using inputs extracted from existing reaction network models) might be adapted back to “uncontrolled” synthesis (using real biological data).

The steps of the proposed approach are summarised in [figure C.4](#). It is inspired by what is classically done by the existing model synthesis methods. In particular, it is formulated as a parameter-fitting task, and the information used to fit the Boolean networks pertains the *structure* and the *dynamics* of the underlying system.

For a given reaction network, we define its structure as a graph which captures the *direct* influences of its components. We can retrieve such a graph by simply parsing the reactions of the input model. As for the dynamics, we explore two ways of retrieving it. Both ways rely on the simulation of ordinary differential equations (ODEs) retrieved from the reaction network. First, the qualitative dynamics of a reaction network is obtained by *abstract simulation*. For this, we construct a first-order logic formula from the ODEs of the reaction network. We call this formula a *first-order Boolean networks with non-deterministic updates* because its abstract interpretation (where the precise values of the variables are replaced by their sign) defines a transition relation on the Boolean configurations of the system. It can thus be used to build a Boolean transition graph. We show that this graph is a correct overapproximation relatively to the Euler simulation of the ODEs. Second, we use a *concrete simulation* of the ODEs, followed by a binarisation step. This second approach may seem naive at first glance, but the problem is not that simple as ODEs of biological system exhibit so-called stiff behaviours (mix of slow and rapid evolutions). Hence, *clever* simulation algorithms have to be used to balance the trade-off between the accuracy and the efficiency of the simulation. Moreover, as it is more direct, it allows us to retrieve process reaction networks for which the abstract simulation is not yet applicable. In particular, if the reaction network is coupled with discontinuous events, which is the case for real-life models available in the literature.

Then, the Boolean network synthesis step consists in finding *all* the Boolean networks *compatible* with a given structure and dynamics. As mentioned above, this step is formulated as a parameter-fitting task. More precisely, we see it as a constraint satisfaction problem mixed with

