Diploma Project

EXPERIMENTAL EVALUATION OF BOOLEAN NETWORK LEARNING SYSTEMS

Costas Frantzides

University of Cyprus
Department of Computer Science

Department of Computer Science

May 2026

UNIVERSITY OF CYPRUS

Faculty of Pure and Applied Sciences

Department of Computer Science

EXPERIMENTAL EVALUATION OF BOOLEAN NETWORK LEARNING SYSTEMS

Costas Frantzides

Advisor: Dr. Yiannis Dimopoulos

The Thesis was submitted in partial fulfillment of the requirements for obtaining the Computer Science degree of the Department of Computer Science of the University of Cyprus.

I declare that this dissertation and any accompanying software are my own original work, conducted in full compliance with the University of Cyprus Code of Conduct for Research (https://www.ucy.ac.cy/legislation/kodikas-deontologias-kai-politikes/), and with full accountability on my part for the accuracy, integrity, and validity of all content.

May 2026

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Yiannis Dimopoulos, for his valuable guidance, support, and advice throughout the preparation of this diploma thesis.

His scientific contribution was essential to the successful completion of this work. I would also like to warmly thank the professors and staff of the Department for their continuous support and for the knowledge they provided during my studies.

Finally, I would like to express my deepest gratitude to my family and friends for their constant support, encouragement, and understanding throughout my academic journey.

ABSTRACT

Boolean networks offer a computationally tractable yet expressive framework for modelling complex dynamical systems, with particular relevance to systems biology, medicine, and engineering. This thesis presents a comprehensive study of Boolean network modelling and inference, serving simultaneously as a practical guide for prospective modellers and as a research-oriented contribution to the inference problem.

The work begins by surveying real-world applications of Boolean networks across gene regulatory systems, signalling pathways, cell fate decisions, cancer biology, digital circuit design, and ecological dynamics. It then establishes the theoretical foundations including update schemes, transition graphs, attractors, and structural properties of Boolean functions before reviewing the principal computational tools available for network construction, simulation, and analysis, such as BoNesis, BoolNet, BoolForge, bioLQM, and the Sketches framework.

A dedicated chapter introduces temporal logic covering Linear Temporal Logic (LTL), Computation Tree Logic (CTL), and Hybrid CTL (HCTL) and explains how these formalisms enable precise specification of dynamical properties for network reconstruction. Building on this, the thesis describes the Boolean network Sketches framework in detail, encompassing influence graphs, partially specified update functions, functional constraints, and HCTL encoded dynamic properties.

The central technical contribution is a configurable, multi-stage pipeline for generating synthetic Boolean network sketches under controlled structural, functional, and dynamical assumptions. The pipeline systematically varies the type and amount of prior knowledge encoded in each sketch, enabling rigorous experiments on the inference problem. Empirical evaluation reveals the relative importance of different knowledge categories structural, functional, and dynamic for achieving feasible and accurate network reconstruction, and characterises the scalability trade-offs involved.

Keywords: Boolean Networks, Systems Biology, Network Inference, Temporal Logic, Sketches Framework, Computational Tools, HCTL

TABLE OF CONTENTS

ABSTRACT iii

TABLE OF CONTENTS iv

LIST OF TABLES viii

LIST OF FIGURES ix

1 Introduction 1

1.1 Purpose and Objectives . . . 2
1.2 Motivation and Significance . . . 2
1.3 Structure of the Thesis . . . 3

2 Applications of Boolean Networks 5

2.1 Systems Biology . . . 6
2.1.1 Foundations: Kauffman's Random Boolean Networks . . . 6
2.1.2 Gene Regulatory Networks . . . 7
2.1.2.1 Segment polarity network in *Drosophila* . . . 7
2.1.2.2 Flower development in *Arabidopsis thaliana* . . . 8
2.1.2.3 T-LGL leukaemia survival network . . . 9
2.1.3 Signalling Pathways . . . 10
2.1.3.1 T-cell receptor signalling . . . 10
2.1.3.2 EGFR signalling . . . 10
2.1.4 Cell Fate Decisions and Differentiation . . . 11
2.1.4.1 T-helper cell differentiation . . . 11
2.1.4.2 Epithelial–mesenchymal transition . . . 11
2.1.5 Boolean Network Inference from Biological Data . . . 12
2.2 Medicine and Therapeutics . . . 12
2.2.1 Cancer . . . 13
2.2.2 Immunology . . . 13
2.2.3 Drug Target Identification . . . 13

2.3 Engineering and Computer Science . . . 14
2.3.1 Digital Circuit Analysis and Design . . . 14
2.3.2 Network Controllability and Control Theory . . . 14
2.3.3 Fault Detection and Reliability . . . 15
2.3.4 Constraint Solving and Formal Verification . . . 15
2.4 Ecology and Evolutionary Biology . . . 15
2.4.1 Ecological Community Dynamics . . . 15
2.4.2 Evolution of Gene Regulatory Networks . . . 16

3 Boolean Networks **17**
3.1 Definition of Boolean Networks . . . 18
3.2 Synchronous and Asynchronous Update Schemes . . . 18
3.2.1 Synchronous Update . . . 19
3.2.2 Asynchronous Update . . . 19
3.2.3 Diagram of Update Schemes . . . 20
3.3 Transition Graph . . . 20
3.4 Attractors . . . 21
3.5 Structural Properties of Boolean Functions . . . 21
3.5.1 Essentiality . . . 22
3.5.2 Monotonicity . . . 22
3.5.3 Canalization . . . 23
3.6 Computational Tools for Boolean Network Analysis and Synthesis . . . 23
3.6.1 BoNesis . . . 23
3.6.1.1 Boolean Network Reconstruction Using BoNesis . . . 25
3.6.1.2 Influence Graph Construction . . . 25
3.6.1.3 Problem Initialisation . . . 25
3.6.1.4 Encoding Temporal Constraints . . . 25
3.6.1.5 Solution Enumeration . . . 26
3.6.1.6 Reconstruction Experiment . . . 26
3.6.2 BoolNet . . . 31
3.6.3 BoolForge . . . 33
3.6.3.1 Network Construction and Representation . . . 33
3.6.3.2 Dynamics Simulation and Attractor Analysis . . . 34
3.6.3.3 Rule Editing, Perturbations, and Model Composition . . . 35
3.6.4 bioLQM . . . 35
3.6.4.1 Architecture and Interoperability . . . 36
3.6.4.2 Fixed Points and Trap Spaces . . . 36

3.6.4.3 Programmatic Usage in Computational Pipelines 37
3.6.4.4 Model Transformations and Perturbations 38
3.6.5 Sketches 39

4 Temporal Logic for Boolean Networks **40**
4.1 Motivation: Why Temporal Logic for Boolean Networks 41
4.2 Foundations of Temporal Logic 41
4.2.1 States and Transition Systems 41
4.2.2 Paths and Executions 42
4.2.3 Temporal Operators 42
4.2.3.1 Linear Temporal Logic (LTL) 43
4.2.3.2 Computation Tree Logic (CTL) 43
4.3 Expressing Boolean Network Properties in Temporal Logic 44
4.3.1 Reachability 44
4.3.2 Stability (Fixed Points) 45
4.3.3 Cyclic Behavior and Attractors 45
4.3.4 Safety and Liveness Properties 45
4.4 Temporal Logic in Real Applications 46
4.4.1 Modeling Real-World Behaviors 46
4.4.2 Examples from Applications 46
4.4.3 Interpretability of Temporal Specifications 46

5 Boolean Network Sketches **47**
5.1 Structure of a Boolean Network Sketch 49
5.1.1 Influence Graph (I) 49
5.1.2 Partially Specified Boolean Network (E) 50
5.1.3 Update Function Properties (Π) 51
5.1.4 Dynamic Properties (Ω) 53
5.2 Specification Language: HCTL 54
5.2.1 Syntax and Basic Operators 54
5.2.2 Temporal Operators 55
5.2.3 Hybrid Operators 55
5.2.4 Encoding Biological Observations 56
5.3 Capabilities and Limitations 57

6 Pipeline Overview **59**
6.1 Main Objective 60
6.2 Design and Architecture 61

6.3 Pipeline Stages . . . 62
6.3.1 Stage 1: Boolean Network Generation . . . 63
6.3.2 Stage 2: Trace Simulation . . . 64
6.3.3 Stage 3: Trace Conversion to Dynamic Properties . . . 65
6.3.4 Stage 4: Structural Sketch Construction . . . 66
6.3.5 Stage 5: Analysis of Long-Term Dynamical Behaviour . . . 69
6.3.6 Stage 6: Conversion of Dynamical Analyses to Sketch Properties . . . 69
6.3.7 Stage 7: Sketch Assembly . . . 72
6.4 Pipeline Capabilities and Configuration . . . 75

7 Experimental Setup, Results, and Evaluation **76**
7.1 Experimental Setup . . . 77
7.1.1 Research Objectives . . . 77
7.1.2 Construction of Experimental Sketch Families . . . 78
7.1.3 Experimental Dimensions . . . 78
7.1.4 Inference Procedure . . . 79
7.1.5 Evaluation Metrics . . . 79
7.2 Experimental Results . . . 80
7.3 Evaluation . . . 82
7.3.1 Effect of Structural and Functional Information . . . 82
7.3.2 Impact of Dynamic Constraints . . . 82
7.3.3 Influence of Exact Function Revelation . . . 83
7.3.4 Scalability and Overall Trade-Off . . . 83

8 Future Work **84**
8.1 Potential Extensions . . . 84
8.2 Open Research Directions . . . 85

BIBLIOGRAPHY **87**

LIST OF TABLES

3.1 Summary of experimental parameters. 28

6.1 Tools used in the experimental pipeline 62
6.2 Main configuration options used in Stage 1 (`configs/boolforge.yaml`). . . . 64
6.3 Main configuration options used in Stage 2 (`configs/traces.txt`). 65
6.4 Main configuration options used in Stage 3 (`configs/trace_properties.txt`). 66
6.5 Main configuration options used in Stage 4 (`configs/structure.txt`). . . . 68
6.6 Main configuration options used in Stage 5 (`configs/dynamics.txt`). 69
6.7 Main configuration options used in Stage 6 (`configs/dynamics_properties.txt`). 72
6.8 Main configuration options used in Stage 7 (`configs/pipeline.txt`). 74

7.1 Qualitative summary of the observed behaviour of the six sketch classes. 82

LIST OF FIGURES

2.1 Possible pathways of gene expression during the transition from vegetative growth to flower development in *Arabidopsis thaliana*, adapted from [1]. The diagram illustrates how a common initial gene-expression state can evolve into different developmental trajectories, leading to the formation of distinct floral organs (sepals, petals, stamens, and carpels). Each node represents a Boolean state of gene activity, encoded as a vector of binary values corresponding to key regulatory genes. Bracketed states denote steady states (attractors). Solid arrows indicate single-gene state changes, while dotted arrows represent simultaneous changes in multiple genes. The transition from vegetative to floral development is triggered by an external induction signal 9

3.1 Comparison of synchronous (solid) and asynchronous (dashed) updates from state $(0, 0)$. 20

3.2 This figure illustrates the concept of attractors and their corresponding basins of attraction in a Boolean network. Each node represents a state of the system, and directed edges represent transitions between states. The highlighted cycles correspond to attractors, while the surrounding states form their basins of attraction, i.e., all states that eventually evolve into the attractor. This visual representation emphasizes how different initial conditions may lead to different long-term behaviors of the system. (adapted from [2]). 21

3.3 Overview of the BoNesis framework (adapted from [3]). The input consists of a domain of Boolean networks (e.g., a single network, a set of networks, or partially specified interaction graphs) together with declarative specifications of desired dynamical properties expressed in a Python-based language. These specifications may include constraints on fixed points, attractors, mutations, and observational consistency. BoNesis then performs constraint solving to determine satisfiability and to enumerate Boolean networks and variable assignments that satisfy the given properties. The output may include valid network models, configurations, attractors, and predictions about system behavior. 24

3.4 Mean Boolean function reconstruction accuracy as a function of network size for each level of data availability. Error bars represent standard error of the mean across replicates. 28
3.5 Heatmap of mean function accuracy across all combinations of network size and data availability. Greener cells indicate conditions under which reconstruction is most reliable. 29
3.6 Mean reconstruction time as a function of network size (log scale), aggregated across all data-fraction conditions and replicates. 29
3.7 Summary statistics of the reconstruction experiment. **(a)** Accuracy consistency per data fraction. **(b)** Cumulative distribution of function accuracy. **(c)** Reconstruction time versus accuracy, coloured by network size. 30

4.1 Illustration of Computation Tree Logic (CTL) semantics, showing branching-time paths and temporal operators. Reproduced from [4]. 44

5.1 Workflow of the Sketches inference method (reproduced from [5]). The initial sketch encodes literature-based prior knowledge. Binarised experimental data are encoded as HCTL formulae and added to yield the data-informed sketch. The inference procedure then computes all consistent candidate Boolean networks. 49
5.2 Running example of a Boolean network sketch (reproduced from [5], Figure 2). The four panels show, from left to right: the influence graph I, the partially specified Boolean network E, the update function properties Π, and the dynamic properties Ω encoded as HCTL formulae derived from prior knowledge and binarised experimental data. 54

6.1 Architecture of the sketch generation pipeline. Solid arrows denote the primary data flow; dashed arrows indicate tool invocations and the direct reuse of the `.bnet` file by Stage 4. The final `.aeon` sketch is assembled in Stage 7 from the structural model part (Stage 4) and the two families of dynamic properties (Stages 3 and 6). Which property families are included is controlled by flags in `configs/pipeline.txt`. 63

7.1 Inference runtime versus candidate space size across all 360 sketches, shown on logarithmic scales and grouped by sketch class. Sketches with only structural and functional constraints remain concentrated in the region of large candidate spaces and relatively low runtimes, whereas sketches enriched with dynamic constraints produce substantially smaller candidate spaces at a noticeably higher computational cost. 80

Chapter 1

Introduction

1.1 Purpose and Objectives **2**
1.2 Motivation and Significance **2**
1.3 Structure of the Thesis **3**

The analysis and modelling of complex biological regulatory systems remains one of the central challenges in computational biology. Gene regulatory networks, signalling pathways, and cell-fate decision circuits involve many interacting components whose individual behaviours are often known only qualitatively: a gene is active or inactive, a protein is phosphorylated or not, a receptor is engaged or idle. In such settings, Boolean networks provide a natural and mathematically rigorous modelling framework. By representing each component as a binary variable and encoding regulatory interactions as logical update functions, Boolean networks can capture important dynamical phenomena including stable states, oscillatory cycles, and transitions between distinct cell fates without requiring detailed kinetic parameters that are rarely available in practice.

Despite their conceptual simplicity, Boolean networks give rise to rich and computationally challenging inference problems. Given incomplete or partial knowledge about a biological system, one wishes to identify all Boolean networks that are consistent with the available structural, functional, and dynamical information. This problem is NP-hard in general, and the number of candidate networks consistent with even modest amounts of knowledge can be astronomically large. At the same time, the form of the available prior knowledge is heterogeneous: it may include known regulatory interactions from the literature, signed edge labels indicating activation or inhibition, functional constraints such as monotonicity or canalization, steady-state gene expression measurements, and time-series observations of system trajectories. Integrating these diverse information sources into a single coherent inference framework is both theoretically and practically demanding.

1.1 Purpose and Objectives

The purpose of this thesis is threefold. The first goal is to establish a clear and self-contained account of Boolean networks as a formal modelling framework, covering their dynamical properties, structural characteristics, and the computational tools available for their analysis and synthesis. The second goal is to connect this framework to formal specification languages, in particular temporal logic, which provides a rigorous vocabulary for describing properties that evolve over time such as reachability, fixed-point stability, and cyclic behaviour and whose operators map directly onto the dynamical concepts used throughout Boolean network analysis.

The third and central goal is the design and evaluation of a configurable pipeline for the systematic construction of Boolean network sketches suitable for inference with the Sketches framework [5]. A sketch is a formal template that encodes partial prior knowledge about a Boolean network its topology, the signs of regulatory interactions, functional constraints on update rules, and dynamical properties derived from observations or known behaviour while leaving uncharacterised components as symbolic placeholders to be resolved by constraint solving. The pipeline developed in this thesis automates the construction of such sketches from a fully specified source network, enabling controlled variation of both the amount and the type of information included in the final sketch.

Concretely, the pipeline supports the integration of three complementary classes of information. Structural information concerns the regulatory topology and update logic of the source network and is encoded in the sketch through selective revelation of variable dependencies and exact update rules. Functional information concerns properties of the update functions themselves, such as the monotonicity of regulatory edges, the essentiality of individual regulators, and canalizing relationships in which a single input can override all others. Dynamical information concerns the long-term and transient behaviour of the network and is encoded either as trace-derived reachability properties obtained by simulation or as exact attractor-related constraints computed by formal analysis. By varying the type and amount of information across these three dimensions, the pipeline enables a systematic empirical study of how different forms of prior knowledge affect the Boolean network inference problem.

1.2 Motivation and Significance

The study of Boolean network inference has long been recognised as a fundamental challenge in computational systems biology [2, 6]. From a theoretical perspective, the problem is difficult because the space of candidate networks grows exponentially with the number of variables, and even small changes in the available information can dramatically alter the size and structure of the consistent candidate set. From a practical perspective, the problem is important because

reliable inference is a prerequisite for using Boolean models to understand disease mechanisms, identify drug targets, and predict the effect of regulatory perturbations.

Existing approaches to Boolean network inference tend to fall into two categories. Reverse modelling methods start from experimental observations time-series data, steady-state measurements, or perturbation responses and search for network structures that best explain those observations [7, 8]. Forward modelling methods start from literature-based knowledge of known or hypothesised interactions and construct a model from first principles [9, 10]. Neither approach alone is sufficient: reverse methods are limited by data volume, noise, and identifiability, while forward methods accumulate inaccuracies when many uncertain assumptions are combined.

The Sketches framework [5] addresses this gap by providing a unified formal language in which structural knowledge, functional hypotheses, and experimental data can all be expressed within a single model and processed in a single inference step. Rather than committing to one candidate network, the framework computes the exhaustive set of all Boolean networks consistent with the sketch constraints, enabling downstream analyses that carry formal guarantees. The significance of the pipeline developed in this thesis lies in making this framework accessible and experimentally useful: by automating the construction of sketches under fully configurable conditions, it becomes possible to study how the inference problem changes as different information classes are added, removed, or combined, and to evaluate the framework on benchmark families of sketches with systematically controlled informativeness.

1.3 Structure of the Thesis

The remainder of this thesis is organised as follows.

Chapter 2 surveys the main application domains of Boolean networks, including systems biology, medicine and therapeutics, engineering and computer science, and ecology. This survey motivates the modelling framework adopted throughout the thesis and illustrates the breadth of contexts in which Boolean network inference is relevant.

Chapter 3 introduces the theoretical foundations of Boolean networks. It covers the formal definition of Boolean networks, synchronous and asynchronous update schemes, the transition graph, attractors and basins of attraction, and the structural properties of Boolean update functions essentiality, monotonicity, and canalization that play a central role in the inference framework. The chapter also surveys the computational tools used in this work: BoNesis, BoolNet, BoolForge, bioLQM, and the Sketches tool.

Chapter 4 provides a self-contained introduction to temporal logic and its role in the specification of Boolean network properties. It presents the foundations of Linear Temporal Logic and Computation Tree Logic, explains how dynamical properties such as reachability, stability, and

cyclic behaviour are expressed as temporal formulae, and discusses the use of temporal logic in real biological applications.

Chapter 5 presents the Boolean Network Sketches framework in detail. It describes the four components of a sketch the influence graph, the partially specified Boolean network, the update function properties, and the dynamic properties and explains how each component encodes a distinct form of prior knowledge. It also introduces the Hybrid Computation Tree Logic (HCTL) specification language used to express dynamic properties, including its hybrid operators and the encoding patterns used for biological observations.