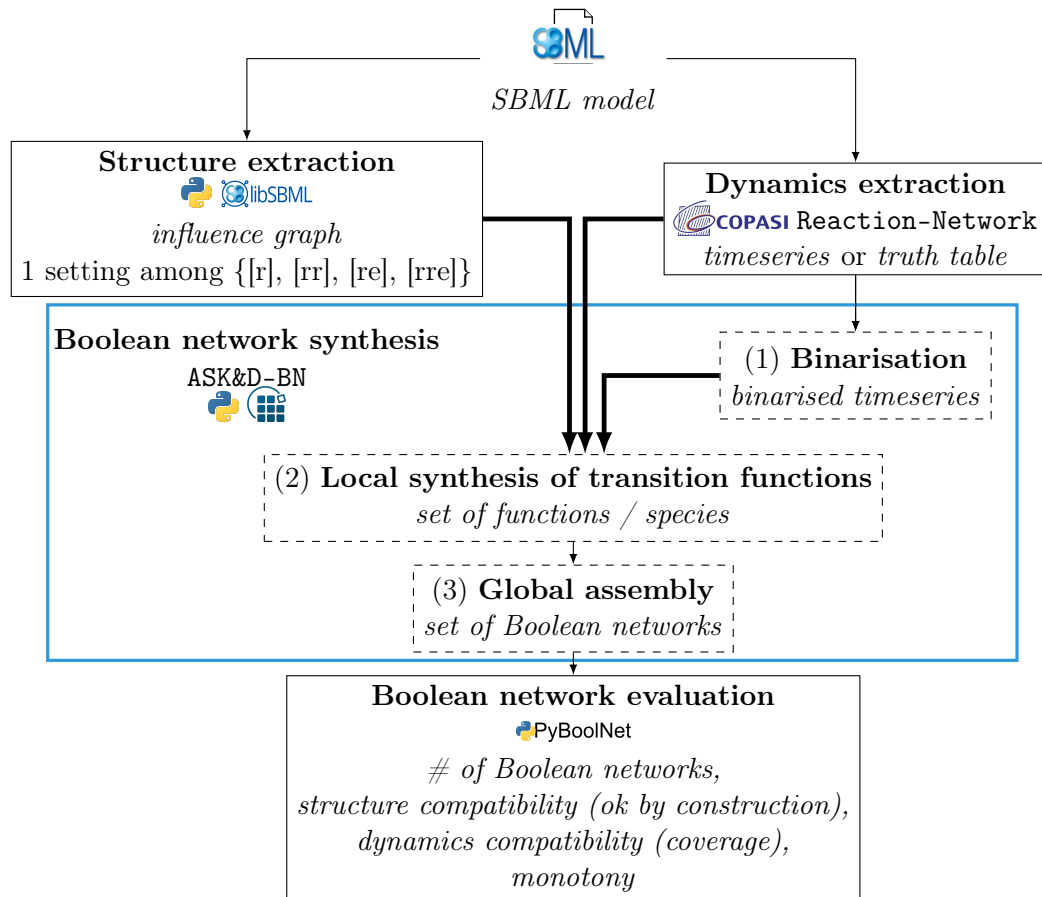


Figure C.4: Workflow of the SBML2BNET pipeline. It starts with a reaction network encoded in SBML and produces Boolean networks in the BNET format. The Boolean network synthesis step (blue box) is composed of three steps among which the local synthesis of the transition functions which uses as inputs (thick arrows): a prior influence graph, and either a truth table or a timeseries and its binarisation.

optimisation. We encode the problem in the answer-set programming framework (ASP) because it is well suited for this kind of problem. In our framework, the structure constraints are hard, and a Boolean network is considered to be compatible with a given structure if the influences that play a role in its dynamics are all allowed by the provided structure. As for the dynamics, we investigate cases where it is given as either a complete truth table, or a partial truth table, or a quantitative timeseries which undergoes a binarisation step. A Boolean network is considered to be compatible with a given dynamics if it can somehow reproduce it. That is, if its transition graph contains a good proportion of the transitions observed in the given dynamic data. The dynamics constraints are soft because it is not always the case that a Boolean network with a perfect fit exists.

The SBML2BNET pipeline is an implementation of all the methods mentioned above. It starts with a reaction network encoded in the systems biology markup language and synthesises a set of Boolean networks encoded in the BNET format. This pipeline is intended to be modular, so that several variants can be explored. We apply SBML2BNET on simple reaction networks and more complex ones, retrieved from the BioModels repository. Our evaluation demonstrates the effectiveness of the pipeline in synthesising valuable Boolean networks that comply with the input reaction network, according to our criteria. Our experiments with different settings really help to grasp several limitations of the SBML2BNET pipeline. In particular, we discussed the limitations and difficulties faced when processing models from the database BioModels, in particular those that make use of features such as discontinuous events, and rules.

Résumé Étendu

La vie ainsi que les processus à sa périphérie, sont rendus possibles grâce à des transformations bio-chimiques qui sont toutes entrelacées. Cela forme ce que l'on appelle les *systèmes* biologiques. Ces systèmes sont souvent qualifiés de *complexes*, du fait que leur dynamique ne soit pas facilement dérivée de la connaissance statique de leurs constituants en isolation [Spi04].

La compréhension globale et systématique de la dynamique de ces systèmes et des mécanismes sous-jacents est le sujet de *biologie des systèmes* [ZBH20; Bre10]. Comme pour d'autres domaines, l'outil principal de la biologie des systèmes est l'utilisation de *modèles* [Laz02; WL05]. Au sens large, un modèle est une représentation abstraite (abrégée et pratique) de la réalité (plus complexe et détaillée) [Koh+10]. Un modèle repose sur certaines hypothèses, et il peut être utilisé pour tirer des conclusions que l'on peut croire si tant est que le modèle soit correct [Gun14]. L'exactitude d'un modèle est de fait difficile à évaluer, mais elle est estimée au travers des points suivants :

1. la validité de ses hypothèses,
2. dans quelle mesure ses prédictions correspondent aux connaissances existantes et aux données expérimentales.

Les hypothèses et les prévisions des modèles concernent généralement la *structure* du système étudié ainsi que sa *dynamique* [RS02]. Le premier point concerne les composants impliqués et la manière dont ils s'influencent mutuellement, tandis que le second consiste en des motifs repérables dans la dynamique du système, tels que les configurations successives et les configurations stables, ou encore la façon dont le système réagit à certaines perturbations. Pour obtenir une

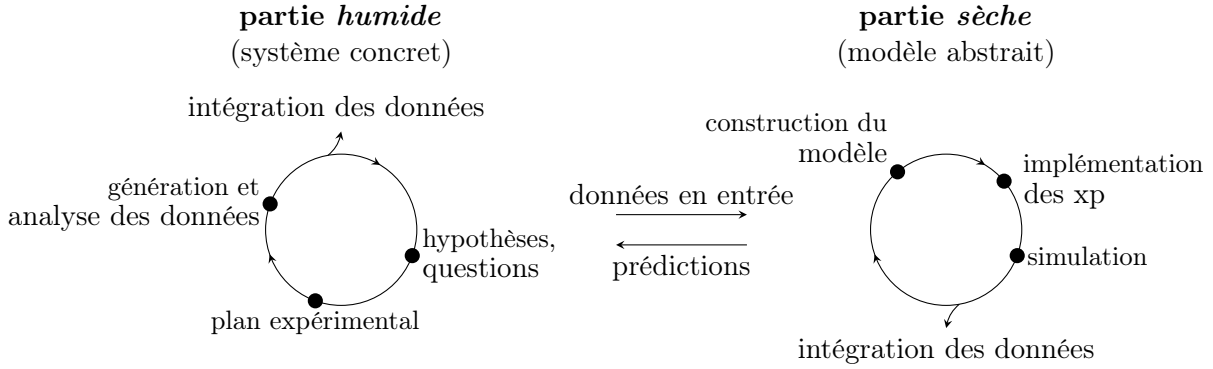


FIGURE C.5 : Consensus actuel du cycle de la recherche en biologie des systèmes, adapté de [BS20, Figure 1]. Les parties « humide » et « sèche » se composent essentiellement des mêmes étapes, et peuvent fournir des informations parallèlement l’une à l’autre.

correspondance suffisante entre le modèle et le système biologique, les modèles et les expériences sont généralement affinés par itérations de formulation d’hypothèses, de modélisation, de prédiction et d’expérimentation. C’est l’idée principale de ce que l’on appelle le cycle de la biologie des systèmes. Le consensus actuel sur ce cycle est résumé en [figure C.5](#).

Ce cycle produit des modèles dans différents formalismes. Ces derniers sont souvent classés en deux catégories : ceux basés sur les mécanismes et ceux basés sur les influences. Les modèles de ces deux classes ont des philosophies très différentes. En effet, les modèles de la première catégorie imitent les processus sous-jacents en utilisant le concept de réaction, tandis que les modèles de la seconde catégorie donnent un sens aux observations en codant les relations entre les composants du système. Les réseaux de réactions et les réseaux booléens sont les exemples typiques respectifs des modèles basés sur les mécanismes et des modèles basés sur l’influence. Les [figures C.6](#) et [C.7](#) montrent respectivement un exemple de réseau de réactions et de réseau booléen. Dans le cas des réseaux booléens, les relations sont codées par n fonctions booléennes (une par composant).



FIGURE C.6 : Un réseau de réactions (a) et sa représentation graphique (b). La réaction \mathcal{R}_1 transforme une paire formée d’une instance de A et d’une instance de B en une instance de C, et la réaction \mathcal{R}_2 transforme un C en un A.

Chaque formalisme a ses propres forces et faiblesses [Nov15]. Le choix duquel est à utiliser est guidé par la question posée : le meilleur est le plus simple, dès lors qu’il est suffisant pour répondre à la question [Bak+18; Bor05]. Dans la littérature, la plupart des modèles correspondent à des réseaux de réactions détaillés et étudiés par le biais d’équations différentielles ordinaires. Cependant, et bien que cela puisse parfois être un peu contre-intuitif, il peut être intéressant de s’orienter vers un formalisme « plus simple » (plus grossier, moins granulaire) [DB08].