Chapter 6 describes the seven-stage pipeline developed in this thesis. It explains the purpose and design of the pipeline, characterises each processing stage and its configuration options, and discusses how the pipeline's two main dimensions of configurability structural revelation and constraint enrichment define the space of inference-ready sketches that can be generated.

Chapter 7 presents the experimental setup and results. It describes the construction of the six sketch families used for evaluation, the experimental dimensions explored, and the inference outcomes with respect to runtime, candidate-space size, and the comparative contribution of each information class.

Finally, Chapter 8 outlines potential extensions of the pipeline and discusses open research directions in Boolean network inference and sketch-based modelling.

Chapter 2

Applications of Boolean Networks

2.1 Systems Biology **6**
2.1.1 Foundations: Kauffman's Random Boolean Networks 6
2.1.2 Gene Regulatory Networks 7
2.1.2.1 Segment polarity network in Drosophila 7
2.1.2.2 Flower development in Arabidopsis thaliana 8
2.1.2.3 T-LGL leukaemia survival network 9
2.1.3 Signalling Pathways 10
2.1.3.1 T-cell receptor signalling 10
2.1.3.2 EGFR signalling 10
2.1.4 Cell Fate Decisions and Differentiation 11
2.1.4.1 T-helper cell differentiation 11
2.1.4.2 Epithelial–mesenchymal transition 11
2.1.5 Boolean Network Inference from Biological Data 12
2.2 Medicine and Therapeutics **12**
2.2.1 Cancer 13
2.2.2 Immunology 13
2.2.3 Drug Target Identification 13
2.3 Engineering and Computer Science **14**
2.3.1 Digital Circuit Analysis and Design 14
2.3.2 Network Controllability and Control Theory 14
2.3.3 Fault Detection and Reliability 15
2.3.4 Constraint Solving and Formal Verification 15
2.4 Ecology and Evolutionary Biology **15**
2.4.1 Ecological Community Dynamics 15
2.4.2 Evolution of Gene Regulatory Networks 16

This chapter surveys the major application domains in which Boolean networks have been used to represent, analyse, and reason about complex systems. Boolean networks are especially useful in situations where the exact numerical parameters of a system are unknown, but the main qualitative interactions are understood. In such cases, it is often enough to know whether a component is active or inactive, present or absent, on or off. Despite this simplified representation, Boolean networks are still able to capture important dynamical behaviours such as stable states, oscillations, switching between modes of behaviour, and sensitivity to perturbations. For this reason, they have been used in biology, medicine, engineering, computer science, and ecology.

2.1 Systems Biology

Boolean networks have been extensively used in systems biology as a qualitative modelling framework for understanding complex biological systems, especially gene regulatory networks and signalling pathways [2, 9, 11, 12]. In such models, each variable usually represents a biological entity such as a gene, protein, or molecule, and its Boolean value indicates whether that entity is active or inactive.

The appeal of Boolean modelling in biology comes from the fact that many biological systems are known in terms of who influences whom, but not in terms of exact kinetic constants or concentrations. For example, biologists may know that one protein activates another, or that one gene suppresses another, without knowing the precise reaction speed or threshold values involved. Boolean networks provide a natural way to encode this kind of knowledge.

2.1.1 Foundations: Kauffman's Random Boolean Networks

One of the earliest and most influential uses of Boolean networks in biology was introduced by Kauffman [11], who studied random Boolean networks as abstract models of genetic regulation. The idea was not to describe one specific organism in detail, but to understand what kinds of global behaviour can emerge when many genes influence one another through simple logical rules.

In a random Boolean network, there are N genes, and each gene receives inputs from K other genes chosen at random. Each gene is also assigned a Boolean function that determines whether it will be active or inactive at the next time step based on the current states of its inputs. Even though these connections and rules are chosen randomly, the resulting network does not behave arbitrarily. Instead, Kauffman showed that the overall behaviour depends strongly on the value of K.

When $K = 1$, each gene depends on only one other gene. In this case, the network tends to be highly ordered: most trajectories quickly settle into simple stable patterns, and perturbations

usually die out. When $K = 2$, the network lies near a critical regime: changes can propagate, but not so strongly that the whole system becomes unstable. When $K \geq 3$, the network tends to become chaotic: small perturbations spread widely, attractors become long and complex, and the system becomes much harder to predict [11].

The importance of this result is conceptual. Kauffman suggested that real biological systems may operate near this critical regime, where they are neither completely rigid nor completely chaotic. Such a regime would allow cells to be stable enough to maintain identity, yet flexible enough to respond to signals and environmental changes. He further proposed that attractors in these networks can be interpreted as distinct cell types, that is, stable patterns of gene activity corresponding to biologically meaningful cellular states [11].

Thomas complemented this line of work by introducing a more biologically structured logical view of regulatory circuits [12]. Rather than focusing on randomly generated networks, Thomas studied regulatory interactions with explicit signs, distinguishing activation from inhibition. He argued that positive feedback loops are necessary for the existence of multiple stable states, while negative feedback loops are necessary for sustained oscillations. This insight became foundational in later biologically motivated Boolean models, because it connected network topology to dynamical behaviour in an interpretable way.

2.1.2 Gene Regulatory Networks

A gene regulatory network describes how genes control one another's expression. In biological terms, some genes help activate other genes, while others suppress them. A Boolean network provides a simplified version of this process: each gene is represented as either active or inactive, and each Boolean rule describes how the activity of one gene depends on the activity of other genes.

This simplification is powerful because many biological questions do not require exact concentration values. Often, what matters is whether a gene is effectively turned on or off, and whether the system eventually reaches a stable pattern.

2.1.2.1 Segment polarity network in Drosophila

A classical example is the segment polarity gene network in *Drosophila melanogaster*, modelled by Albert and Othmer [9]. To understand why this problem matters, it helps to know that during early development, the embryo must divide itself into repeating body segments in a very precise way. If the underlying regulatory system fails, the body plan is disrupted.

Albert and Othmer constructed a Boolean model of a regulatory network involving genes and proteins such as *wingless*, *engrailed*, and *hedgehog*. Each variable represented whether a gene

or protein was active in a cell, and the Boolean functions encoded the known activating and inhibitory interactions among them [9].

The important achievement of this model was that, starting from biologically meaningful initial conditions, it converged to stable patterns that matched the observed segment pattern of the developing embryo. In other words, the Boolean model reproduced the correct spatial gene-expression pattern without using detailed kinetic parameters. Even more importantly, the model was robust: many nearby initial states still led to the same correct final pattern. This helped explain why embryonic development can be reliable even in the presence of noise or small perturbations.

For a non-biologist, this example can be understood as follows: the embryo is like a system that must reliably paint a striped pattern. The Boolean network encodes the logical rules that determine which stripe-related genes are active in which cells. The fact that the model reaches the correct pattern from many starting points means that the real biological system has built-in error tolerance.

2.1.2.2 Flower development in Arabidopsis thaliana

Another influential example is the Boolean model of flower development in *Arabidopsis thaliana* developed by Mendoza, Thieffry, and Alvarez-Buylla [1]. In flowering plants, different regions of the flower develop into different organs such as sepals, petals, stamens, and carpels. This happens because different combinations of regulatory genes become active in different spatial regions.

The Boolean model represents these genes as binary variables and encodes the logical rules through which they regulate one another. The resulting attractors correspond to distinct stable patterns of gene activity, and these stable patterns can be interpreted as the different cell types or organ identities found in the flower [1].

The intuitive value of this example is clear: Boolean networks allow us to treat cell differentiation as a decision-making process. A cell begins in one regulatory condition, receives inputs from its regulatory environment, and eventually settles into one of several stable identities. In the flower, these identities correspond to the organs that form in different positions. Thus, the attractors are not abstract mathematical objects only; they correspond to visible biological structures.

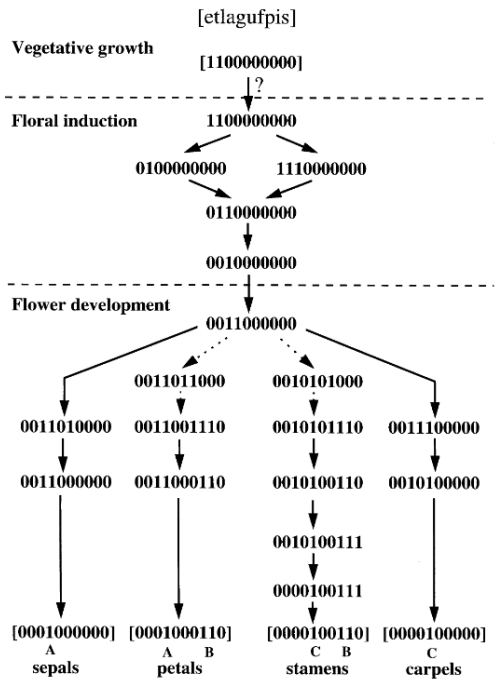


Figure 2.1: Possible pathways of gene expression during the transition from vegetative growth to flower development in *Arabidopsis thaliana*, adapted from [1]. The diagram illustrates how a common initial gene-expression state can evolve into different developmental trajectories, leading to the formation of distinct floral organs (sepals, petals, stamens, and carpels). Each node represents a Boolean state of gene activity, encoded as a vector of binary values corresponding to key regulatory genes. Bracketed states denote steady states (attractors). Solid arrows indicate single-gene state changes, while dotted arrows represent simultaneous changes in multiple genes. The transition from vegetative to floral development is triggered by an external induction signal .

2.1.2.3 T-LGL leukaemia survival network

A more medically oriented example is the Boolean model of T-cell large granular lymphocyte leukaemia developed by Zhang et al. [10]. This disease involves immune cells that fail to die when they should. Under normal conditions, activated immune cells often undergo programmed cell death, or apoptosis, after finishing their role. In T-LGL leukaemia, this death program is disrupted, and the cells survive abnormally.

Zhang et al. built a Boolean network with 18 variables representing molecules involved in the survival and death signalling of these cells. The model included proteins that promote survival and proteins that promote apoptosis, and the Boolean rules encoded how these factors influence one another [10].

The model exhibited two main attractors. One attractor corresponded to the healthy outcome in which apoptosis is active and the abnormal cell dies. The other corresponded to the disease state in which apoptosis is suppressed and survival-promoting pathways remain active. This is a good example of how attractors can be interpreted clinically: one attractor corresponds to normal behaviour, while the other corresponds to pathology.

The authors then used the model to test interventions computationally. By forcing some variables on or off, they identified perturbations that eliminated the disease attractor and redirected the dynamics toward apoptosis. In practical terms, this means that the Boolean model can be used to suggest potential drug targets. For a non-biologist, the key point is that the network acts like a map of possible cell behaviours, and changing certain nodes can shift the system from an unhealthy stable state to a healthy one.

2.1.3 Signalling Pathways

Gene regulatory networks focus mainly on how genes influence gene expression. Signalling pathways, in contrast, describe how signals are transmitted inside the cell, often through chains of protein activations and inhibitions. These pathways are essential because cells constantly need to detect signals from their environment and convert them into internal responses.

Boolean networks are useful here because many signalling events can be described in binary terms: a receptor is activated or not, a protein is phosphorylated or not, a transcription factor enters the nucleus or not.

2.1.3.1 T-cell receptor signalling

Saez-Rodriguez et al. developed a Boolean model of T-cell receptor signalling [13]. T cells are immune cells that must detect foreign antigens and respond appropriately. When a T-cell receptor is engaged, it triggers a cascade of intracellular events involving kinases, adaptor proteins, and transcription factors. The end result is a decision about whether the T cell should become activated.

In the Boolean model, each relevant molecule was represented by a binary variable, and the update functions encoded how activation propagates through the signalling cascade. The model was then compared with experimental data collected under different stimulation conditions [13].

The significance of this example is that a relatively simple logical model was able to reproduce the qualitative activation patterns observed experimentally, even though no kinetic constants were used. For a reader without a biology background, one can think of this as a logic circuit inside the cell: an external signal is received, passed through a sequence of internal logic gates, and eventually produces a cellular response.

2.1.3.2 EGFR signalling

Samaga et al. constructed a larger Boolean model of epidermal growth factor receptor signalling in a breast cancer cell line [14]. The EGFR/ErbB system is important because it controls cell growth, survival, and proliferation, and is often altered in cancer.

The network in this study contained more than one hundred nodes and described how receptor activation connects to major downstream pathways such as Ras/MAPK, PI3K/Akt, and JAK/STAT [14]. The model was trained and tested against many experimental perturbation conditions, including drug treatments.

What makes this example important is that the Boolean model not only reproduced known signalling responses, but also predicted the effect of perturbations not used during model construction. In other words, it had predictive power. For a non-specialist, this can be understood as follows: once the logical structure of the pathway is captured correctly, the model can be used like a virtual laboratory to test how the system might respond to interventions.

2.1.4 Cell Fate Decisions and Differentiation

Cells in multicellular organisms often need to choose between alternative identities. A stem-like cell may become one of several specialised cell types depending on its regulatory environment. Boolean networks are well suited for modelling this process because such decisions can be represented as transitions toward different attractors.

2.1.4.1 T-helper cell differentiation

Naldi et al. developed a Boolean model of T-helper cell differentiation [15]. Naive $CD4^+$ T cells can differentiate into several subtypes, such as Th1, Th2, Th17, or regulatory T cells, depending on the cytokine signals they receive. Each subtype has a different role in the immune system.

The model represented key transcription factors and their interactions. Different attractors corresponded to different immune cell identities [15]. In this case, the biological interpretation of attractors is especially intuitive: each attractor corresponds to a stable immune cell program.

The model also captured plasticity, meaning that under some conditions a cell can move from one subtype toward another. This is important because it shows that Boolean networks can represent not only stable identities, but also the possibility of switching between them when the regulatory context changes.

2.1.4.2 Epithelial–mesenchymal transition

Lu et al. modelled the epithelial–mesenchymal transition (EMT) using a Boolean network [16]. EMT is a process in which a cell changes from an epithelial state, where it is relatively fixed in place and strongly connected to neighbouring cells, to a mesenchymal state, where it becomes more mobile and invasive. This process is important both in normal development and in cancer metastasis.

The Boolean model focused on a small core regulatory circuit involving factors such as miR-200,

ZEB, and SNAIL [16]. Under asynchronous dynamics, the model showed not only epithelial and mesenchymal attractors, but also a third, hybrid attractor corresponding to a partially epithelial and partially mesenchymal state.

This example is especially useful for non-biologists because it illustrates how Boolean models can reveal intermediate states that are not obvious from simple binary thinking. Even though each individual variable is binary, the network as a whole can still exhibit several qualitatively distinct stable behaviours. The hybrid state is biologically important because it has been associated with high metastatic potential and stem-cell-like properties.

2.1.5 Boolean Network Inference from Biological Data

In many applications, the network is not known in advance and must be reconstructed from data. This is one of the most important practical challenges in the field.

Early work by Akutsu et al. showed that, under ideal conditions, a Boolean network with bounded in-degree can be reconstructed exactly from time-series observations [6]. This result was important theoretically because it showed that reconstruction is possible in principle, but it also highlighted the computational difficulty of the problem in general.

In practice, biological data are noisy, incomplete, and often too short to determine a unique model. BoolNet provides one practical approach by searching for Boolean functions that best fit observed transitions, while tolerating limited inconsistencies due to noise [7]. More recently, the Sketches framework combines prior structural knowledge with temporal logic constraints derived from data and computes the complete set of networks consistent with both sources of information [5, 17].

The advantage of this latter approach is that it makes uncertainty explicit. Instead of forcing a single answer when the data are ambiguous, it returns all consistent candidate networks. This is useful scientifically because it shows exactly which parts of the model are well determined by the data and which parts remain uncertain.

2.2 Medicine and Therapeutics

Boolean networks are not only useful for explaining biological systems, but also for studying disease and possible treatments. The main idea is that a disease state can often be viewed as an undesirable attractor of the regulatory network. If this is the case, then a treatment can be interpreted as an intervention that removes that attractor or redirects the dynamics toward a healthy one.

2.2.1 Cancer

Cancer involves abnormal regulation of cell growth, survival, and death. Because these processes are controlled by complex interacting networks, Boolean models are well suited to studying them.

Calzone et al. constructed a Boolean model of the signalling network that determines whether a cell survives, undergoes apoptosis, or dies through necrosis [18]. The network included death receptors, apoptosis-related proteins, and survival pathways. Under normal conditions, the model predicted healthy behaviour, while oncogenic mutations shifted the dynamics toward survival, thereby reproducing cancer-like behaviour.

The practical importance of such models is that they allow researchers to test perturbations computationally. One can force a node off to mimic a drug inhibition, or force it on to mimic constitutive activation, and observe how the attractor structure changes. In this way, the model becomes a tool for exploring therapeutic interventions.

This is exactly what happened in the T-LGL case discussed earlier [10, 19]. By simulating perturbations, researchers identified candidate targets that could drive the disease network toward apoptosis. Similar logic was later applied to bladder cancer models, where combinations of perturbations were found to be more effective than single interventions [20].

2.2.2 Immunology

The immune system is a complex regulatory system that must detect threats, coordinate responses, and then return to equilibrium. Boolean networks are useful here because they can represent how many interacting signals combine to produce a cellular decision.

For example, Saadatpour et al. studied a reduced Boolean model of T-LGL survival signalling and analysed its attractors and basins of attraction [19]. Their goal was not only to identify the disease attractor, but also to understand which initial conditions and stimuli favour transition toward that attractor.

This is important because it moves the analysis beyond a simple distinction between healthy and diseased states. It asks a deeper question: under what conditions does the system become trapped in the disease state? For a non-biologist, this is similar to asking not only where the system can end up, but also from which starting conditions that outcome becomes likely.

2.2.3 Drug Target Identification

A major goal in therapeutic modelling is to identify control points: variables whose manipulation can move the network from an undesirable attractor to a desirable one.

Zanudo and Albert introduced the idea of stable motifs, which are self-sustaining subnetworks that help lock the whole system into a particular attractor [21]. If one can disrupt the stable motifs that support a disease state, then one may be able to destabilise that state and push the system elsewhere.

A related structural approach uses feedback vertex sets [22]. These are sets of nodes whose removal breaks all cycles in the network. Since cycles are often responsible for maintaining long-term behaviour, controlling such nodes can strongly influence the network dynamics.

For a reader without a biological background, both methods can be understood as ways of finding strategically important control points in a complex logical system. Instead of trying to manipulate everything, they identify a small number of variables that exert disproportionate influence over the long-term behaviour.

2.3 Engineering and Computer Science

Although Boolean networks were introduced in a biological context, their mathematical structure makes them relevant to many problems in engineering and computer science. In both areas, systems are often described in terms of states, transitions, logic, and control, which aligns naturally with the Boolean-network framework.

2.3.1 Digital Circuit Analysis and Design

A synchronous Boolean network is formally very similar to a synchronous digital circuit. In both cases, the system has a collection of binary variables and update rules that determine the next state from the current one.

This connection is useful in two directions. First, methods developed for hardware verification, such as BDD-based model checking and SAT solving, can be applied to Boolean-network analysis [23, 24]. Second, Boolean networks themselves can be used as abstract models of circuit behaviour, especially in asynchronous settings where different components do not update simultaneously.

For a non-specialist, the analogy is simple: a Boolean network can be viewed as a logic circuit that evolves over time. Instead of gates wired on a chip, the nodes may represent genes or software states, but the formal analysis tools are often very similar.

2.3.2 Network Controllability and Control Theory

Control theory studies how to guide a system from one state to another. In Boolean networks, this becomes the problem of steering the system toward a desired attractor.

Cheng and Qi developed an algebraic framework for analysing controllability and observability in Boolean control networks [25]. A more graph-based view is provided by the concept of driver nodes, which are variables whose external control is sufficient to influence the rest of the network [26].

The intuition here is straightforward: if we want a complex system to reach a target behaviour, we do not necessarily need to control every component. We only need to identify the components through which influence can propagate effectively.

2.3.3 Fault Detection and Reliability

Boolean networks have also been applied to reliability analysis. In digital circuits, a stuck-at fault means that a wire is permanently fixed to 0 or 1. In biological modelling, this is analogous to a gene knockout or constitutive activation. Because the formal structure is similar, tools from one domain can often be transferred to the other.

This connection is conceptually valuable because it shows that Boolean networks are not tied to one scientific field. They provide a general framework for studying how local disruptions affect global behaviour.

2.3.4 Constraint Solving and Formal Verification

Boolean-network analysis is closely related to logic programming, model checking, and constraint solving. BoNesis formulates Boolean-network synthesis as an Answer Set Programming problem, where candidate networks are constrained by desired dynamical properties [3, 8]. The Sketches framework similarly relies on symbolic model checking with BDDs to analyse whole families of candidate networks at once [5, 17].

For a reader from computer science, this is one of the most natural application areas: Boolean networks provide a domain in which formal methods can be used not only for verification, but also for model construction and inference.

2.4 Ecology and Evolutionary Biology

Boolean networks have also been used beyond the cellular scale, especially in ecology and evolutionary biology.

2.4.1 Ecological Community Dynamics

In ecology, one can represent each species as present or absent and define Boolean rules describing how the persistence of one species depends on the presence of others. Predation, com-

petition, and mutual dependence can all be encoded qualitatively in this way.

The attractors of such ecological Boolean networks correspond to stable community compositions, that is, groups of species that can coexist over time. Their basins of attraction indicate which initial community states lead to which stable outcomes. This offers a qualitative alternative to continuous ecological models and can be useful when detailed quantitative data are unavailable.

2.4.2 Evolution of Gene Regulatory Networks

Boolean networks have also been used to study how regulatory architectures evolve. Aldana showed that scale-free Boolean networks can remain dynamically ordered even when connectivity is relatively high [27]. This suggests that certain biologically observed topologies may be favoured because they balance robustness and flexibility.

Wagner used Boolean networks to study the evolution of canalization, that is, the tendency of development to produce stable phenotypes despite variation and noise [28]. By evolving populations of Boolean networks under selection for stable attractors, he showed that robust regulatory structures can emerge spontaneously.

These studies are important because they use Boolean networks not only to model existing systems, but also to ask why certain kinds of network structures may have been favoured over evolutionary time.

Chapter 3

Boolean Networks

3.1 Definition of Boolean Networks **18**
3.2 Synchronous and Asynchronous Update Schemes **18**
3.2.1 Synchronous Update 19
3.2.2 Asynchronous Update 19
3.2.3 Diagram of Update Schemes 20
3.3 Transition Graph **20**
3.4 Attractors **21**
3.5 Structural Properties of Boolean Functions **21**
3.5.1 Essentiality 22
3.5.2 Monotonicity 22
3.5.3 Canalization 23
3.6 Computational Tools for Boolean Network Analysis and Synthesis **23**
3.6.1 BoNesis 23
3.6.2 BoolNet 31
3.6.3 BoolForge 33
3.6.4 bioLQM 35
3.6.5 Sketches 39

This chapter introduces the theoretical foundations of Boolean networks and the concepts that are most relevant to their analysis and practical use. Boolean networks provide a simple yet powerful mathematical framework for modelling complex dynamical systems composed of interacting components, each of which can take only a finite number of states, typically binary [11, 12]. They have been widely used in areas such as systems biology, gene regulatory networks, and computer science [2, 9].

At an abstract level, a Boolean network can be viewed as a discrete dynamical system defined over a finite state space. Each component (or node) represents a variable whose value evolves over time according to a Boolean function that depends on other variables in the system. The global state of the network is given by the combination of all variable values, and the evolution of the system is determined by repeatedly applying these update functions. In this way, a Boolean network captures how local interactions between components give rise to global system behaviour over time.

3.1 Definition of Boolean Networks

A Boolean network is a discrete dynamical system defined by a set of variables $X = \{x_1, x_2, \ldots, x_n\}$, where each variable x_i takes values in $\{0, 1\}$ [29]. Each variable represents the state of a component in the system (e.g., a gene being active or inactive).

The evolution of the system is governed by a set of Boolean functions $F = \{f_1, f_2, \ldots, f_n\}$, where each function $f_i : \{0, 1\}^n \to \{0, 1\}$ determines the next value of x_i based on the current state of the network [29].

Formally, a Boolean network can be represented as a function:

$$F : \{0, 1\}^n \to \{0, 1\}^n, \quad F(x_1, \ldots, x_n) = (f_1(x), \ldots, f_n(x)).$$

The update of the network can be performed synchronously, where all variables are updated simultaneously, or asynchronously, where only a subset of variables is updated at each step [30]. The choice of update scheme significantly affects the system's dynamics.

3.2 Synchronous and Asynchronous Update Schemes

The behavior of a Boolean network is strongly influenced by the update scheme used to apply its transition functions. Two of the most common update strategies are synchronous and asynchronous.

A *state* of a Boolean network is a snapshot of the system at a given time, defined by the values

of all its variables. If the network consists of n variables, then each state can be represented as a vector in $\{0,1\}^n$, where each position corresponds to the value (0 or 1) of a specific variable. Since each variable can take only two values, the total number of possible states is 2^n, forming the *state space* of the network.

States are not arbitrary; they are generated through the application of the network's update functions. Starting from an initial configuration, the Boolean functions associated with each variable determine how the system evolves, producing a sequence of states over time. This sequence is often referred to as a *trajectory*, and its structure depends on the chosen update scheme. [2,30].

3.2.1 Synchronous Update

In the synchronous update scheme, all variables of the network are updated simultaneously at each discrete time step. Formally, given a state $s(t) = (x_1(t), \ldots, x_n(t))$, the next state is uniquely determined as:

$$s(t+1) = F(s(t)) = (f_1(s(t)), \ldots, f_n(s(t))).$$

This results in a deterministic system where each state has exactly one successor [29]. Consequently, the dynamics can be represented as a directed graph in which every node has out-degree one.

Synchronous models are simple to analyze and are commonly used in theoretical studies. However, they assume that all components of the system update at the same time, which may not always be realistic in biological systems.

3.2.2 Asynchronous Update

In contrast, the asynchronous update scheme allows only a subset of variables typically one to be updated at each time step. Given a current state s, multiple successor states may exist, each corresponding to updating a different variable:

$$s' = (x_1, \ldots, f_i(s), \ldots, x_n),$$

for some $i \in \{1, \ldots, n\}$.

This leads to a non-deterministic system where a state can have multiple possible successors [30]. Asynchronous updating is often considered more realistic for modeling biological systems, where interactions do not occur simultaneously.

3.2.3 Diagram of Update Schemes

To illustrate the difference, consider a Boolean network with two variables x_1 and x_2 and update functions:

$$f_1(x_1, x_2) = \neg x_2, \quad f_2(x_1, x_2) = \neg x_1.$$

Starting from the state $(0, 0)$:

- Under **synchronous update**, the next state is:

$$(0, 0) \to (1, 1).$$

- Under **asynchronous update**, two possible transitions exist:

$$(0, 0) \to (1, 0) \quad \text{or} \quad (0, 0) \to (0, 1).$$

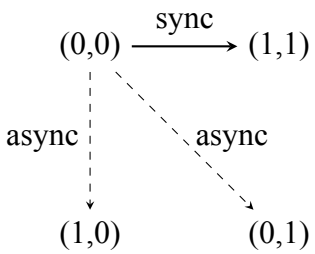


Figure 3.1: Comparison of synchronous (solid) and asynchronous (dashed) updates from state $(0, 0)$.

3.3 Transition Graph

The dynamics of a Boolean network can be represented by a transition graph (also known as a state transition graph) [29]. The nodes of this graph correspond to all possible configurations (states) of the network, i.e., elements of $\{0, 1\}^n$.

An edge from state s to state s' exists if s' is obtained from s by applying the update rules of the network. In the synchronous case, each state has exactly one outgoing edge, since the next state is uniquely determined by F. In the asynchronous case, multiple successor states may exist, depending on which variables are updated [30].

The transition graph provides a complete representation of the system's dynamics and allows for the analysis of reachable states, cycles, and long-term behavior.

3.4 Attractors

Attractors are fundamental objects in the study of Boolean networks, as they characterize the long-term behavior of the system [11].

A fixed point (or steady state) is a state s such that $F(s) = s$. Once the system reaches a fixed point, it remains there indefinitely.

A cyclic attractor is a sequence of states $(s_1, s_2, \ldots, s_k)$ such that $F(s_i) = s_{i+1}$ for $i = 1, \ldots, k-1$, and $F(s_k) = s_1$. The system cycles through these states indefinitely.

The set of states that eventually lead to an attractor is called its basin of attraction. In applications such as biology, attractors are often interpreted as stable patterns or phenotypes of the system [9]. Efficient detection and enumeration of attractors is an important computational problem [31].

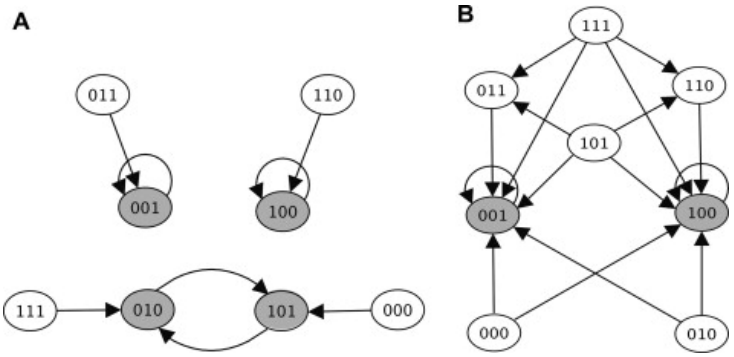


Figure 3.2: This figure illustrates the concept of attractors and their corresponding basins of attraction in a Boolean network. Each node represents a state of the system, and directed edges represent transitions between states. The highlighted cycles correspond to attractors, while the surrounding states form their basins of attraction, i.e., all states that eventually evolve into the attractor. This visual representation emphasizes how different initial conditions may lead to different long-term behaviors of the system. (adapted from [2]).

3.5 Structural Properties of Boolean Functions

The behavior of a Boolean network is not only determined by its structure, but also by the properties of its update functions. Certain structural properties of Boolean functions such as essentiality, monotonicity, and canalization play a crucial role in shaping the dynamics, interpretability, and robustness of the network [2, 17].

These properties are particularly important in applications such as biological modeling, where they correspond to meaningful regulatory mechanisms and constraints.

3.5.1 Essentiality

Essentiality captures whether a variable has a genuine influence on the output of a Boolean function.

Formally, a variable x_i is said to be *essential* in a Boolean function f if there exists at least one input assignment for which changing the value of x_i changes the output of the function [17]. That is, there exists a state x such that:

$$f(x[i \mapsto 0]) \neq f(x[i \mapsto 1]).$$

Intuitively, if a variable is not essential, then it has no impact on the function and can be removed without altering the behavior of the system. Identifying essential variables is therefore useful for simplifying models and reducing complexity.

In biological networks, essential variables correspond to regulators that actively influence the behavior of a gene or protein. Detecting such variables helps identify critical components of the system.

3.5.2 Monotonicity

Monotonicity describes whether the influence of a variable on a Boolean function is consistently increasing or decreasing.

A Boolean function f is said to be *monotone increasing* in a variable x_i if increasing the value of x_i (from 0 to 1) does not decrease the output of the function. Formally:

$$f(x[i \mapsto 0]) \leq f(x[i \mapsto 1]) \quad \text{for all } x.$$

Similarly, f is *monotone decreasing* in x_i if:

$$f(x[i \mapsto 1]) \leq f(x[i \mapsto 0]) \quad \text{for all } x.$$

Monotonicity provides insight into the type of interaction between variables. A positive monotonic relationship can be interpreted as activation, while a negative monotonic relationship corresponds to inhibition [17].

From a modeling perspective, monotone functions are easier to analyze and often lead to more predictable system behavior.

3.5.3 Canalization

Canalization refers to the situation where a specific value of an input variable determines the output of a Boolean function, regardless of the values of other inputs.

Formally, a function f is *canalizing* in variable x_i if there exist values $b \in \{0,1\}$ and $o \in \{0,1\}$ such that:

$$f(x[i \mapsto b]) = o \quad \text{for all } x.$$

[17]

In other words, whenever $x_i = b$, the output of the function is fixed to o, independently of all other variables.

Canalization is closely related to robustness, as it implies that certain inputs dominate the behavior of the system. This property is frequently observed in biological regulatory networks, where specific signals can override others and enforce a particular outcome.

Understanding canalization helps in identifying dominant regulators and simplifying Boolean models by focusing on the most influential interactions.

3.6 Computational Tools for Boolean Network Analysis and Synthesis

Several software tools have been developed to support the modelling, analysis, and verification of Boolean networks. These tools enable researchers to simulate dynamics, analyse attractors, reconstruct network logic from data, and synthesise models from formal specifications. This section surveys four representative tools that are relevant to the work presented in this thesis.

3.6.1 BoNesis

BoNesis is a tool designed for the reasoning and synthesis of Boolean networks based on formal specifications [3, 8]. It allows users to define constraints on the behavior of a network and automatically generate models that satisfy these constraints.

The tool is particularly useful in scenarios where the network structure is partially known, and one seeks to infer missing interactions while ensuring consistency with observed behaviors.

BoNesis operates by combining a description of the space of possible Boolean networks with a set of logical constraints that encode desired dynamical properties.

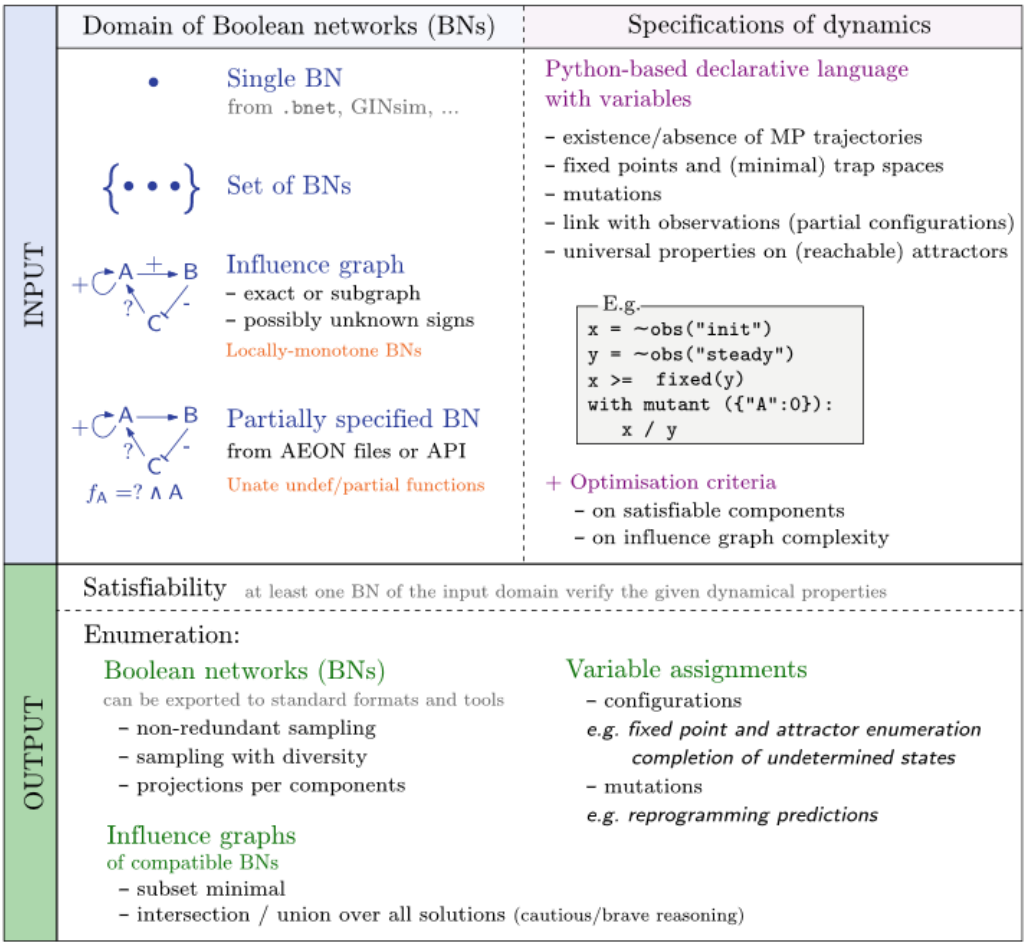


Figure 3.3: Overview of the BoNesis framework (adapted from [3]). The input consists of a domain of Boolean networks (e.g., a single network, a set of networks, or partially specified interaction graphs) together with declarative specifications of desired dynamical properties expressed in a Python-based language. These specifications may include constraints on fixed points, attractors, mutations, and observational consistency. BoNesis then performs constraint solving to determine satisfiability and to enumerate Boolean networks and variable assignments that satisfy the given properties. The output may include valid network models, configurations, attractors, and predictions about system behavior.

As illustrated in Figure 3.3, the BoNesis framework formulates Boolean network inference as a constraint-solving problem that separates structural assumptions from dynamical specifications.

On the one hand, the user defines a *domain of Boolean networks*, which may consist of a single fully specified network or, more commonly, a set of candidate networks described through partial knowledge such as an influence graph or partially specified update functions. This representation allows uncertainty in the network structure to be explicitly captured.

On the other hand, the user provides *dynamical properties* derived from observations or biological knowledge. These are expressed declaratively as constraints, specifying requirements such as the existence or absence of trajectories, the presence of fixed points or attractors, and consistency with experimental data.

BoNesis integrates these two components into a single logical formulation and uses Answer Set Programming (ASP) to compute all Boolean networks within the given domain that satisfy the specified constraints. As a result, the framework does not return a single model, but rather an *ensemble of valid networks*, enabling systematic analysis of model uncertainty and the identification of properties shared across all consistent solutions.

3.6.1.1 Boolean Network Reconstruction Using BoNesis

3.6.1.2 Influence Graph Construction

The first step in every BoNesis reconstruction is to encode prior knowledge about regulatory interactions as a signed directed graph, referred to as the *influence graph*. Each edge carries a sign ($+1$ for activation, -1 for inhibition) that restricts which Boolean functions are permissible for each target node: an activating regulator must be capable of increasing the target's value, while an inhibiting regulator must be capable of decreasing it.

```
ig = bonesis.InfluenceGraph(influences)
```

Listing 3.1: Constructing an influence graph in BoNesis.

Here, `influences` is a list of triples `(source, target, {"sign": ±1})`. The resulting object encodes the structural hypothesis space: only Boolean functions whose monotonicity properties are compatible with the signed edges are considered during reconstruction.

3.6.1.3 Problem Initialisation

The influence graph is combined with a collection of observed states to instantiate the reconstruction problem. The observations represent snapshots of the system's state at distinct time points, derived from trajectory data.

```
bo = bonesis.BoNesis(ig, observations)
```

Listing 3.2: Initialising the BoNesis reconstruction problem.

The `BoNesis` object couples structural constraints inherited from the influence graph with empirical constraints encoded through the observations, and exposes a declarative interface for adding further dynamical requirements.

3.6.1.4 Encoding Temporal Constraints

A key capability of BoNesis is the direct imposition of dynamical constraints on the reconstruction problem. In this work, consecutive observations along a simulated trajectory are linked

by *reachability* constraints: the reconstructed network must admit at least one trajectory along which state o_i can reach state o_{i+1}.

```
obs_keys = sorted(observations.keys())
for i in range(len(obs_keys) - 1):
    ~bo.obs(obs_keys[i]) >= ~bo.obs(obs_keys[i + 1])
```

Listing 3.3: Adding reachability constraints between consecutive observed states.