FIGURE C.7 : Un réseau booléen (a) et son graphe d'influences (b).

Le défi relevé dans cette thèse est de contribuer à l'amélioration du cycle de développement des modèles. La nouveauté de l'approche réside dans le fait que nous le faisons en recyclant les connaissances et les données extraites des modèles existants, au lieu d'en produire de nouveaux par le biais d'expériences. Plus précisément, nous développons une approche pour synthétiser *automatiquement* des réseaux booléens à partir de réseaux de réactions *existants*. Nous pensons que la conversion de l'un en l'autre est pertinente, au moins pour les trois raisons suivantes :

1. D'un point de vue pratique, il s'agit là d'une solution *ad hoc* qui facilite certaines analyses difficiles à effectuer sur des modèles de réseaux de réactions. Ces analyses sont liées au contrôle ou à l'exploration des propriétés dynamiques du système.
2. D'un point de vue théorique, l'accès à une telle conversion pourrait aider à mieux comprendre la relation formelle entre les réseaux de réactions et les réseaux booléens.
3. En retour, appréhender cette conversion pourrait améliorer les méthodes de synthèse des réseaux booléens qui utilisent de vraies données biologiques. En effet, les réseaux de réactions sont très proches des systèmes biologiques sous-jacents. Dès lors, on peut considérer que la synthèse de réseaux booléens à partir de vraies données biologiques correspond indirectement à une conversion implicite d'un réseau de réactions en un réseau booléen. Par conséquent, toute leçon que l'on peut tirer d'une synthèse dans un environnement « contrôlé » (qui utilise des données extraites de modèles de réseaux de réactions existants) peut être adaptée à une synthèse dans un environnement « non contrôlé » (qui utilise des données biologiques expérimentales).

Les différentes étapes de l'approche proposée sont résumées dans la [figure C.8](#). L'approche s'inspire de ce qui est généralement fait par les méthodes existantes de synthèse de modèles. En particulier, elle est formulée comme une tâche d'optimisation de paramètres, et les données en entrée concernent la *structure* et la *dynamique* du système sous-jacent.

Pour un réseau de réactions donné, nous définissons sa structure comme un graphe qui capture les influences *directes* de ses composants. Nous pouvons récupérer un tel graphe en parsant les réactions du modèle d'entrée. En ce qui concerne la dynamique, nous proposons deux façons de l'extraire. Les deux méthodes reposent sur la simulation d'équations différentielles ordinaires (EDO) extraites du réseau de réactions. Premièrement, la dynamique qualitative d'un réseau de réaction est obtenue par *simulation abstraite*. Pour ce faire, nous construisons une formule logique du premier ordre à partir des EDO du réseau de réactions. Nous appelons cette formule un *réseau booléen du premier ordre avec mises à jour non-déterministes* car son interprétation