The `~bo.obs()` operator dereferences a named observation as a symbolic state within the ASP encoding. The overloaded `>=` operator asserts that the left-hand state can *reach* the right-hand state through the network's update dynamics. Each such constraint eliminates from the solution space any Boolean network that cannot reproduce the observed temporal ordering.

3.6.1.5 Solution Enumeration

After all constraints are declared, BoNesis invokes its ASP back-end to enumerate consistent Boolean networks. The search is bounded by a user-specified limit to prevent combinatorial explosion in under-constrained settings.

```
solutions = list(bo.boolean_networks(limit=n_solutions))
```

Listing 3.4: Enumerating candidate Boolean networks.

Each element of `solutions` is a fully specified Boolean network whose update functions satisfy every structural and dynamical constraint simultaneously. When multiple solutions exist, the reconstruction problem is *under-determined*: the available data are insufficient to identify the ground-truth network uniquely. The number of returned solutions therefore serves as a direct measure of reconstruction ambiguity, and its relationship to data availability is examined in Section 3.6.1.6.

3.6.1.6 Reconstruction Experiment

Motivation and Objectives The reconstruction of Boolean networks from observational data is a central challenge in systems biology, particularly when the available data are incomplete or noisy. In practical settings, experimental measurements often provide only partial information about the system's dynamics, making it difficult to uniquely determine the underlying regulatory structure. As a result, multiple candidate networks may be consistent with the same observations, leading to ambiguity in the reconstruction process.

This experiment is designed to evaluate the performance of the BoNesis tool under such conditions. In particular, it investigates how the quality of reconstruction is influenced by the amount

of available data. By systematically varying the fraction of observed trajectory states provided as input, the experiment aims to assess how effectively BoNesis can recover the original (ground-truth) Boolean network when only limited information is available.

In addition to data availability, the experiment also considers the effect of network complexity, measured by the number of nodes in the system. As the size of the network increases, the number of possible configurations grows exponentially, making the inference problem more challenging. By analysing the interaction between these two factors, the experiment provides insight into the conditions under which BoNesis can reliably reconstruct Boolean networks, as well as the limitations of the approach in data-scarce scenarios.

Experimental Design **Ground-truth networks.** For each experimental condition a random Boolean network of n nodes is generated with in-degree $k = 2$. Update functions are drawn from a restricted family of simple logical gates identity, negation, conjunction (and), and disjunction (or) chosen to keep the ground truth within a class that is in principle reconstructible from limited data.

Trajectory simulation. Each ground-truth network is simulated from a random initial state for a fixed number of synchronous update steps, producing a state trajectory. Simulation terminates early upon reaching a fixed point.

Observation subsampling. From the full trajectory, a contiguous prefix of length proportional to f is extracted and supplied to BoNesis as the observation set. Smaller values of f represent conditions of higher data scarcity.

Reconstruction and evaluation. BoNesis is invoked with the subsampled observations and the ground-truth influence graph. Up to N_{sol} candidate networks are enumerated. Each candidate is compared to the ground truth node by node; the best-matching candidate determines the accuracy score for that condition. The experiment is replicated R times per condition to account for stochasticity in network generation and initial states. Table 3.1 summarises the experimental parameters.

Table 3.1: Summary of experimental parameters.

Parameter	Symbol	Value / Range
Network size (nodes)	n	*[5,10,...,50]*
In-degree	k	2
Data fractions	f	*20%, 40%, ..., 100%*
Replicates per condition	R	*100*
Candidate networks checked	N_{sol}	*10*
Trajectory length	T	*100 steps*
Reconstruction timeout	τ	*100 seconds*

Results The following figures summarise reconstruction performance across all experimental conditions.

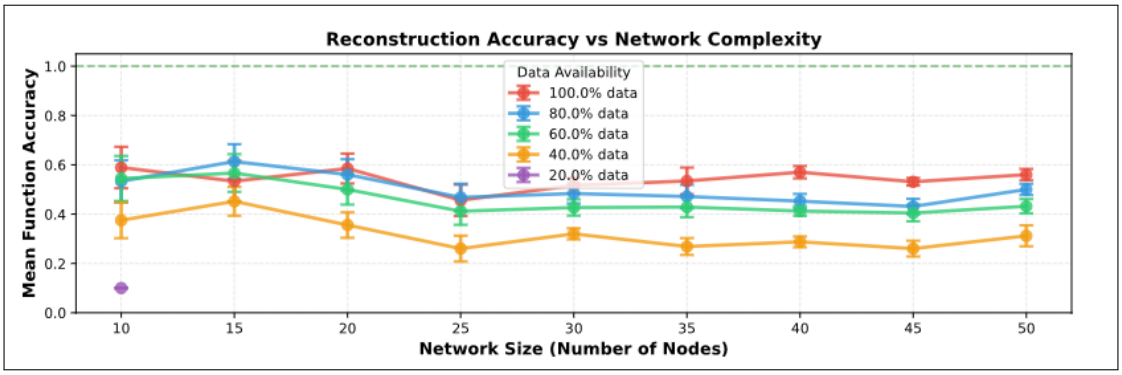


Figure 3.4: Mean Boolean function reconstruction accuracy as a function of network size for each level of data availability. Error bars represent standard error of the mean across replicates.

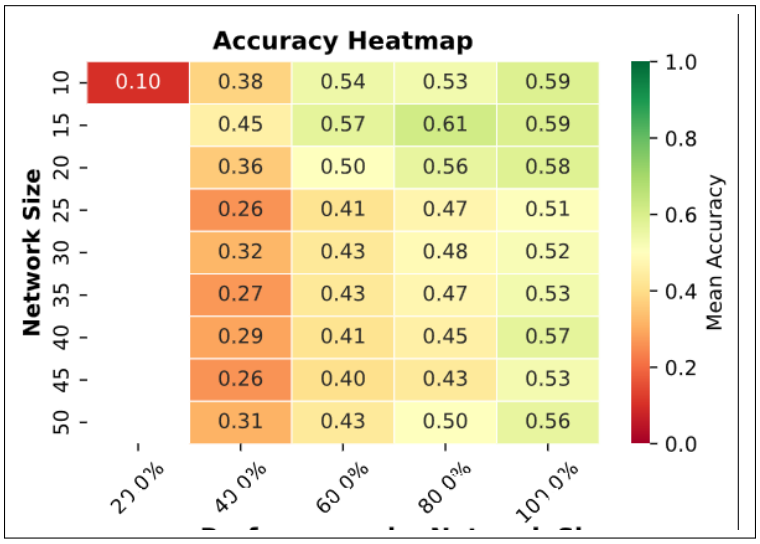


Figure 3.5: Heatmap of mean function accuracy across all combinations of network size and data availability. Greener cells indicate conditions under which reconstruction is most reliable.

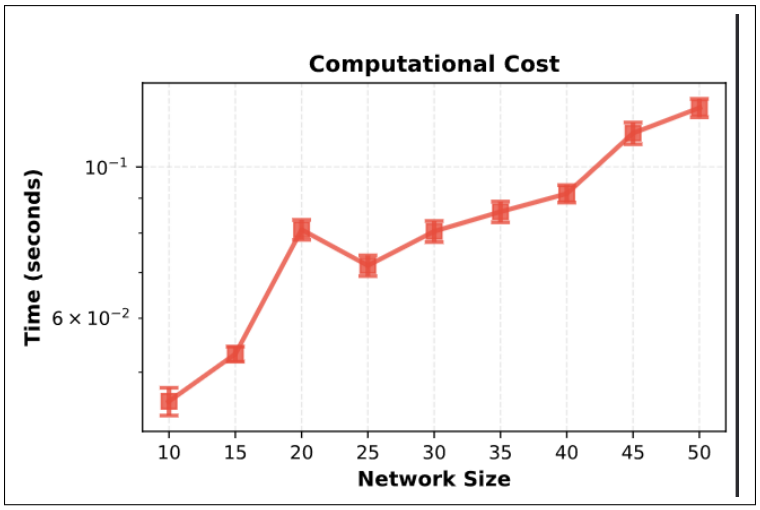
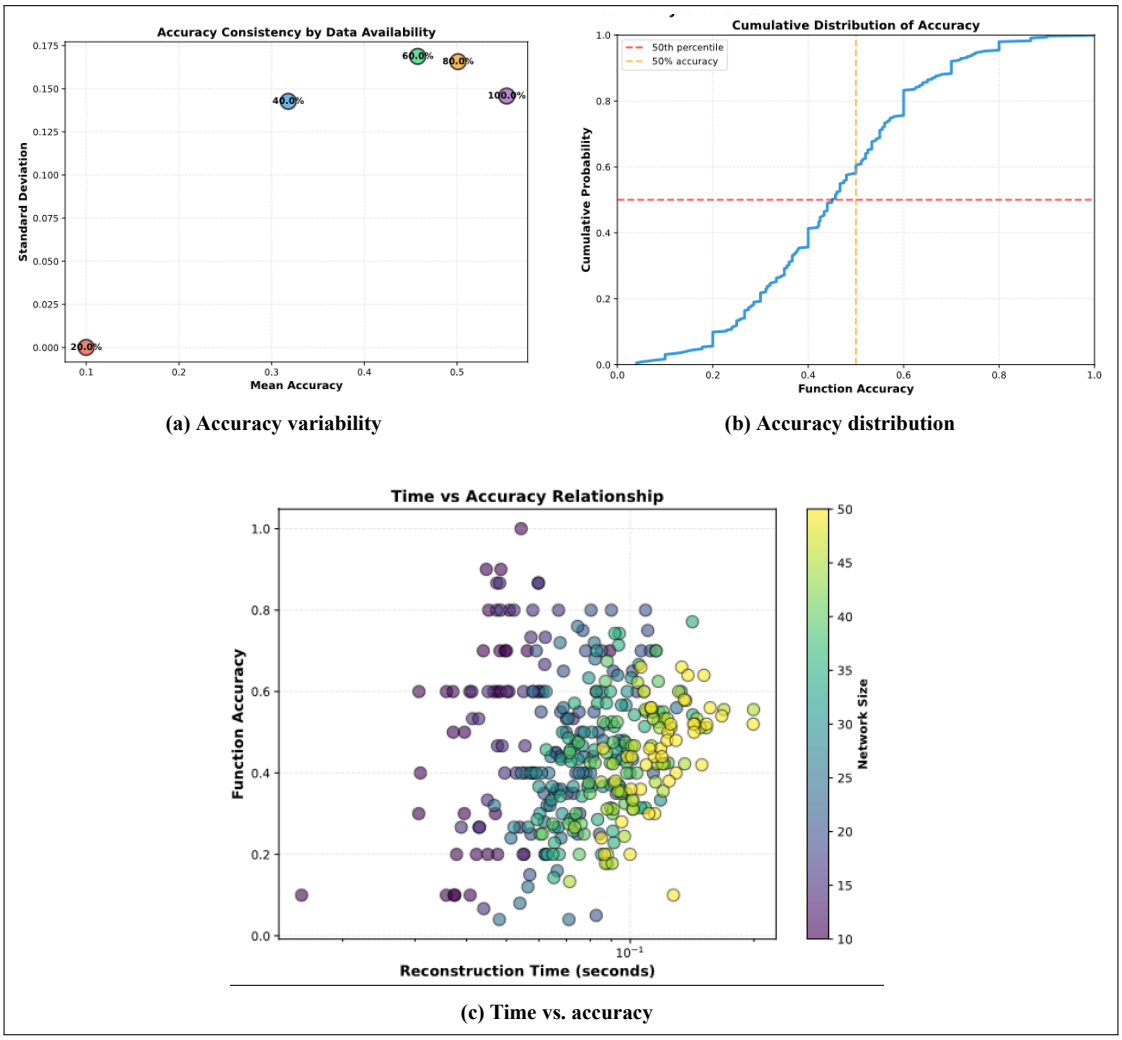


Figure 3.6: Mean reconstruction time as a function of network size (log scale), aggregated across all data-fraction conditions and replicates.



(a) Accuracy variability

(b) Accuracy distribution

(c) Time vs. accuracy

Figure 3.7: Summary statistics of the reconstruction experiment. (a) Accuracy consistency per data fraction. (b) Cumulative distribution of function accuracy. (c) Reconstruction time versus accuracy, coloured by network size.

Evaluation. The results show a clear relationship between data availability and reconstruction accuracy. As the fraction of observed data increases, the mean accuracy improves consistently across all network sizes, with the highest performance achieved at full data availability. In contrast, low data fractions (e.g. 20%) lead to significantly poorer reconstruction, indicating that the information provided is insufficient to uniquely constrain the solution space.

Network size has a moderate but noticeable impact: larger networks tend to exhibit slightly lower accuracy and higher variability, reflecting the increased complexity of the inference problem. The heatmap further confirms that reconstruction is most reliable when both sufficient data and moderate network sizes are present.

In terms of computational cost, reconstruction time grows with network size but remains relatively low overall, demonstrating that BoNesis scales efficiently within the tested range. The

distribution and consistency plots indicate that while average performance improves with more data, variability remains non-negligible, particularly at intermediate data levels. Overall, the experiment highlights that BoNesis performs robustly when provided with sufficient observational data, but its accuracy degrades predictably under data scarcity.

3.6.2 BoolNet

BoolNet is an R package for the generation, simulation, reconstruction, and analysis of Boolean networks [**?**]. It provides a comprehensive framework for studying discrete dynamical systems under synchronous, asynchronous, and probabilistic update schemes, and includes tools for both synthetic and real biological data.

Network definition and representation. Boolean networks in BoolNet are defined through simple, human-readable text files, where each node is associated with a Boolean update function. For example:

Listing 3.5: Example BoolNet network definition file (`network.bn`).

```
targets, factors
A,     B & !C
B,     A | C
C,     !A
```

Each line specifies how the future state of a node depends on the current states of other nodes. The network can then be loaded into R and visualised:

```
library(BoolNet)
net <- loadNetwork("network.bn")
plotNetworkWiring(net)
```

The resulting wiring diagram provides a graphical representation of the regulatory structure, highlighting activation and inhibition relationships between nodes.

Simulation of trajectories. Once a network is defined, its dynamics can be explored by simulating state trajectories. Starting from an initial state, BoolNet applies the update functions iteratively to generate a sequence of states over time.

Listing 3.6: Simulating a state trajectory.

```
# simulate 10 steps from a random initial state
trajectory <- generateTimeSeries(
    net,
    numSeries = 1,
```

```
    numSteps = 10,
    type     = "synchronous"
)
print(trajectory)
```

Such trajectories represent the evolution of the system and form the basis for understanding its long-term behaviour. Depending on the update scheme, different trajectories may be observed even for the same initial state.

Attractor analysis. A key aspect of Boolean network analysis is the identification of attractors, which correspond to stable long-term behaviours such as fixed points or cycles. BoolNet provides both exact and approximate methods for attractor detection:

```
attractors <- getAttractors(net, type = "synchronous")
plotAttractors(attractors)
plotStateGraph(attractors)
```

The state-transition graph represents all possible states and transitions between them, while attractors highlight the states toward which the system eventually converges.

Perturbation and robustness. BoolNet also supports the analysis of system robustness under perturbations, which is particularly relevant in biological contexts. Nodes can be fixed to simulate gene knockouts or overexpression:

```
knocked_out <- fixGenes(net, fixIndices = "A", values = 0)
attr_ko <- getAttractors(knocked_out)
```

In addition, robustness can be quantified by introducing random changes to the network and measuring how often its attractors are preserved:

```
robustness <- testAttractorRobustness(
    net,
    attractors = attractors,
    perturb    = "functions",
    numSamples = 1000
)
```

Reconstruction from data. Finally, BoolNet provides methods for reconstructing Boolean networks from time-series data. Given a sequence of observed states, the package searches for update functions that best explain the observed transitions:

```
data(exampleTimeSeries)
```

```
2
3 reconstructed <- reconstructNetwork(
4     measurements = exampleTimeSeries,
5     method       = "bestfit",
6     maxK         = 3
7 )
8 plotNetworkWiring(reconstructed)
```

The parameter `maxK` limits the number of inputs per node, controlling the complexity of the inferred model. This reconstruction process is particularly useful when the underlying regulatory structure is only partially known.

3.6.3 BoolForge

BoolForge is a Python framework for constructing, editing, and manipulating Boolean network models in a modular and programmatic fashion. Its architecture cleanly separates the structural definition of a network from higher-level operations such as rule editing, model composition, and automated configuration, making it particularly well-suited to large or complex models that must be assembled from reusable components or subjected to systematic configuration sweeps.

Unlike tools that expose a fixed graphical or file-based interface, BoolForge is designed to be embedded inside larger computational pipelines. Models are defined and manipulated entirely through a Python API, which means that network construction, simulation, perturbation analysis, and model composition can all be orchestrated programmatically without manual intervention at any stage.

3.6.3.1 Network Construction and Representation

In BoolForge, a Boolean network is an object that is built incrementally by adding nodes and associating each node with a Boolean update rule. Rules are expressed as logical formulae using standard connectives (`and`, `or`, `not`), which are parsed and stored internally in a form that supports both evaluation and manipulation.

```
1 import boolforge as bf
2
3 # Create an empty network
4 net = bf.BooleanNetwork()
5
6 # Add nodes with logical update rules
7 net.add_node("A", rule="B and not C")
8 net.add_node("B", rule="A or C")
```

```
net.add_node("C", rule="not A")

# Visualise the regulatory graph
net.draw(layout="spring", save_path="network.png")
```

Listing 3.7: Constructing a three-node Boolean network in BoolForge.

The network defined above encodes the following regulatory logic: node `A` is active when `B` is active and `C` is inactive; node `B` is active when either `A` or `C` is active; node `C` is active when `A` is inactive.

3.6.3.2 Dynamics Simulation and Attractor Analysis

Once a network is defined, BoolForge can simulate its dynamics under both synchronous and asynchronous update schemes. In the synchronous scheme all nodes are updated simultaneously at each time step, producing a deterministic trajectory from any given initial state. In the asynchronous scheme a single node is updated at each step; BoolForge explores the resulting non-deterministic dynamics by enumerating all possible update orderings.

```
# Simulate from a specific initial state (synchronous update)
init_state = {"A": 1, "B": 0, "C": 1}
trajectory = net.simulate(init_state, steps=20, mode="synchronous")

# Print the trajectory as a state sequence
for t, state in enumerate(trajectory):
    print(f"t={t}: {state}")

# Compute all attractors via exhaustive state-space traversal
attractors = net.find_attractors(mode="synchronous")
for attr in attractors:
    print(attr)
```

Listing 3.8: Trajectory simulation and attractor identification in BoolForge.

`simulate` returns a list of state dictionaries, each mapping node names to their Boolean values at that time step. The trajectory terminates either when the specified number of steps is reached or when a previously visited state is encountered, signalling entry into an attractor. `find_attractors` performs an exhaustive traversal of the state-transition graph, enumerating all fixed points and limit cycles reachable from any initial condition. For a network of n nodes the state space has 2^n states, so exhaustive search is feasible for moderate network sizes and is complemented by heuristic sampling for larger ones.

3.6.3.3 *Rule Editing, Perturbations, and Model Composition*

A distinguishing feature of BoolForge is its support for structured rule editing after initial construction. Individual update rules can be replaced at any time, and nodes can be *clamped* fixed to a constant Boolean value to simulate gene knockouts (clamp to 0) or constitutive activations (clamp to 1). This perturbation capability is essential for studying the effect of regulatory interventions on attractor landscapes.

```
# Replace an update rule after construction
net.set_rule("A", "B and C")

# Simulate a gene knockout: fix node C to 0
net.fix_node("C", value=0)
ko_attractors = net.find_attractors()

# Define a second sub-network
subnet1 = bf.BooleanNetwork()
subnet1.add_node("X", rule="Y")
subnet1.add_node("Y", rule="not X")

# Merge both sub-networks with an inter-module coupling rule
merged = bf.merge([net, subnet1], link_rules={"A": "A or X"})
```

Listing 3.9: Rule editing, gene knockout simulation, and sub-network composition in BoolForge.

The `fix_node` call clamps `C` to 0, effectively removing it from the dynamics and reducing the state space from $2^3 = 8$ to $2^2 = 4$ states. Comparing the attractors of the original and perturbed networks reveals whether the attractor landscape is robust to the knockout a standard robustness test in systems biology. The `merge` function combines independently defined sub-networks into a single model. The `link_rules` argument specifies coupling rules that introduce dependencies between variables in different sub-networks: here, `A`'s update rule is extended so that it also activates when `X` is active. This composability makes BoolForge well-suited for building large regulatory models from smaller, independently validated modules.

3.6.4 bioLQM

`bioLQM` (Biological Logical Qualitative Modelling) [32] is a Java library and command-line tool designed for the manipulation and analysis of logical models of regulatory and signalling networks [32]. It is especially suited to Boolean and multi-valued qualitative models, where each component is represented by a discrete activity level and updated according to logical rules.

Its design philosophy is that of a general-purpose logical modelling environment: rather than targeting a single analysis task, `bioLQM` provides a uniform framework for model loading, format conversion, dynamical analysis, and model transformation that can be used both interactively and as a backend component inside larger automated pipelines.

3.6.4.1 Architecture and Interoperability

A practical strength of `bioLQM` is its broad support for model interchange formats. Logical models can be imported from and exported to several widely used representations, including the `.bnet` format used by tools such as BoolNet and the SBML-qual extension of the Systems Biology Markup Language. This interoperability is important in practice because different tools in the Boolean network ecosystem expect different input formats; `bioLQM` serves as a conversion and analysis layer that bridges them.

A model conversion is performed from the command line as follows:

```
# Convert a model from SBML-qual to bnet format
bioLQM model.sbml model_converted.bnet
```

The same syntax generalises to any supported pair of input and output formats, making format conversion entirely transparent to the user.

3.6.4.2 Fixed Points and Trap Spaces

For Boolean network research, the most important analytical capabilities of `bioLQM` are the computation of *fixed points* and *trap spaces*.

3.6.4.2.1 Fixed points. A fixed point is a state $x \in \mathbb{B}^n$ of the Boolean network F that is invariant under the update:

$$F(x) = x.$$

Fixed points are the simplest class of attractors: once reached, the system remains there indefinitely. In biological terms, fixed points are typically interpreted as stable cellular phenotypes or long-term regulatory regimes for example, the differentiated state of a cell type or the survival versus apoptosis decision of a signalling network. `bioLQM` computes all fixed points of a model efficiently using constraint propagation over the logical structure of the update functions, without requiring exhaustive enumeration of all 2^n states.

3.6.4.2.2 Trap spaces. A trap space is a *partial* state description a hypercube $T \subseteq \mathbb{B}^n$ defined by fixing a subset of the n variables to constant values and leaving the rest free that is closed under the network dynamics: once the system enters T, it cannot leave it. Formally, for

every state $x \in T$ and every successor state y with $x \to y$ in the state-transition graph, we require $y \in T$.

Trap spaces are important for several reasons. First, they provide a *compact* characterisation of long-term behaviour: every attractor is contained within some minimal trap space, so computing trap spaces gives information about the attractor landscape without requiring full attractor enumeration. Second, minimal trap spaces often correspond directly to attractors (fixed points and oscillatory cycles), making them a useful proxy in large networks where complete attractor analysis is computationally intractable. Third, trap spaces can be computed using efficient Boolean satisfiability techniques that scale to networks with hundreds of variables.

The two analyses are invoked through a uniform command-line syntax:

```
# Compute all fixed points of the model
bioLQM model.bnet -r fixpoints

# Compute all minimal trap spaces of the model
bioLQM model.bnet -r trapspaces
```

Both commands read the model from a `.bnet` file and print the results to standard output in a structured format that can be parsed by downstream scripts.

3.6.4.3 Programmatic Usage in Computational Pipelines

When `bioLQM` is used as a component of a larger computational workflow, the most common pattern is to invoke it as a subprocess from a Python script, passing the model path and the desired analysis as command-line arguments:

```python
import subprocess

def run_biolqm(model_path, analysis):
    """
    Run a bioLQM analysis on a .bnet model file.

    Parameters
    ----------
    model_path : str
        Path to the Boolean network model in .bnet format.
    analysis : str
        Analysis type: 'fixpoints' or 'trapspaces'.

    Returns
```

```
    -------
    str
        Raw text output from bioLQM.
    """
    cmd = ["bioLQM", str(model_path), "-r", analysis]
    result = subprocess.run(cmd, capture_output=True, text=True)
    return result.stdout

# Compute fixed points
fp_output  = run_biolqm("model.bnet", "fixpoints")

# Compute minimal trap spaces
ts_output  = run_biolqm("model.bnet", "trapspaces")

print("Fixed points:\n", fp_output)
print("Trap spaces:\n", ts_output)
```

Listing 3.10: Constructing `bioLQM` analysis commands programmatically in Python.

This pattern wrapping command-line calls in thin Python functions is typical of how `bioLQM` is integrated into automated experiment pipelines. Because both `fixpoints` and `trapspaces` are accessed through the same interface, the same wrapper function can be reused for either analysis by simply changing the `analysis` argument. The raw outputs can then be parsed and fed directly into downstream inference or verification tools.

3.6.4.4 Model Transformations and Perturbations

Beyond static analysis, `bioLQM` supports a range of model transformations. Nodes can be fixed to constant values to simulate gene knockouts or constitutive activations; the model can be reduced by eliminating variables whose values are determined by the values of other nodes; and the update scheme can be modified to study the effect of different updating assumptions on the attractor landscape. These transformations are specified as additional command-line flags and compose naturally, so that, for example, a knockout followed by a trap-space analysis can be expressed as a single command invocation.

This makes `bioLQM` useful not only for observing the baseline behaviour of a model but also for systematic perturbation studies of the kind routinely performed in computational systems biology, where one wishes to understand how the attractor landscape changes as individual regulatory components are activated, inhibited, or removed.

3.6.5 Sketches

Sketches is a formal framework for specifying and completing partially defined Boolean network models [5, 17]. A *sketch* is a structured template that encodes prior knowledge about a network its topology, the signs of regulatory interactions, and qualitative dynamical properties while leaving unspecified components as placeholders to be resolved automatically through constraint solving. The result is a set of fully specified Boolean networks that are consistent with all stated requirements.

Because the Sketches formalism is central to a dedicated chapter of this thesis, a full technical treatment including its formal semantics, the class of properties it can express, and the algorithms used for completion is deferred to Chapter 5.

Chapter 4

Temporal Logic for Boolean Networks

4.1 Motivation: Why Temporal Logic for Boolean Networks **41**
4.2 Foundations of Temporal Logic **41**
4.2.1 States and Transition Systems 41
4.2.2 Paths and Executions 42
4.2.3 Temporal Operators 42
4.2.3.1 Linear Temporal Logic (LTL) 43
4.2.3.2 Computation Tree Logic (CTL) 43
4.3 Expressing Boolean Network Properties in Temporal Logic **44**
4.3.1 Reachability 44
4.3.2 Stability (Fixed Points) 45
4.3.3 Cyclic Behavior and Attractors 45
4.3.4 Safety and Liveness Properties 45
4.4 Temporal Logic in Real Applications **46**
4.4.1 Modeling Real-World Behaviors 46
4.4.2 Examples from Applications 46
4.4.3 Interpretability of Temporal Specifications 46

This chapter introduces the temporal-logic concepts needed to describe and verify dynamical properties of Boolean networks. Temporal logic provides a formal framework for reasoning about properties that evolve over time [33, 34].

4.1 Motivation: Why Temporal Logic for Boolean Networks

Boolean networks evolve over discrete time steps, producing sequences of states that represent system behavior. Such systems are naturally modeled as dynamical systems with temporal evolution. Temporal logic enables the specification of properties over execution paths, such as "eventually a condition holds" or "a condition always holds" [34]. It has become a standard tool in system verification and formal reasoning.

4.2 Foundations of Temporal Logic

4.2.1 States and Transition Systems

Let AP be a finite set of atomic propositions. A *state* is an assignment

$$s : AP \to \{0, 1\}$$

indicating which propositions are true.

In the context of Boolean networks, a state corresponds to a valuation of all variables, i.e., a vector in $\{0, 1\}^n$, and atomic propositions typically represent properties such as "gene x_i is active" [2].

A *transition system* is a tuple:

$$TS = (S, R)$$

where:

- S is a finite set of states,
- $R \subseteq S \times S$ is a transition relation.

The relation $(s, s') \in R$ indicates that the system can evolve from state s to state s' in one step.

In Boolean networks, the transition relation is induced by the update rules. In the synchronous case, each state has exactly one successor, whereas in the asynchronous case, multiple successors may exist [30].

A *Kripke structure* is a tuple:

$$K = (S, R, L)$$

where:

- S is a finite set of states,
- $R \subseteq S \times S$ is a total transition relation,
- $L : S \to 2^{AP}$ is a labeling function assigning to each state the set of atomic propositions that hold in that state.

Kripke structures provide the standard semantic model for temporal logics [24].

Example. Consider a Boolean network with two variables (x_1, x_2). The state $(1, 0)$ may represent that x_1 is active while x_2 is inactive. A transition $(1, 0) \to (1, 1)$ indicates that x_2 becomes active in the next step.

4.2.2 Paths and Executions

A *path* (or execution) in a transition system is an infinite sequence of states:

$$\pi = s_0, s_1, s_2, \ldots$$

such that $(s_i, s_{i+1}) \in R$ for all $i \geq 0$.

Paths represent possible evolutions of the system over time. In Boolean networks, each path corresponds to a trajectory of the system starting from an initial state.

We denote by $\pi[i]$ the i-th state of the path, and by $\pi[i :]$ the suffix

$$\pi[i :] = s_i, s_{i+1}, \ldots$$

Example. A possible execution of a Boolean network is:

$$(0, 0) \to (1, 0) \to (1, 1) \to (1, 1) \to \ldots$$

This path eventually stabilizes in a fixed point $(1, 1)$, illustrating how temporal logic can describe convergence and long-term behavior.

Depending on the update scheme, the set of paths may vary significantly. In asynchronous Boolean networks, multiple possible paths may originate from the same initial state, reflecting non-deterministic behavior [30].

4.2.3 Temporal Operators

Temporal logic extends propositional logic with operators that refer to time. These operators allow the expression of properties concerning the future evolution of a system.

4.2.3.1 Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) is interpreted over individual paths [24]. Its syntax is defined inductively as:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi_1 \, U \, \varphi_2$$

where $p \in AP$.

The semantics of LTL is defined over paths π:

- $\pi \models p$ iff $p \in L(\pi[0])$
- $\pi \models X\varphi$ iff $\pi[1:] \models \varphi$
- $\pi \models F\varphi$ iff $\exists i \geq 0 : \pi[i:] \models \varphi$
- $\pi \models G\varphi$ iff $\forall i \geq 0 : \pi[i:] \models \varphi$
- $\pi \models \varphi_1 \, U \, \varphi_2$ iff $\exists i \geq 0$ such that $\pi[i:] \models \varphi_2$ and for all $j < i$, $\pi[j:] \models \varphi_1$

Interpretation.

- $F\,p$: eventually p holds (e.g., a gene becomes active)
- $G\,p$: p always holds (e.g., a gene remains active)
- $X\,p$: p holds in the next state
- $p\,U\,q$: p holds until q becomes true

LTL is particularly suitable for expressing properties of single executions, such as whether a system eventually reaches a stable configuration.

4.2.3.2 Computation Tree Logic (CTL)

Computation Tree Logic (CTL) extends temporal logic with path quantifiers, allowing reasoning over multiple possible executions [24].

Its syntax includes operators of the form:

$$AX,\ EX,\ AF,\ EF,\ AG,\ EG$$

where A denotes "for all paths" and E denotes "there exists a path".

The semantics of CTL is defined over states of a Kripke structure:

- $K, s \models EX\varphi$ iff there exists a successor s' such that $(s, s') \in R$ and $K, s' \models \varphi$
- $K, s \models AX\varphi$ iff for all successors s' with $(s, s') \in R$, $K, s' \models \varphi$

Interpretation.

- $EF\,p$: there exists a path where eventually p holds
- $AF\,p$: for all paths, eventually p holds
- $AG\,p$: for all paths, p always holds
- $EG\,p$: there exists a path where p always holds

CTL is particularly useful for reasoning about branching behaviors, which naturally arise in asynchronous Boolean networks where multiple future evolutions are possible.

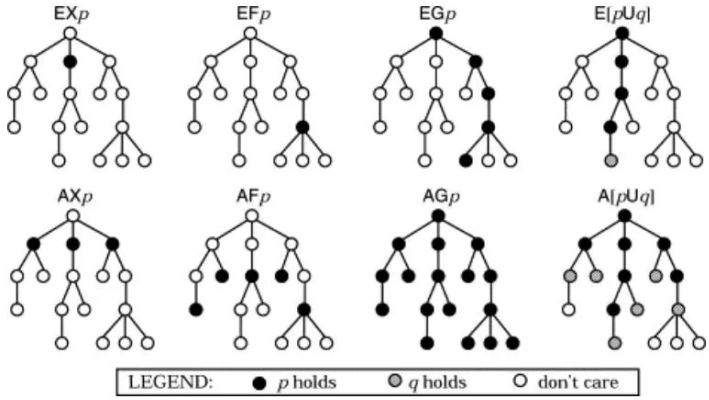


Figure 4.1: Illustration of Computation Tree Logic (CTL) semantics, showing branching-time paths and temporal operators. Reproduced from [4].

4.3 Expressing Boolean Network Properties in Temporal Logic

4.3.1 Reachability

Reachability properties describe whether a system can evolve from an initial state to a state satisfying a given condition. In temporal logic, this is typically expressed using formulas such as:

$$F\,p \quad \text{(LTL)} \qquad \text{or} \qquad EF\,p \quad \text{(CTL)}$$

which state that a state satisfying p is eventually reached.

In the context of Boolean networks, reachability is used to model biological questions such as whether a gene can become activated under certain conditions, or whether a system can transition to a desired phenotype. Such properties are commonly analyzed using temporal logic and model checking techniques in biological regulatory networks [35].

4.3.2 Stability (Fixed Points)

Stability properties characterize fixed points of the system, i.e., states that remain unchanged under the dynamics of the Boolean network. In temporal logic, stability can be expressed as:

$$G\,p \quad \text{(LTL)} \qquad \text{or} \qquad AG\,p \quad \text{(CTL)}$$

indicating that a property p holds indefinitely along a path or across all paths.

In Boolean network models of biological systems, fixed points correspond to stable cellular states or phenotypes. Temporal logic has been used to formally verify such steady states and their robustness in regulatory networks [35, 36].

4.3.3 Cyclic Behavior and Attractors

Cyclic behaviors, including oscillations and attractors, are central to the analysis of Boolean networks. These behaviors can be expressed using temporal logic formulas such as:

$$GF\,p \quad \text{(LTL)}$$

which states that the condition p holds infinitely often along a path.

In CTL, similar properties can be expressed using formulas such as:

$$EG\,F\,p$$

indicating the existence of a path where p recurs indefinitely.

Such formulations are used to characterize attractors and oscillatory dynamics in biological networks, for example in models of gene regulation and signaling pathways [37]. These properties allow distinguishing between transient and recurring behaviors in the system.

4.3.4 Safety and Liveness Properties

Safety and liveness properties provide a general framework for specifying correctness conditions in dynamical systems.

Safety properties ensure that undesirable states are never reached. They are typically expressed as:

$$G\neg p \quad \text{(LTL)} \qquad \text{or} \qquad AG\neg p \quad \text{(CTL)}$$

meaning that a condition p is never true.

Liveness properties ensure that desirable states are eventually reached. These are expressed as:

$$F\,p \quad \text{(LTL)} \qquad \text{or} \qquad AF\,p \quad \text{(CTL)}$$

In Boolean networks, safety properties can represent constraints such as the absence of undesirable gene expression patterns, while liveness properties capture behaviors such as eventual activation of regulatory pathways. These concepts are widely used in the formal verification of biological systems [24, 36].

4.4 Temporal Logic in Real Applications

Temporal logic has been successfully applied to the analysis of real-world systems, particularly in systems biology and formal verification.

4.4.1 Modeling Real-World Behaviors

In biological systems, temporal logic allows the formal specification of behaviors such as activation, inhibition, persistence, and oscillation of genes and proteins. These properties are essential for understanding regulatory mechanisms and system dynamics.

For example, one may specify that a gene eventually becomes active ($F\,p$), remains active indefinitely ($G\,p$), or is periodically activated ($GF\,p$), capturing different biological phenomena.

4.4.2 Examples from Applications

Temporal logic has been extensively used to analyze gene regulatory networks and signaling pathways. For instance, formal verification techniques have been applied to biological models to ensure consistency with experimental observations and to predict system behavior under perturbations [36].

Such approaches allow researchers to verify whether a model satisfies desired biological properties, such as reaching a specific steady state or avoiding undesirable configurations.

4.4.3 Interpretability of Temporal Specifications

An important advantage of temporal logic is its interpretability. Temporal formulas provide a high-level and declarative way to describe system behavior, making them suitable for domain experts. Reusable patterns and templates further facilitate the specification of common properties.

Chapter 5

Boolean Network Sketches

5.1 Structure of a Boolean Network Sketch **49**
5.1.1 Influence Graph (I) 49
5.1.2 Partially Specified Boolean Network (E) 50
5.1.3 Update Function Properties (Π) 51
5.1.4 Dynamic Properties (Ω) 53
5.2 Specification Language: HCTL **54**
5.2.1 Syntax and Basic Operators 54
5.2.2 Temporal Operators 55
5.2.3 Hybrid Operators 55
5.2.4 Encoding Biological Observations 56
5.3 Capabilities and Limitations **57**

The Boolean Network Sketches framework is a formal instrument for the inference of Boolean networks that unifies prior biological knowledge with experimental observations into a single, coherent model [5]. The central idea is that, instead of committing to a single candidate network, the framework computes the complete set of all Boolean networks consistent with whatever knowledge is available. This set is represented symbolically and can be queried, analysed, and refined as new information becomes available.

A key design principle of the framework is the explicit representation of uncertainty. Rather than making arbitrary choices about uncharacterised regulatory mechanisms, the modeller declares them as unknown, and the framework treats them as free variables to be resolved against the constraints.

Boolean network inference can be approached from two complementary directions [5]. Reverse modelling starts from experimental data and seeks to identify causal structure from measurements, but is fundamentally limited by data volume, noise, and incompleteness. Forward modelling starts from literature-based knowledge of known or hypothesised interactions and constructs a model from first principles, but does not scale well and is prone to the accumulation of inaccuracies when many assumptions are combined.

Neither strategy alone is sufficient for reliable inference of real-world biological networks. The Sketches framework advocates a principled combination of both: it provides a formal language to express what is known from the literature alongside what is known from experimental data, and then systematically determines all models consistent with this combined body of knowledge. The framework is unique in that it computes the *exhaustive* set of consistent candidates rather than sampling from it, which means that downstream analyses such as identifying regulatory functions shared by all valid models, or designing experiments to disambiguate between candidates can be carried out with formal guarantees.