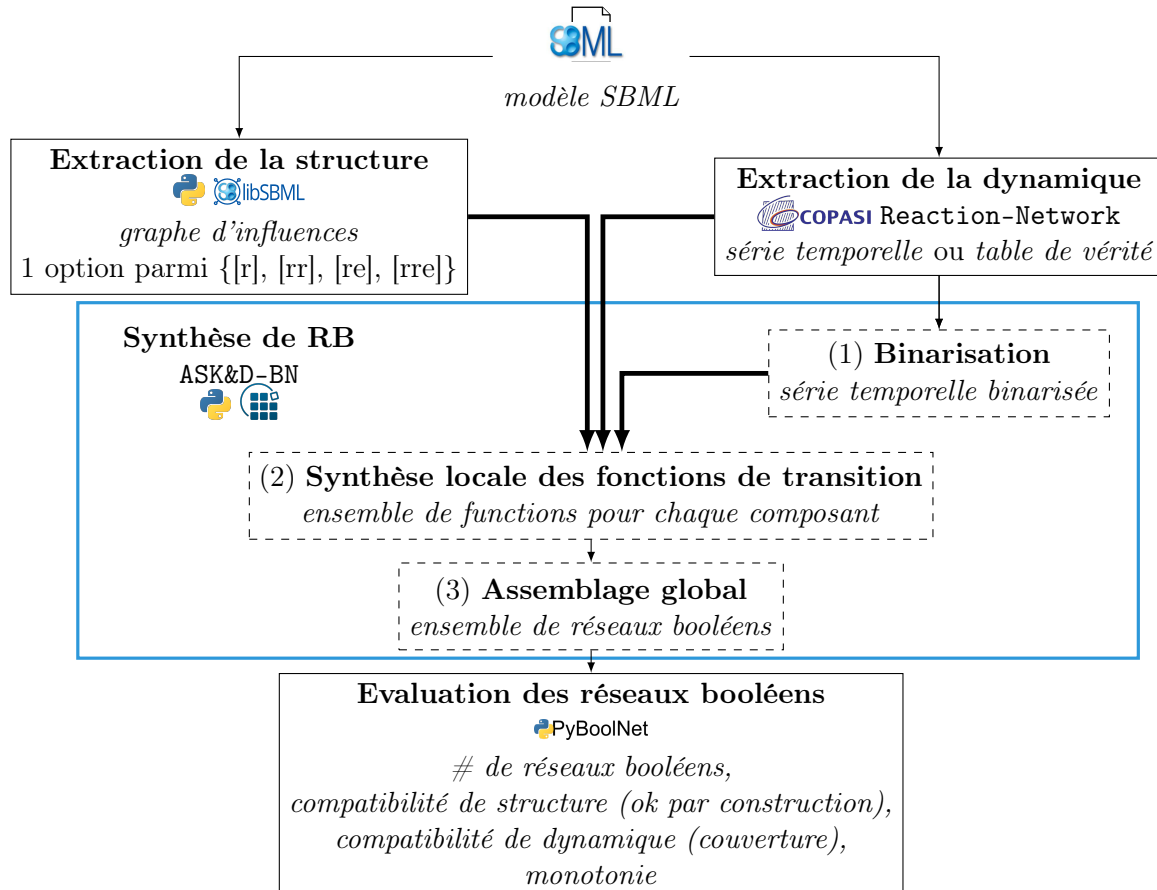


FIGURE C.8 : Déroulé du pipeline SBML2BNET. Le pipeline commence avec un réseau de réactions encodé en SBML, et produit des réseaux booléens dans le format BNET. La synthèse des réseaux booléens (en bleu) se compose de trois étapes parmi lesquelles la synthèse locale des fonctions de transition. Cette étape utilise en entrées (flèches épaisses) : un graphe d'influences a priori et soit une table de vérité, soit une série temporelle et sa binarisation.

abstraite (où la valeur précise de chaque variable est remplacée par son signe) définit une relation de transition sur les configurations booléennes du système. Cette formule logique peut donc être utilisée pour construire un graphe de transitions entre configuration booléennes. Nous montrons que ce graphe de transitions est une sur-approximation correcte vis-à-vis d’une simulation des EDO avec l’algorithme d’Euler. Deuxièmement, nous utilisons une simulation concrète des EDO, suivie d’une étape de binarisation. Cette deuxième approche peut sembler naïve à première vue, mais le problème n’est en fait pas si simple du fait que les EDO représentant des systèmes biologiques mélangent des évolutions lentes et rapides. Il nous faut donc utiliser des algorithmes de simulation malins qui font un compromis entre la précision et l’efficacité de la simulation. En outre, comme cette approche est plus directe, elle nous permet de récupérer une dynamique pour des réseaux de réactions pour lesquels la méthode de simulation abstraite n’est pas encore applicable. C’est en particulier le cas pour des réseaux de réactions disponibles dans la littérature qui sont couplés avec des événements discontinus.

Vient ensuite, l’étape de synthèse des réseaux booléens. Elle consiste à trouver *tous* les réseaux booléens *compatibles* avec une structure et une dynamique données. Comme nous l’avons mentionné plus haut, cette étape est formulée comme une tâche d’optimisation des paramètres. Plus précisément, nous considérons qu’il s’agit d’un problème de satisfaction de contraintes mêlé à une optimisation. Nous formulons le problème au moyen de la programmation par ensembles-réponses (Answer-Set Programming, ASP) qui est particulièrement bien adaptée à la tâche.