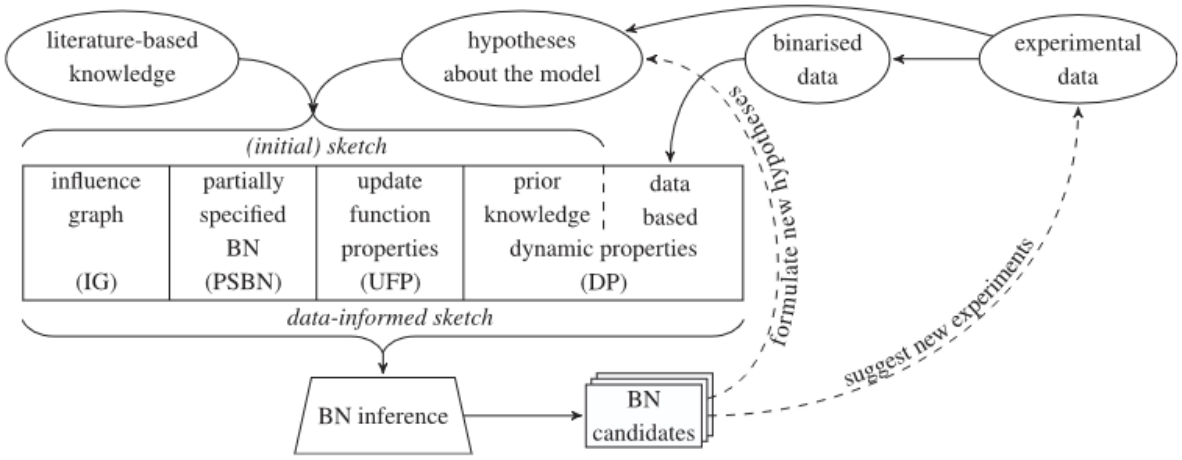


Figure 5.1: Workflow of the Sketches inference method (reproduced from [5]). The initial sketch encodes literature-based prior knowledge. Binarised experimental data are encoded as HCTL formulae and added to yield the data-informed sketch. The inference procedure then computes all consistent candidate Boolean networks.

5.1 Structure of a Boolean Network Sketch

Formally, a Boolean network sketch is defined as a four-tuple:

$$\mathcal{S} = (I,\, E,\, \Pi,\, \Omega)$$

Each component captures a distinct type of knowledge about the biological system under study. The influence graph I defines the permitted regulatory topology. The partially specified Boolean network E encodes what is known about the update logic. The update function properties Π impose functional constraints on the unknown parts of the update logic. Finally, the dynamic properties Ω specify constraints on the behaviour of the network over time, expressed in a temporal logic. A Boolean network F is said to be *consistent* with $\mathcal{S}$ if it simultaneously satisfies all four components.

5.1.1 Influence Graph (I)

The influence graph $I \subseteq \mathrm{Var}_n \times \mathrm{Var}_n$ specifies the permitted regulatory dependencies between network variables. An edge $(v_j, v_i) \in I$ means that v_j is *allowed* to regulate v_i. Crucially, this is a permission, not a requirement: the influence graph does not mandate that all listed interactions are actually used in the update functions, but it does forbid any interaction that is absent from the graph. This allows the modeller to encode the known regulatory topology of a network without over-committing to specific functional relationships that may not be supported by the available data.

In practice, the influence graph is provided as a list of directed edges, where each edge carries a label indicating the type of regulation (activation or inhibition) if this information is known. The following example encodes a small regulatory network in the sketch file format:

```
# Influence graph
x2 ->  x1     # x2 activates x1
x4 ->  x1     # x4 activates x1
x4 -|  x2     # x4 inhibits  x2
x1 ->  x3
x5 ->  x3
x3 -|  x4
x5 ->  x4
```

This fragment specifies that x_2 and x_4 may regulate x_1, that x_4 inhibits x_2, and so on. Any candidate network whose update functions depend on variables not present in the influence graph is immediately excluded from consideration.

5.1.2 Partially Specified Boolean Network (E)

The component E defines the update logic of the network, but does so only partially. Each update function E_i is a *partially specified Boolean expression* that may combine standard Boolean operators with *uninterpreted function symbols* named placeholders representing regulatory mechanisms that have not yet been characterised. An *interpretation* of these symbols assigns a concrete Boolean function to each placeholder, yielding a fully specified candidate network. The inference procedure determines all interpretations that are consistent with the remaining components of the sketch.

This mechanism allows the modeller to encode exactly as much as is known about the update logic and leave everything else open. A fully specified rule leaves no freedom for the inference; a rule containing uninterpreted symbols introduces degrees of freedom that are resolved against the sketch constraints. It is also possible to leave a function entirely unspecified, in which case the inference explores all Boolean functions compatible with the remaining constraints.

The following example illustrates all three cases:

```
# Partially specified update functions
$x1: x2 & x4            # fully specified: x1 = x2 AND x4
$x2: x1 & !x4           # fully specified: x2 = x1 AND NOT x4
$x3: f(x1, x5)          # fully unknown: f is an uninterpreted
                         # function of x1 and x5
$x4: x1 & g(x5)         # partially known: x4 = x1 AND g(x5),
                         # where g is uninterpreted
```

```
$x5:                    # entirely free: no constraint on x5's
                        # update function
```

Here, the update functions for x_1 and x_2 are fully determined. The function for x_4 is partially constrained it must be a conjunction of x_1 with some Boolean function of x_5, but the precise form of that function is left open. The functions for x_3 and x_5 are entirely unspecified, meaning the inference explores all Boolean functions consistent with the influence graph and the remaining sketch components.

5.1.3 Update Function Properties (Π)

The component Π specifies additional constraints on the admissible Boolean update functions, beyond what can be expressed by fixing their syntactic form. These properties restrict the set of valid interpretations of the uninterpreted function symbols without fully determining them, allowing the modeller to encode partial knowledge about the nature of regulatory interactions in a precise, formal way. Each property is a predicate over Boolean functions that can be expressed in first-order logic over Booleans [5, 17].

Importantly, these three properties are not enforced through separate declarations but are instead encoded *structurally*, by shaping the influence graph edges and the syntactic form of the partially specified update functions in the E component. The three properties used in this work are the following.

Monotonicity. Monotonicity is used to assign a sign to each regulatory edge in the network. If a regulator v_i has only a positive effect on v_j, the corresponding edge is declared as an *activation*; if its effect is only negative, it is declared as an *inhibition*. Formally, positive and negative monotonicity are defined as:

$$\text{positive}_i(f) \coloneqq \forall x.\ f(x[i \mapsto 0]) \Rightarrow f(x[i \mapsto 1]),$$

$$\text{negative}_i(f) \coloneqq \forall x.\ f(x[i \mapsto 1]) \Rightarrow f(x[i \mapsto 0]).$$

In the AEON sketch format, this information is encoded directly in the influence graph declaration using signed edge notation. When the sign is known, the edge is written as an activation (`->`) or inhibition (`-|`); when the sign is uncertain, the edge is marked as unspecified (`-?`):

```
x7 ->  x2     # x7 activates x2 (positive monotonicity)
x1 -|  x3     # x1 inhibits x3  (negative monotonicity)
x6 -?  x1     # sign of x6's effect on x1 is unknown
```

Essentiality. A variable v_i is *essential* in the update function of v_j if there exist two states, differing only in the value of v_i, for which the output of the function differs:

$$\text{essential}_i(f) := \exists x \in \mathbb{B}^n.\ f(x[i \mapsto 0]) \oplus f(x[i \mapsto 1]).$$

This captures the notion that v_i genuinely influences v_j, ruling out regulations that are structurally present in the influence graph but functionally inactive. Essentiality is enforced structurally by restricting the *support* of the uninterpreted function symbols in E: rather than allowing a symbolic function to range over all regulators permitted by the influence graph, only the essential regulators are included as its arguments. For example, if x_4 and x_5 are determined to be the essential regulators of x_9, the update function is written as:

```
$x9: f_x9(x4, x5)     # f_x9 depends only on essential
                       # regulators x4 and x5
```

Canalization. A function is *canalizing* with respect to v_i at value b with output o if fixing $v_i = b$ forces the output to o regardless of all other inputs:

$$\text{sufficient}_{i,b,o}(f) := \forall x.\ f(x[i \mapsto b]) \Leftrightarrow o.$$

This captures the biological notion that a dominant input can override the influence of all other regulators. Canalization is also encoded structurally, by replacing the fully uninterpreted symbolic function with a more constrained syntactic template that makes the canalizing relationship explicit. For example, if $x_7 = 1$ is known to force $x_9 = 0$ (i.e. x_7 canalizes x_9 to false), instead of writing a fully unknown function over all three regulators:

```
$x9: f_x9(x7, x6, x19)     # unconstrained
```

the canalization constraint is incorporated directly into the syntactic form of the update function, and the remaining uncertainty is expressed only over the non-canalizing regulators:

```
$x9: !x7 | g_x9(x6, x19)  # x7=1 forces x9=0 (canalization);
                           # g_x9 is free over x6 and x19
```

In this way, the canalizing variable is made explicit in the formula and the uninterpreted part g_{x9} is reduced to a function only of the remaining regulators.

More complex functional behaviours, such as *veto functions* (where a negative regulator overrides all positive ones) and *conditional essentiality* (where a regulator is essential only in the presence of a catalyst), can also be expressed as logical combinations of the above primitives [17].

5.1.4 Dynamic Properties (Ω)

The component Ω specifies constraints on the *dynamics* of the network that is, on how the system evolves over time through its state-transition graph. These properties are expressed in *Hybrid Computation Tree Logic* (HCTL), described in detail in Section 5.2. They encode a wide range of biological observations, including the existence and location of attractors, reachability relationships between states, time-series trajectories, and basins of attraction. A candidate network is consistent with Ω only if its state-transition graph satisfies every formula in the set.

Dynamic properties are typically derived in two ways: from prior biological knowledge (for example, the known existence of a stable apoptotic state), or automatically from binarised experimental data (for example, observed steady-state gene expression profiles encoded as attractor membership constraints).

The following example shows dynamic property declarations in the sketch format, covering the three most common cases encountered in practice:

```
# Dynamic properties (HCTL formulae)

# 1. Fixed-point attractor: all variables inactive (e.g. apoptosis)
#! dynamic_property: fixed_point_off:
#`3{x}: @{x}: ((~x1 & ~x2 & ~x3 & ~x4 & ~x5)
    & AX {x})`#

# 2. Attractor membership: an observed steady state lies
#    in some attractor of the system
#! dynamic_property: observed_attractor:
#`3{x}: @{x}: ((x1 & ~x2 & x3 & ~x4 & x5)
    & AG EF {x})`#

# 3. Reachability: a target state is reachable from
#    an observed initial condition
#! dynamic_property: reachability:
#`3{x}: @{x}: (x1 & ~x2) & EF (~x1 & x2)`#
```

The first formula encodes the existence of a fixed-point attractor where all variables are zero, commonly used to represent a quiescent or cell-death phenotype. The second encodes the requirement that a particular observed state, matching a binarised experimental measurement, lies within some system attractor. The third encodes a reachability constraint derived from a time-series observation: the system can transition from a state where x_1 is active and x_2 is inactive to one where the roles are reversed. The complete running example integrating all four components

is shown in Figure 5.2.

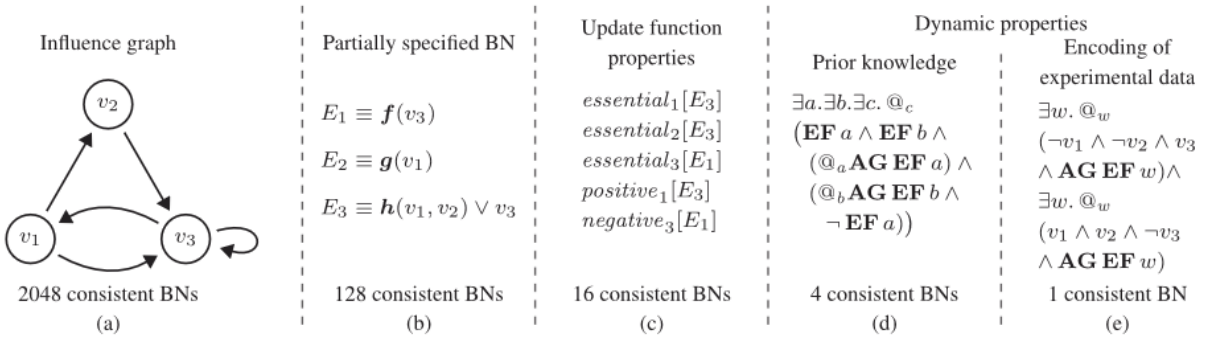


Figure 5.2: Running example of a Boolean network sketch (reproduced from [5], Figure 2). The four panels show, from left to right: the influence graph I, the partially specified Boolean network E, the update function properties Π, and the dynamic properties Ω encoded as HCTL formulae derived from prior knowledge and binarised experimental data.

5.2 Specification Language: HCTL

Dynamic properties in a Boolean network sketch are expressed in *Hybrid Computation Tree Logic* (HCTL), a formal temporal logic that extends the classical branching-time logic CTL with *hybrid operators* that allow explicit reference to and quantification over individual states [5]. HCTL is implemented in the Biodivine HCTL model checker and is the language used to write the Ω component of a sketch.

The key purpose of the language is to describe *what the network should do over time*, independently of how its update functions are constructed. A modeller writes logical formulae that constrain the possible dynamics of the system its attractors, reachable states, trajectories and the inference procedure finds all networks whose state-transition graph satisfies these constraints.

5.2.1 Syntax and Basic Operators

An HCTL formula is built from atomic propositions, logical connectives, temporal operators, and hybrid operators. The full grammar is:

$$\varphi \;::=\; p \;\mid\; x \;\mid\; \neg\varphi \;\mid\; \varphi \wedge \varphi \;\mid\; @_x\, \varphi \;\mid\; \downarrow x.\, \varphi \;\mid\; \exists x.\, \varphi \;\mid\; \mathrm{EX}\, \varphi \;\mid\; \mathrm{E}[\varphi\, \mathcal{U}\, \varphi] \;\mid\; \mathrm{A}[\varphi\, \mathcal{U}\, \varphi]$$

where p ranges over atomic propositions (network variables) and x ranges over state variables. The following standard abbreviations from CTL are also used throughout:

- $\mathrm{AX}\, \varphi \equiv \neg\, \mathrm{EX}\, \neg\varphi$
- $\mathrm{EF}\, \varphi \equiv \mathrm{E}[\mathit{true}\, \mathcal{U}\, \varphi]$
- $\mathrm{AF}\, \varphi \equiv \mathrm{A}[\mathit{true}\, \mathcal{U}\, \varphi]$

- $\mathrm{EG}\,\varphi \equiv \neg\,\mathrm{AF}\,\neg\varphi$
- $\mathrm{AG}\,\varphi \equiv \neg\,\mathrm{EF}\,\neg\varphi$

5.2.2 Temporal Operators

The temporal operators of HCTL allow reasoning about the future evolution of the system through its state-transition graph:

- $\mathrm{EX}\,\varphi$: there *exists* a direct successor state in which φ holds. In asynchronous Boolean networks, a successor is obtained by updating exactly one variable.
- $\mathrm{AX}\,\varphi$: φ holds in *all* direct successor states. Since fixed points have only a self-loop as their successor, $\mathrm{AX}\,\varphi$ at a fixed point reduces to checking whether φ holds in the same state.
- $\mathrm{EF}\,\varphi$: there *exists* a path along which φ eventually holds — i.e. φ is reachable.
- $\mathrm{AF}\,\varphi$: φ is *inevitably* reached along every possible path of the system.
- $\mathrm{EG}\,\varphi$: there *exists* a path along which φ holds forever.
- $\mathrm{AG}\,\varphi$: φ holds in *all* reachable states along all possible paths — a global invariant.
- $\mathrm{E}[\varphi_1\,\mathcal{U}\,\varphi_2]$: there exists a path along which φ_1 holds *until* φ_2 eventually becomes true — the general existential until operator from which EF is derived as a special case.
- $\mathrm{A}[\varphi_1\,\mathcal{U}\,\varphi_2]$: on every path, φ_1 holds until φ_2 eventually becomes true.

These operators are sufficient to express standard biological properties such as reachability (`EF`), stability (`AG`), and inevitable convergence to a particular state (`AF`). However, they are not enough to express properties that require referring back to a specific named state. The canonical example is attractor membership, which requires stating that a state can always eventually return to *itself*. For such properties the hybrid extension is essential.

5.2.3 Hybrid Operators

The hybrid operators of HCTL allow the modeller to *name* specific states during formula evaluation and to *jump back* to named states at a later point. This is the feature that makes HCTL substantially more expressive than standard CTL and particularly well suited to expressing biological properties [5]. The three hybrid operators are:

- $\downarrow x.\,\varphi$ **(bind)**: binds the state variable x to the *current* state and then evaluates φ under this binding. This allows the current state to be remembered and referenced later within the formula.

- $@_x\,\varphi$ **(jump)**: teleports the point of evaluation to the state stored in x and evaluates φ there. This allows reasoning about a specific named state regardless of the current position in the state-transition graph.
- $\exists x.\,\varphi$ **(state quantification)**: asserts that there exists *some* state in the state space that can be bound to x such that φ holds. This is analogous to existential quantification over states.

The canonical example of a property requiring hybrid operators is *attractor membership*. A state belongs to an attractor if and only if, from every state reachable from it, the system can eventually return to it. In HCTL this is written as:

$$\downarrow x.\ \mathrm{AG}\,\mathrm{EF}\,x$$

The bind operator names the current state x; AG quantifies over all future reachable states; and $\mathrm{EF}\,x$ requires that from each such state, x can be reached again. This formula is satisfied in exactly all attractor states of the system, and cannot be expressed in plain CTL.

5.2.4 Encoding Biological Observations

A central use of HCTL in the Sketches framework is the systematic encoding of real-world biological observations as formal constraints on network dynamics [17]. The following patterns cover the most common types of data encountered in practice.

Partial observations. A measurement that fixes k out of n network variables is encoded as a conjunction of literals. An observation where $x_1 = 1$, $x_2 = 0$, and $x_3 = 1$ is written as $\varphi_o \equiv x_1 \wedge \neg x_2 \wedge x_3$, satisfied by all 2^{n-3} states consistent with the measurement.

Attractor membership. The requirement that a partial observation o is present in some system attractor the most frequently used constraint, arising naturally from steady-state gene expression data is encoded as:

$$\varphi_{A(o)} \ \equiv\ \exists x.\ @_x\left(\varphi_o \wedge \mathrm{AG}\,\mathrm{EF}\,x\right)$$

This holds if there exists a state matching o that lies in some attractor.

Fixed-point attractors. The requirement that observation o holds in some fixed point is:

$$\varphi_{FP(o)} \ \equiv\ \exists x.\ @_x\left(\varphi_o \wedge \mathrm{AX}\,x\right)$$

where $\mathrm{AX}\,x$ asserts that the only successor of the current state is itself the defining property of a fixed point under asynchronous semantics.

Reachability from time-series data. A sequence of partial observations $o_1, o_2, \ldots, o_k$ is encoded as a nested reachability formula, without imposing assumptions about what occurs between observations:

$$\varphi_{o_1,\ldots,o_k} \;\equiv\; \varphi_{o_1} \wedge \mathrm{EF}\big(\varphi_{o_2} \wedge \mathrm{EF}\big(\cdots \wedge \mathrm{EF}\,\varphi_{o_k}\big)\big)$$

If the time series is known to occur with certainty rather than merely being possible, EF can be replaced by AF.

Periodic behaviour. When the observed time series represents a single period of a repeating oscillation such as a cell cycle the bind operator allows the formula to require that the final state can return to the initial state, guaranteeing indefinite repeatability:

$$\varphi_{(o_1,\ldots,o_k)^+} \;\equiv\; \downarrow x.\, \Big(\varphi_{o_1} \wedge \mathrm{EF}\big(\cdots \wedge \mathrm{EF}\big(\varphi_{o_k} \wedge \mathrm{EF}\, x\big)\big)\Big)$$

This property cannot be expressed without hybrid operators, as it requires referring back to the initial state by name.

Absence of spurious attractors. A common refinement step is to require that the inferred network has no attractors unrelated to the observed data. If the data supports two attractor states φ_{o_1} and φ_{o_2}, the constraint that no other attractors exist is:

$$\neg\, \exists z.\, @_z\, \neg\, \mathrm{AG}\,\mathrm{EF}\,(\varphi_{o_1} \vee \varphi_{o_2})$$

This rules out all candidate networks that possess attractor states not accounted for by the data.