Dans notre approche, les contraintes concernant la structure sont dures : un réseau booléen n’est considéré compatible avec une structure donnée que lorsque les influences qui jouent un rôle dans sa dynamique sont toutes belles et bien autorisées par la structure fournie. En ce qui concerne la dynamique, nous considérons les cas où elle est donnée soit comme une table de vérité (complète ou partielle), soit une série temporelle (quantitative ou binarisée). Un réseau booléen est considéré compatible avec une dynamique donnée s’il peut la reproduire. En d’autres termes, si son graphe de transitions contient une bonne proportion des transitions observées dans les données dynamiques. Les contraintes de dynamique sont souples car il n’existe pas toujours de réseau booléen parfaitement compatible.

Le pipeline SBML2BNET est une implémentation de toutes les méthodes mentionnées ci-dessus. Il part d’un réseau de réactions encodé dans le langage SBML, et synthétise un ensemble de réseaux booléens encodés dans le format BNET. Ce pipeline se veut modulaire, de sorte que plusieurs variantes peuvent être explorées. Nous l’appliquons sur des réseaux de réactions simples et sur des réseaux plus complexes extraits de la base de données BioModels. Notre évaluation démontre l’efficacité du pipeline pour synthétiser des réseaux booléens qui, selon nos critères, est compatible au réseau de réactions d’entrée. Nos expériences nous aident vraiment à comprendre les limites des différentes variantes du pipeline. En particulier, nous discutons des limites et des difficultés rencontrées lors du traitement des modèles issus de la base de données BioModels, notamment pour ceux qui utilisent des caractéristiques telles que les événements discontinus et les règles.

References

- [Bak+18] R. E. Baker, J.-M. Peña, J. Jayamohan, and A. Jérusalem. “Mechanistic Models versus Machine Learning, a Fight Worth Fighting for the Biological Community?” In: *Biology Letters* 14.5 (2018), p. 20170660. DOI: [10.1098/rsbl.2017.0660](https://doi.org/10.1098/rsbl.2017.0660) (cit. on pp. 182, 186).
- [BS20] M. Banwarth-Kuhn and S. Sindi. “How and Why to Build a Mathematical Model: A Case Study Using Prion Aggregation”. In: *Journal of Biological Chemistry* 295.15 (2020), pp. 5022–5035. DOI: [10.1074/jbc.REV119.009851](https://doi.org/10.1074/jbc.REV119.009851) (cit. on pp. 181, 186).
- [Bor05] S. Bornholdt. “Less Is More in Modeling Large Genetic Networks”. In: *Science* 310.5747 (2005), pp. 449–451 (cit. on pp. 182, 186).
- [Bre10] R. Breitling. “What Is Systems Biology?” In: *Frontiers in Physiology* 1 (2010). ISSN: 1664-042X (cit. on pp. 181, 185).
- [DB08] M. Davidich and S. Bornholdt. “The Transition from Differential Equations to Boolean Networks: A Case Study in Simplifying a Regulatory Network Model”. In: *Journal of Theoretical Biology* 255.3 (2008), pp. 269–277. DOI: [10.1016/j.jtbi.2008.07.020](https://doi.org/10.1016/j.jtbi.2008.07.020) (cit. on pp. 182, 186).
- [Gun14] J. Gunawardena. “Models in Biology: ‘Accurate Descriptions of Our Pathetic Thinking’”. In: *BMC Biology* 12.1 (2014), p. 29. DOI: [10.1186/1741-7007-12-29](https://doi.org/10.1186/1741-7007-12-29) (cit. on pp. 181, 185).
- [Koh+10] P. Kohl, E. J. Crampin, T. A. Quinn, and D. Noble. “Systems Biology: An Approach”. In: *Clinical Pharmacology & Therapeutics* 88.1 (2010), pp. 25–33. DOI: [10.1038/clpt.2010.92](https://doi.org/10.1038/clpt.2010.92) (cit. on pp. 181, 185).
- [Laz02] Y. Lazebnik. “Can a Biologist Fix a Radio?—Or, What I Learned While Studying Apoptosis”. In: *Cancer Cell* 2.3 (2002), pp. 179–182. DOI: [10.1016/S1535-6108\(02\)00133-2](https://doi.org/10.1016/S1535-6108(02)00133-2) (cit. on pp. 181, 185).
- [Nov15] N. Le Novère. “Quantitative and Logic Modelling of Molecular and Gene Networks”. In: *Nature Reviews Genetics* 16.3 (2015), pp. 146–158. DOI: [10.1038/nrg3885](https://doi.org/10.1038/nrg3885) (cit. on pp. 182, 186).
- [RS02] A. Regev and E. Shapiro. “Cellular Abstractions: Cells as Computation”. In: *Nature* 419.6905 (2002), pp. 343–343. DOI: [10.1038/419343a](https://doi.org/10.1038/419343a) (cit. on pp. 182, 185).
- [Spi04] A. Spivey. “Systems Biology: The Big Picture”. In: *Environmental Health Perspectives* 112.16 (2004), A938–A943. ISSN: 0091-6765 (cit. on pp. 181, 185).
- [WL05] J. C. Wooley and H. S. Lin. *Computational Modeling and Simulation as Enablers for Biological Discovery*. National Academies Press (US), 2005. URL: <https://www.ncbi.nlm.nih.gov/books/NBK25466/> (cit. on pp. 181, 185).
- [ZBH20] A. Zupanec, H. C. Bernstein, and I. Heiland. “Systems Biology: Current Status and Challenges”. In: *Cellular and Molecular Life Sciences* 77.3 (2020), pp. 379–380. DOI: [10.1007/s00018-019-03410-z](https://doi.org/10.1007/s00018-019-03410-z) (cit. on pp. 181, 185).