5.3 Capabilities and Limitations

Capabilities. The Sketches framework offers several important properties that distinguish it from existing Boolean network inference methods:

- **Exhaustiveness.** The inference procedure computes the complete set of all consistent Boolean networks, guaranteeing that no valid candidate is missed and enabling reliable downstream analyses such as identifying update functions shared by all valid models.
- **Expressiveness.** The combination of partial update functions, functional properties (Π), and HCTL dynamic properties (Ω) provides a substantially richer specification language than comparable methods. Both the Griffin tool [38] and the RE:IN framework [39] are formally subsumed as special cases of the sketch formalism [17].

- **Integration of heterogeneous knowledge.** Structural knowledge from the literature, functional hypotheses, and experimental data are all accommodated within the same formal model and processed in a single inference step.
- **Iterative refinement.** Because the result of inference is a symbolic set that can be queried and further constrained, the framework naturally supports an iterative workflow: candidates are analysed, the sketch is extended with new hypotheses, and inference is re-run, with each iteration reusing results from the previous one.

Limitations.

- **Exponential state-space growth.** The state space of a Boolean network grows exponentially in the number of variables, and performance degrades for networks with high average regulatory in-degree or a large number of uncharacterised update functions.
- **Dependence on input quality.** Vague constraints may yield an intractably large candidate set, while contradictory constraints yield an empty set, indicating that the available knowledge is mutually incompatible.
- **Binarisation requirement.** The framework assumes that all experimental data have already been binarised prior to encoding. The quality of this preprocessing step can significantly affect the biological validity of the inferred candidates.

Chapter 6

Pipeline Overview

6.1 Main Objective **60**
6.2 Design and Architecture **61**
6.3 Pipeline Stages **62**
6.3.1 Stage 1: Boolean Network Generation 63
6.3.2 Stage 2: Trace Simulation 64
6.3.3 Stage 3: Trace Conversion to Dynamic Properties 65
6.3.4 Stage 4: Structural Sketch Construction 66
6.3.5 Stage 5: Analysis of Long-Term Dynamical Behaviour 69
6.3.6 Stage 6: Conversion of Dynamical Analyses to Sketch Properties 69
6.3.7 Stage 7: Sketch Assembly 72
6.4 Pipeline Capabilities and Configuration **75**

This chapter presents the pipeline developed in this thesis for the fully configurable creation of Boolean network sketches ready for inference with the Sketches framework [5]. Rather than acting as a simple conversion script, the pipeline serves as a configurable experimental tool that constructs sketches with precisely chosen combinations of structural, functional, and dynamical information. The chapter introduces the purpose of the pipeline, the motivation behind its design, the kinds of sketches it can generate, and the main processing stages through which a source Boolean network is transformed into an inference-ready sketch.

6.1 Main Objective

The main objective of the pipeline is to enable the systematic and fully configurable generation of Boolean network sketches that are ready to be used as input to the Sketches inference tool. Starting from a fully specified Boolean network, the pipeline produces an `.aeon` sketch in which the included information is not fixed in advance, but selected by the user through configuration.

More specifically, the pipeline was designed to allow controlled construction of sketches with different amounts and different types of prior knowledge. By adjusting the parameters of its individual stages, the user can determine what information will be included in the final sketch. For example, one may generate a sketch for a network with a chosen number of variables, restrict the generated network to acyclic topologies, reveal only a given percentage of the structure, or enforce specific functional constraints such as monotonicity, essentiality, or canalization. In the same way, one may include or exclude different classes of dynamical observations, such as trace-based reachability properties, attractor candidates, fixed points, or trap spaces.

The pipeline is therefore best understood as a sketch-generation tool rather than a fixed workflow. Its purpose is to support the creation of a wide range of inference-ready sketches under controlled conditions, making it possible to vary the available information in a precise and reproducible way.

This capability is useful for several purposes. First, it supports systematic experimentation on the Sketches inference framework by allowing one to study how different forms of prior knowledge affect inference performance, solution quality, or ambiguity. Second, it enables the generation of benchmark families of sketches with gradually increasing or decreasing information content. Third, it provides a practical way to model different realistic knowledge scenarios, ranging from very limited structural knowledge to richer settings where functional constraints and dynamical observations are available. More broadly, the pipeline offers a unified environment for exploring how structural, functional, and dynamical information can be combined when constructing Boolean network sketches for inference.

For this reason, the pipeline should not be regarded merely as a converter from Boolean networks

to sketches. Rather, it constitutes a configurable framework for the construction of inference-ready sketch instances, intended to support controlled experimentation, comparative evaluation, and the systematic study of the role played by different information classes in the Boolean network inference problem. Its relevance, however, extends beyond methodological analysis. The pipeline may also be viewed as a practical instrument for assessing the adequacy of available knowledge in concrete modelling settings. In particular, a modeller investigating a biological regulatory system may possess only partial structural information, limited evidence concerning functional constraints such as monotonicity, essentiality, or canalization, and a restricted set of dynamical observations. In such cases, the pipeline makes it possible to construct sketches that reflect different assumptions about the available prior knowledge and to evaluate whether this knowledge is sufficient to meaningfully constrain the inference task. Conversely, it enables the systematic examination of which additional forms of information would most improve the realism and identifiability of the inferred models. In this sense, the pipeline provides not only a means of generating benchmark instances, but also a framework for studying the relationship between available biological knowledge and the computational tractability and informativeness of Boolean network inference. All behaviour is specified through human-readable configuration files, allowing sketches to be generated and varied without modification of the underlying implementation.

6.2 Design and Architecture

The pipeline was designed as a modular, stage-oriented framework. Rather than implementing sketch construction as a single monolithic transformation, the workflow is decomposed into seven well-defined processing stages, each implemented as a dedicated script under `src/` and coordinated by a top-level entry point, `run_pipeline.py`. This design separates concerns cleanly across the workflow: each stage has a single well-defined responsibility, consumes a set of inputs, and produces a set of intermediate artefacts that are consumed by later stages.

A key architectural objective is configurability. Each stage is driven by its own configuration file, and the top-level `configs/pipeline.txt` controls both which stage configurations are used and which families of information are ultimately included in the final sketch. As a result, the pipeline can generate sketches spanning a wide spectrum of informativeness from minimally revealed structural skeletons to fully enriched sketches carrying both structural and dynamical constraints without any changes to the source code.

The pipeline combines several external tools that operate at different levels of the workflow. Boolean network generation uses `BoolForge`, which produces `.bnet` models from a declarative YAML specification. Trace simulation is performed in R using the `BoolNet` package. Attractor analysis uses `bioLQM` [40], a Java-based tool for the computation of fixed points and trap

spaces of logical qualitative models. The outputs of these tools are translated by the pipeline's Python scripts into a common `.aeon` sketch representation whose structure is compatible with the Sketches inference tool.

Stage	Tool	Role
Generation	BoolForge	Create ground-truth network
Simulation	BoolNet	Generate traces
Analysis	bioLQM	Compute fixed points / trap spaces
Reconstruction	Sketches	Infer consistent networks
Comparison	BoNesis	Alternative constraint-based inference

Table 6.1: Tools used in the experimental pipeline

The final output of the pipeline is a single `.aeon` sketch file assembled from three independently produced parts: the structural model section (`## MODEL`), the trace-derived dynamic properties, and the attractor-related dynamic properties. Which parts are included in the final sketch is controlled by boolean flags in `configs/pipeline.txt`.

6.3 Pipeline Stages

The complete flow of the pipeline is illustrated in Figure 6.1. The seven stages fall naturally into three groups: network generation and simulation (Stages 1–2), extraction of sketch components (Stages 3–6), and final assembly (Stage 7). Each stage is described in detail below.

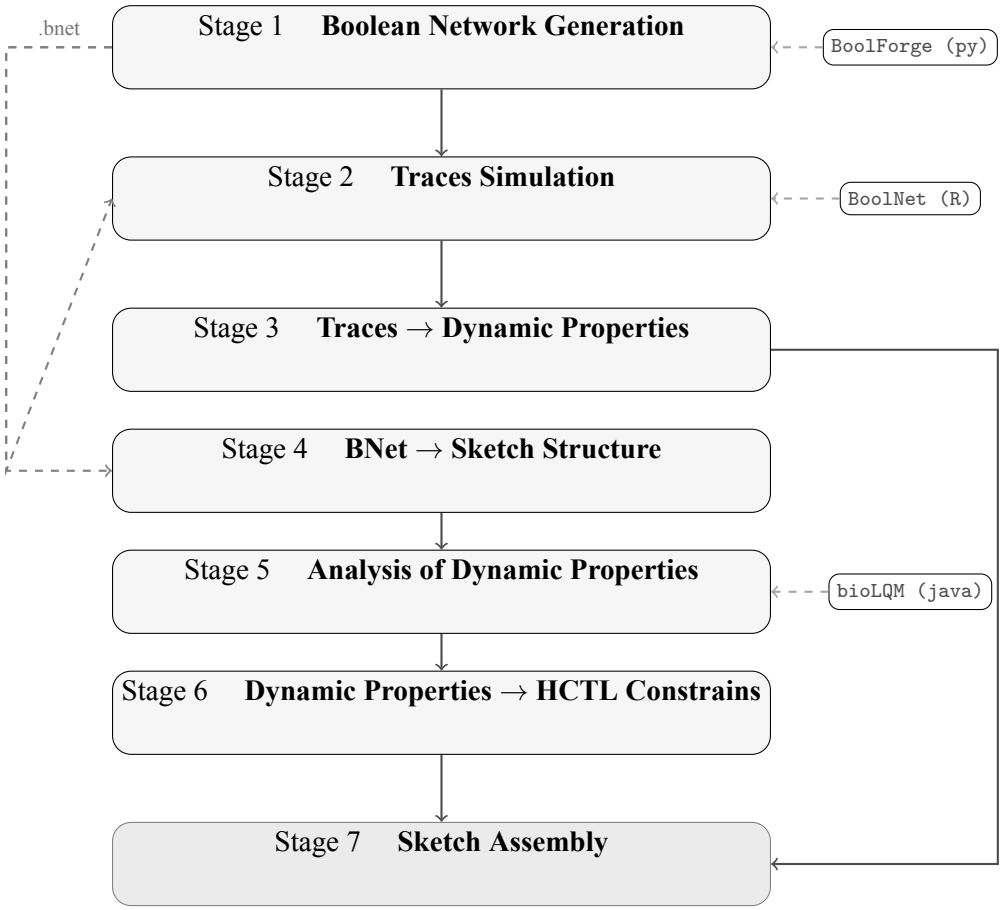


Figure 6.1: Architecture of the sketch generation pipeline. Solid arrows denote the primary data flow; dashed arrows indicate tool invocations and the direct reuse of the `.bnet` file by Stage 4. The final `.aeon` sketch is assembled in Stage 7 from the structural model part (Stage 4) and the two families of dynamic properties (Stages 3 and 6). Which property families are included is controlled by flags in `configs/pipeline.txt`.

6.3.1 Stage 1: Boolean Network Generation

The first stage establishes the fully specified source Boolean network from which all subsequent information is derived. In the default workflow, this is achieved using `BoolForge`, which generates a Boolean network from a declarative configuration and exports it in `.bnet` format. The output of this stage is therefore a complete Boolean network containing explicit update rules for all variables.

Conceptually, this stage defines the ground-truth model underlying the entire pipeline. All subsequent stages depend either directly or indirectly on the network produced here. In particular, the trace simulation stage uses this `.bnet` model as the basis for generating state trajectories, while the structural sketch-construction stage reuses the same model in order to derive the partially revealed `## MODEL` component of the final sketch. Accordingly, Stage 1 determines the structural and dynamical substrate from which all later information is extracted.

In addition to network size and rule-generation parameters, this stage permits control over topological properties of the generated model. For example, the user may constrain the source network to satisfy structural conditions such as the absence of self-regulation, strong connectivity, or acyclicity. The latter is particularly important in experimental settings where one wishes to compare the effect of information classes under controlled assumptions about the class of source networks being considered. Alternatively, the generation step may be bypassed entirely by supplying an already existing `.bnet` file to the pipeline, in which case the remainder of the workflow proceeds from that model.

Table 6.2: Main configuration options used in Stage 1 (`configs/boolforge.yaml`).

Configuration key	Purpose
`output`	Path of the generated `.bnet` file.
`N`	Number of variables in the generated Boolean network.
`n`	Maximum in-degree used when constructing the regulatory graph.
`indegree_distribution`	Distribution governing regulator counts.
`bias`	Bias used during rule generation.
`depth`	Logical depth of generated update functions.
`NO_SELF_REGULATION`	Prevents variables from regulating themselves.
`STRONGLY_CONNECTED`	Requests generation of a strongly connected interaction graph.
`AT_LEAST_ONE_REGULATOR_PER_NODE`	Ensures that each variable has at least one regulator.
`acyclic`	Requests an acyclic interaction structure.
`acyclic_method`	Selects the strategy used to construct the acyclic graph.
`rng`	Random seed for reproducible generation.

6.3.2 Stage 2: Trace Simulation

The second stage derives simulated dynamical observations from the Boolean network established in Stage 1. Using the `.bnet` model produced previously, the pipeline invokes `BoolNet` to generate state trajectories under a chosen update semantics. The primary output of this stage is thus a collection of trace files representing sequences of states visited by the source network

under simulation.

This stage depends directly on the output of Stage 1: without the fully specified Boolean network, no simulation can be performed. In this sense, Stage 2 constitutes the first transformation of complete mechanistic knowledge into a more realistic observational form. Rather than exposing the update rules themselves, it produces only partial dynamical evidence in the form of observed state sequences. These observations are then passed to Stage 3, where they are converted into formal sketch constraints.

The traces generated here may be interpreted as an abstraction of experimental or observed time-series information. By varying the number of traces, their length, the update mode, and the amount of stochastic perturbation, the user can simulate differing observational regimes, ranging from sparse and noisy evidence to more informative trajectory sets. This flexibility is important because the informativeness of the trace-derived properties generated later depends directly on the character of the simulation data established at this stage.

Table 6.3: Main configuration options used in Stage 2 (`configs/traces.txt`).

Configuration key	Purpose
`num_traces`	Number of trajectories to generate.
`num_steps`	Number of states recorded in each trajectory.
`update_type`	Choice of update semantics, e.g. synchronous or asynchronous.
`noise_level`	Noise level used during simulation.
`seed`	Random seed for reproducible simulations.
`output_dir`	Directory in which trace files are written.
`output_prefix`	Prefix used for naming trace files.
`output_suffix`	Suffix used for naming trace files.
`write_trajectory_header`	Controls whether trace files contain an explicit header.
`write_genes_file`	Controls whether a file containing the variable names is also produced.

6.3.3 Stage 3: Trace Conversion to Dynamic Properties

The third stage transforms the raw trajectories produced in Stage 2 into formal dynamic properties suitable for inclusion in a Boolean network sketch. The input to this stage is therefore not the source network directly, but the simulated observational data derived from it in the preceding stage. The output is a sketch-compatible properties file expressing reachability-related and trace-derived asymptotic constraints.

This dependency is important at the conceptual level. Whereas Stage 2 generates state sequences from the fully specified source model, Stage 3 reinterprets those sequences as partial prior knowledge that may constrain inference. The information thus becomes progressively more abstract: complete update rules are first turned into traces, and traces are then turned into logical properties. The resulting properties can encode individual reachability relations, longer chains of observed transitions, endpoint relations, or selected sparse milestones. In addition, terminal repetition patterns in traces may be used to derive candidate information about recurrence or cyclic behaviour.

The stage may also reduce the dimensionality of the derived properties by projecting observed states onto subsets of variables before formula generation. This permits the construction of sketches whose dynamic constraints reflect only partial state information, which is often more realistic in practical modelling contexts. The set of properties produced in this stage is passed unchanged to the final assembly stage, where it forms one of the two dynamic-property families available for inclusion in the final sketch.

Table 6.4: Main configuration options used in Stage 3 (`configs/trace_properties.txt`).

Configuration key	Purpose
`traces_dir`	Directory containing the traces produced in Stage 2.
`output`	Path of the generated sketch-property snippet.
`genes`	Optional path to the file listing the variables.
`trace_glob`	Pattern used to select trace files.
`pair_mode`	Strategy used to derive reachability constraints from traces.
`keep_percent`	Fraction of generated properties retained after sampling.
`partial_state_size`	Number of variables retained when projecting states.
`partial_state_mode`	Strategy for selecting the retained variables.
`seed`	Random seed for property sampling or variable selection.
`compress_stutter`	Removes consecutive duplicate states prior to analysis.
`property_prefix`	Naming prefix for reachability properties.
`fixed_point_prefix`	Naming prefix for singleton-tail recurrence candidates.
`cycle_prefix`	Naming prefix for cycle candidates.
`cycle_lengths`	Cycle lengths considered when detecting repeated patterns.

6.3.4 Stage 4: Structural Sketch Construction

Stage 4 constructs the structural component of the final sketch. Its input is the same `.bnet` model established in Stage 1, but its role is distinct from that of Stage 2. Whereas Stage 2 uses the source network to generate observational dynamical data, Stage 4 uses it to derive a partial

structural and functional description suitable for the `## MODEL` section of the sketch. The output of this stage is therefore the sketch model component itself.

The central operation carried out here is controlled revelation. Rather than copying the source model directly into the sketch, the stage determines how much of the original interaction structure and update logic should remain visible. Some target functions may be revealed exactly, in which case their Boolean rules are preserved in the sketch without abstraction. For other targets, only the presence of regulators may be retained, while the update rule itself is replaced by a symbolic placeholder. Still other targets may be left largely or entirely unrevealed, depending on the selected hiding policy. In this way, the stage constructs a partially specified model whose degree of structural informativeness is explicitly configurable.

This stage may also enrich the sketch with functional constraints inferred from the exact rules available in the source model. In particular, when such analysis is enabled, exactly revealed rules may be used to infer monotonicity-related sign information, to determine which regulators are essential for the target function, and to detect canalizing dependencies. These derived properties are then encoded into the structural component of the sketch either as constrained edge annotations, restricted symbolic supports, or canalization-aware symbolic templates. Stage 4 thus occupies a central position in the pipeline: it is the stage at which the source model is transformed into the partially known structural object that constitutes the core of the final sketch.

Table 6.5: Main configuration options used in Stage 4 (`configs/structure.txt`).

Configuration key	Purpose
`bnet`	Path to the `.bnet` model produced in Stage 1.
`output`	Path of the generated `## MODEL` snippet.
`reveal_functions_percent`	Fraction of target functions for which regulator support is revealed.
`reveal_regulators_percent`	Fraction of regulators revealed for support-revealed targets.
`reveal_exact_functions_percent`	Fraction of target functions copied exactly into the sketch.
`seed`	Random seed controlling reveal choices.
`edge_op`	Default edge symbol used for unconstrained interactions.
`hidden_policy`	Policy for representing targets whose structure remains hidden.
`infer_monotonicity_for_exact`	Enables sign inference for exactly revealed rules.
`infer_essentiality`	Enables detection of essential and non-essential regulators.
`apply_essentiality_to_symbolic_supports`	Restricts symbolic supports to essential regulators only.
`annotate_essentiality_comments`	Adds essentiality annotations to the structural output.
`infer_canalization_for_exact`	Enables canalization analysis for exactly revealed rules.
`apply_canalization_templates`	Replaces generic symbolic rules by canalization-aware templates when possible.
`annotate_canalization_comments`	Adds canalization annotations to the structural output.

6.3.5 Stage 5: Analysis of Long-Term Dynamical Behaviour

The fifth stage derives exact attractor-related information from the source Boolean network. Like Stage 2 and Stage 4, it depends directly on the `.bnet` model established in Stage 1; however, it extracts a different kind of knowledge from that model. Whereas Stage 2 produces sampled dynamical observations and Stage 4 produces a partially revealed structural representation, Stage 5 computes formal descriptions of long-term dynamical behaviour using `bioLQM` [40]. In the current pipeline, this includes fixed-point analysis and trap-space analysis.