Résumé

Synthèse de réseaux booléens à partir de la structure et de la dynamique de réseaux de réactions

La conversion entre différents formalismes de modélisation est un défi majeur en biologie des systèmes, car elle aide à obtenir de nouveaux éclairages sur les systèmes biologiques. Cette thèse propose une approche pour synthétiser automatiquement des réseaux booléens à partir de réseaux de réactions existants, permettant la conversion entre deux formalismes très utilisés. La thèse se compose de trois contributions. La première contribution concerne l'extraction de la structure et de la dynamique d'un réseau de réactions, c'est-à-dire : un graphe qui encode les potentielles influences directes entre les composants du système, et soit une série temporelle (binarisée), soit un graphe de transitions (potentiellement partiel). La deuxième contribution est une approche basée sur la programmation d'ensembles-réponses pour synthétiser des réseaux booléens conformes à la structure et à la dynamique extraites. La troisième contribution concerne l'évaluation du pipeline sur des exemples jouets ainsi que sur des réseaux de réactions issus de la base de données BioModels. Ces modèles sont encodés dans le langage SBML. Nos résultats démontrent l'efficacité de notre approche pour générer des réseaux booléens pertinents à partir de réseaux de réactions.

Mots-clés : biologie des systèmes, synthèse de modèle, réseau booléen, réseau de réactions, équations différentielles, programmation par ensembles-réponses, SBML

Abstract

Synthesis of Boolean networks from the structure and dynamics of reaction networks

Conversion between modelling formalisms is a critical challenge in systems biology, as it helps to get new insights into complex biological systems. This thesis proposes an approach to automatically synthesise Boolean networks from existing reaction networks, enabling the conversion between two widely used formalisms. The thesis consists of three contributions. First, we introduce a method for extracting the structure and dynamics of a reaction network. That is, a graph that encodes the potential direct influences between the components, and either a (binarised) timeseries or a (partial) transition graph. Second, we present an answer-set programming-based approach to synthesise Boolean networks that comply with the extracted structure and dynamics. Finally, we evaluate the resulting pipeline on both toy examples and reaction networks from the BioModels repository, encoded in the system biology markup language. Our results demonstrate the effectiveness of our approach in synthesising relevant Boolean networks from reaction networks.

Keywords: systems biology, model synthesis, Boolean network, reaction network, differential equations, answer-set programming, SBML