The significance of this stage lies in the fact that it derives exact dynamical objects from the source model itself, rather than from finite simulated traces. It therefore provides a complementary form of prior knowledge to that produced in Stages 2 and 3. The outputs of this stage are not yet directly incorporated into the sketch. Instead, they serve as intermediate analysis artefacts that are subsequently transformed into sketch properties in Stage 6.

In this way, the pipeline maintains a clear distinction between raw analytical results and sketch-level constraints. Stage 5 is therefore best understood as an intermediate analysis stage whose role is to compute exact dynamical summaries from the source network so that these summaries may later be recast in a form suitable for inference.

Table 6.6: Main configuration options used in Stage 5 (`configs/dynamics.txt`).

Configuration key	Purpose
`bnet`	Path to the `.bnet` model produced in Stage 1.
`fixpoints_output`	Path of the file containing fixed-point analysis results.
`trapspaces_output`	Path of the file containing trap-space analysis results.
`biolqm_cmd`	Command used to invoke `bioLQM`.
`java_cmd`	Java executable used when required by the analysis back-end.
`biolqm_jar`	Optional path to the `bioLQM` archive.
`skip_fixpoints`	Disables fixed-point analysis.
`skip_trapspaces`	Disables trap-space analysis.

6.3.6 Stage 6: Conversion of Dynamical Analyses to Sketch Properties

Stage 6 converts the analysis artefacts produced in Stage 5 into formal dynamical properties that can be included in the sketch. Its input is therefore not the source network directly, but the fixed points and trap spaces computed from that network in the preceding stage. This stage mirrors, at a conceptual level, the role played by Stage 3: both stages transform derived dynamical evidence into sketch-level logical constraints, but they do so from different sources of prior knowledge.

The properties generated here encode exact attractor-related information in a form that is compatible with the final sketch representation. Fixed points are translated into constraints corresponding to stable states, while trap spaces are translated into constraints describing recurrent or dynamically closed state patterns. In contrast to the trace-derived properties of Stage 3, which are based on sampled trajectories, the properties produced here originate from exact analysis of the source network. The two families are therefore complementary and capture distinct aspects of dynamical knowledge.

For a fixed point with state pattern p, the generated HCTL formula has the general form

$$\exists x.\ @_x\big(p \wedge AX(p)\big),$$

which states that there exists a state satisfying p, and that all immediate successors of that state also satisfy p. Trap-space properties are encoded using the related form

$$\exists x.\ @_x\big(p \wedge AG\,EF(p)\big),$$

which requires the pattern p to hold at some state and to remain recurrently reachable from all future states.

An additional option introduced in this stage is the generation of constant-depth dynamical properties. When this option is enabled, the pipeline does not only analyse the original network. It also constructs temporary subnetworks in which one or more variables are fixed to constants. For example, for a network with variables x_1, x_2, x_3, a depth-one constant context may fix $x_1 = 1$, producing a temporary network in which the update rule of x_1 is replaced by the constant value 1. The fixed points and trap spaces of this clamped network are then computed with `bioLQM`.

The results from such clamped subnetworks cannot be inserted directly as ordinary fixed-point or trap-space properties of the full network, because the clamped network and the original network have different update rules. Instead, they are lifted back into the original sketch as conditional HCTL properties. For example, suppose the subnetwork obtained by fixing $x_1 = 1$ has a fixed point with pattern

$$p = x_1 \wedge \neg x_2 \wedge x_3.$$

The corresponding lifted property is written as

$$(\exists c.\ @_c AG(x_1)) \Rightarrow (\exists x.\ @_x(AG(x_1) \wedge p \wedge AX(p))).$$

In the concrete syntax used by the pipeline, this has the form

```
#! dynamic_property: constant_x1_is_1_fixed_point_1:
#`(3{c}: (@{c}: AG (x1))) =>
```

```
  (3{x}: ( @{x}: ( (AG (x1)) &
                   (x1 & ~x2 & x3) &
                   (AX ((x1 & ~x2 & x3))) ) ))`#
```

The left-hand side states that the original full network has some state from which x_1 remains true forever. The right-hand side then requires the fixed-point pattern found in the $x_1 = 1$ clamped subnetwork to exist inside such an x_1-stable region. Thus the constraint does not force x_1 to be true globally. It only applies if the full network naturally admits a region in which x_1 remains true.

Trap-space results from clamped subnetworks are lifted in the same way, but with $AG\,EF(p)$ replacing $AX(p)$. For the same pattern p, the lifted trap-space property is

$$(\exists c.\ @_c AG(x_1)) \Rightarrow (\exists x.\ @_x(AG(x_1) \wedge p \wedge AG\,EF(p))).$$

In concrete syntax:

```
#! dynamic_property: constant_x1_is_1_trap_space_1:
#`(3{c}: (@{c}: AG (x1))) =>
  (3{x}: ( @{x}: ( (AG (x1)) &
                   (x1 & ~x2 & x3) &
                   (AG EF ((x1 & ~x2 & x3))) ) ))`#
```

The depth of this construction is controlled by the parameter `constant_depth`. If `constant_depth = 1`, only the original network is analysed, corresponding to the previous behaviour of the pipeline. If `constant_depth = 2`, all one-variable clamped subnetworks are also analysed, such as $x_1 = 0$, $x_1 = 1$, $x_2 = 0$, and $x_2 = 1$. If `constant_depth = 3`, all two-variable clamped subnetworks are included as well, such as $x_1 = 0, x_2 = 1$. For a multi-variable clamp, the guard is the conjunction of the fixed values, for example

$$AG(\neg x_1 \wedge x_2).$$

Because the number of clamped subnetworks grows combinatorially with the number of variables and the chosen depth, the pipeline also provides the parameter `constant_subnetwork_limit`. This parameter limits how many clamped subnetworks are analysed and therefore allows constant-depth properties to be used in larger examples without making the analysis prohibitively expensive.

The output of Stage 6 is passed directly to the final assembly stage, where it forms the second major family of dynamical constraints that may be selected for inclusion in the final sketch.

Table 6.7: Main configuration options used in Stage 6 (`configs/dynamics_properties.txt`).

Configuration key	Purpose
`fixpoints`	Path to the fixed-point results produced in Stage 5.
`trapspaces`	Path to the trap-space results produced in Stage 5.
`output`	Path of the generated sketch-property snippet.
`mode`	Selects whether fixed points, trap spaces, or both are converted.
`property_prefix_fixed`	Naming prefix for fixed-point properties.
`property_prefix_trap`	Naming prefix for trap-space properties.
`start_index`	Initial numbering index for generated properties.
`include_forbid_extra`	Controls whether additional exclusion constraints are emitted.
`no_dedup`	Disables duplicate removal.
`no_properties_header`	Omits the property-section header in the output.
`bnet`	Path to the original `.bnet` model used for constant-depth analysis.
`constant_depth`	Controls the depth of clamped-subnetwork analysis. A value of 1 keeps only the original network; 2 adds one-variable clamps; 3 adds two-variable clamps.
`constant_subnetwork_limit`	Maximum number of clamped subnetworks analysed. A value of 0 means no limit.
`constant_property_prefix`	Naming prefix for lifted constant-depth properties.
`biolqm_cmd`	Command used to invoke `bioLQM` for clamped subnetworks.

6.3.7 Stage 7: Sketch Assembly

The final stage assembles the complete sketch from the components produced in the preceding transformation stages. It consumes the structural model component generated in Stage 4, the trace-derived property component generated in Stage 3, and the attractor-related property component generated in Stage 6. In this sense, Stage 7 is the unique point in the pipeline at which the distinct information streams established earlier in the workflow are brought together into a single inference-ready artefact.

The role of this stage is not to derive new information, but to determine how the previously derived information classes are combined. The resulting sketch always contains the structural model component produced in Stage 4, since this constitutes the core of the sketch represen-

tation.By contrast, the various families of dynamical properties may be included or excluded selectively. These include trace-derived reachability properties, trace-derived recurrence candidates, fixed-point and trap-space properties computed from the original network, and lifted constant-depth properties obtained from clamped subnetworks. This means that the final sketch can be constructed so as to contain only trace-derived evidence, only exact attractor-related evidence, both, or neither, depending on the intended experimental design or modelling scenario.

The output of this stage is a single `.aeon` file ready to be used as input to the Sketches inference framework. It is therefore the terminal product of the entire workflow and the point at which all preceding structural and dynamical derivations become available as a unified sketch instance.

Table 6.8: Main configuration options used in Stage 7 (`configs/pipeline.txt`).

Configuration key	Purpose
`boolforge_config`	Configuration file used in Stage 1 for network generation.
`trace_config`	Configuration file used in Stage 2 for trace simulation.
`traces_properties_config`	Configuration file used in Stage 3 for trace-to-property conversion.
`structure_config`	Configuration file used in Stage 4 for structural sketch construction.
`biolqm_dynamics_config`	Configuration file used in Stage 5 for attractor-related analysis.
`biolqm_properties_config`	Configuration file used in Stage 6 for converting analyses to sketch properties.
`combined_sketch_output`	Path of the final assembled sketch.
`existing_bnet`	Optional pre-existing `.bnet` model used instead of Stage 1 generation.
`skip_attractor_properties`	Skips Stages 5 and 6.
`skip_combine`	Stops execution before final assembly.
`include_trace_reachability_properties`	Includes reachability properties derived from Stage 3.
`include_trace_attractor_candidate_properties`	Includes trace-based recurrence candidates from Stage 3.
`include_trace_cycle_candidate_properties`	Includes cycle candidates derived from Stage 3.
`include_biolqm_fixed_point_properties`	Includes fixed-point properties derived from Stage 6.
`include_biolqm_trap_space_properties`	Includes trap-space properties derived from Stage 6.
`include_essentiality_structure_constraints`	Activates essentiality-related structural enrichment in Stage 4.
`include_canalization_structure_annotations`	Activates canalization-related structural enrichment in Stage 4.

6.4 Pipeline Capabilities and Configuration

A defining characteristic of the pipeline is the extent to which the content of the final sketch can be controlled through configuration. This configurability may be understood along two principal dimensions. The first concerns the degree to which the original structure and update logic of the source network are revealed in the sketch. The second concerns the degree to which the sketch is enriched with additional structural, functional, and dynamical constraints derived from analysis of the source model.

The first dimension may be described as the level of structural and functional revelation. By varying the proportions of exactly revealed functions, support-revealed functions, and visible regulators, the user can determine how closely the `## MODEL` section resembles the source Boolean network. At one extreme, the sketch may contain only very weak structural information; at the other, substantial parts of the update logic may be retained explicitly. In this way, the pipeline supports the systematic generation of sketches spanning a continuum from highly underspecified to highly informative structural instances.

The second dimension concerns enrichment by additional constraints. Functional properties such as monotonicity, essentiality, and canalization may be enforced through the structural transformation stage, while dynamical information may be introduced through either sampled traces or exact attractor-related analysis. Because these information classes are generated separately and combined only at the final stage, they can be studied independently or in combination. This makes it possible to construct sketches that differ not only in how much of the source model is revealed, but also in what kind of auxiliary prior knowledge is made available to the inference procedure.

Taken together, these two dimensions define the principal design space explored by the pipeline. By varying the revelation level and the enrichment level in a controlled manner, one can generate families of inference-ready sketches suited to benchmark construction, comparative experimentation, or modelling studies in which the contribution of particular information classes must be assessed systematically.

Chapter 7

Experimental Setup, Results, and Evaluation

7.1 Experimental Setup **77**
7.1.1 Research Objectives 77
7.1.2 Construction of Experimental Sketch Families 77
7.1.3 Experimental Dimensions 78
7.1.4 Inference Procedure 79
7.1.5 Evaluation Metrics 79
7.2 Experimental Results **79**
7.3 Evaluation **81**
7.3.1 Effect of Structural and Functional Information 82
7.3.2 Impact of Dynamic Constraints 82
7.3.3 Influence of Exact Function Revelation 82
7.3.4 Scalability and Overall Trade-Off 83

This chapter presents a systematic experimental evaluation of the sketch-generation pipeline introduced in Chapter 6. The goal of the study is to examine how different forms and amounts of prior knowledge affect Boolean network inference when that knowledge is encoded as a sketch.

More specifically, the experiments investigate how progressively enriching a sketch with structural, functional, and dynamical information influences both the computational cost of inference and the size of the resulting candidate space. To support this analysis, families of sketches were constructed in a controlled and cumulative way, so that the contribution of each additional layer of information can be examined separately.

The remainder of this chapter is organised as follows. Section 7.1 describes the experimental design, including the research objectives, the construction of sketch families, the inference procedure, and the evaluation metrics. The subsequent sections present and analyse the experimental results.

7.1 Experimental Setup

7.1.1 Research Objectives

The objective of the experimental study is to analyse how the Boolean network inference problem changes as the prior knowledge encoded in a sketch becomes progressively richer. In particular, the experiments were designed to evaluate the effect of adding different classes of structural and dynamical information while also varying the proportion of local update functions that remain exactly revealed.

The experimental study is guided by the following research questions:

- **RQ1:** How does increasing structural and functional information affect the size of the candidate network space?
- **RQ2:** What is the impact of dynamic constraints, including trace-based and attractor-based properties, on the inference problem?
- **RQ3:** How does the proportion of exactly revealed local update functions influence inference complexity?
- **RQ4:** How does inference runtime scale with network size and sketch informativeness?

For each source Boolean network, the experimental design generates a family of related sketches ordered by increasing informativeness. Each successive sketch class therefore corresponds to a more constrained reconstruction problem. This makes it possible to evaluate the contribution of each layer of information both in terms of computational behaviour and in terms of restriction of the compatible candidate space.

The central hypothesis is that richer sketches should reduce the number of compatible candidate networks by imposing stronger constraints, while at the same time potentially increasing the computational cost of inference because those constraints are more demanding to enforce.

7.1.2 Construction of Experimental Sketch Families

Using the pipeline described in Chapter 6, a total of 360 experimental sketches were generated. Each sketch is derived from a fully specified source Boolean network and encodes a selected subset of its structural, functional, and dynamical properties. The construction process is governed by two main experimental dimensions: the size of the underlying Boolean network and the percentage of exactly revealed local update functions.

For each source network, a sequence of sketch classes was constructed such that informativeness increases cumulatively. This design enables controlled comparison between sketches that differ by exactly one additional layer of information.

The following six sketch classes were considered:

- `S0_exact`: baseline sketch containing partial structural revelation and a fixed proportion of exactly specified local update functions.
- `S1_monotonicity`: extends the baseline by adding monotonicity constraints on regulatory interactions.
- `S2_essentiality`: further incorporates essentiality constraints, specifying which regulators have a decisive influence on update functions.
- `S3_canalization`: augments the sketch with canalization properties, capturing dominant regulatory relationships.
- `S4_reachability`: introduces dynamic constraints derived from simulated traces in the form of reachability properties.
- `S5_biolqm`: adds attractor-related constraints, including fixed points and trap spaces, computed using `bioLQM`.

This cumulative structure ensures that each successive sketch class strictly increases the amount of encoded knowledge. Consequently, observed differences in inference outcomes can be attributed directly to the additional constraints introduced at each stage.

7.1.3 Experimental Dimensions

The experimental study explores three main dimensions:

1. **Network size:** source Boolean networks were generated with sizes ranging from 5 to 50 variables, in increments of 5, resulting in ten size categories.

2. **Reveal level:** the proportion of exactly revealed local update functions was varied across six levels, from 40% to 90%.
3. **Sketch class:** the six sketch classes described above represent increasing levels of informational richness.

Combining these dimensions yields the full experimental dataset:

$$6 \text{ sketch classes} \times 6 \text{ reveal levels} \times 10 \text{ network sizes} = 360 \text{ sketches}.$$

This design enables systematic comparisons along each axis while controlling for the others. In particular, it allows the separate assessment of structural revelation, functional constraints, and dynamical properties.

7.1.4 Inference Procedure

Each generated sketch was submitted to the inference engine in order to compute the set of Boolean networks that satisfy the encoded constraints. All runs were executed in batch mode under fixed configuration settings to ensure consistency across the experimental study.

For each sketch instance, the solver returned:

- the number of compatible candidate networks,
- up to ten witness networks,
- diagnostic information describing the solving process.

The inference results were collected and stored in a structured format to support systematic analysis across all experimental dimensions.

7.1.5 Evaluation Metrics

For each inference run, the following metrics were recorded:

- **Execution time:** the total runtime of the inference procedure, used to assess computational scalability.
- **Candidate space size:** the number of Boolean networks consistent with the sketch, used as a measure of how much ambiguity remains after the constraints are applied.
- **Witness networks:** representative solutions returned by the solver, used for qualitative inspection.

The candidate space size serves as the primary measure of constraint effectiveness, while execution time captures the computational cost of enforcing those constraints. Together, these metrics provide a direct view of the trade-off between sketch informativeness and inference complexity.

To support interpretation, the results are visualised using scatter plots and comparative charts that show how each metric changes across network sizes, reveal levels, and sketch classes.

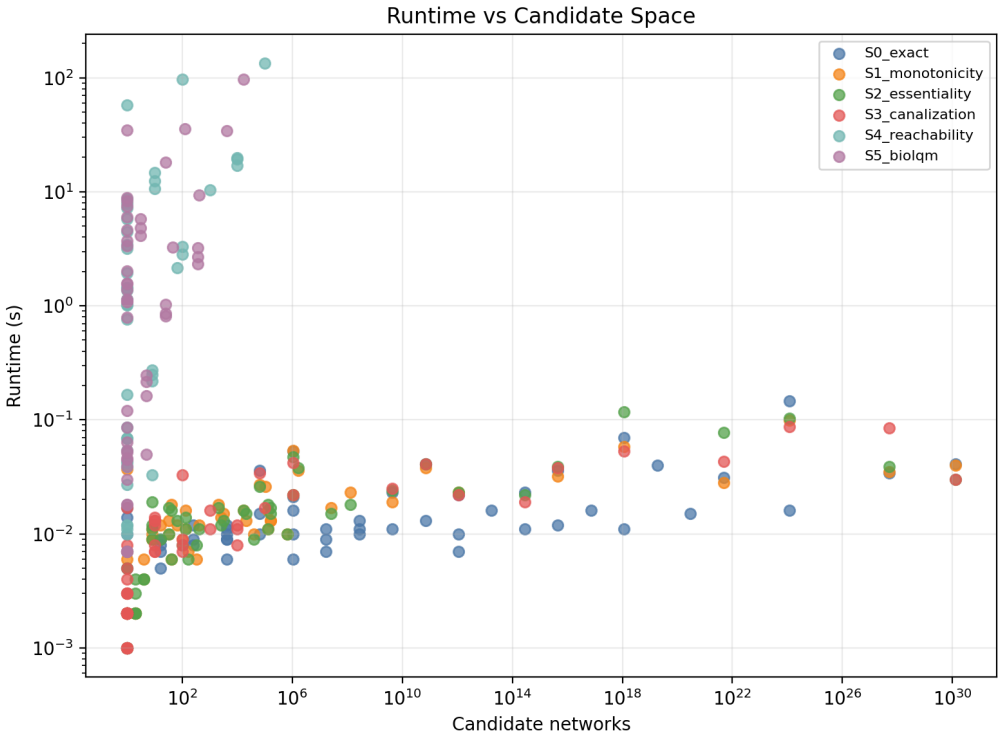


Figure 7.1: Inference runtime versus candidate space size across all 360 sketches, shown on logarithmic scales and grouped by sketch class. Sketches with only structural and functional constraints remain concentrated in the region of large candidate spaces and relatively low runtimes, whereas sketches enriched with dynamic constraints produce substantially smaller candidate spaces at a noticeably higher computational cost.

7.2 Experimental Results

Figure 7.1 provides an aggregated view of the behaviour of all 360 inference runs by plotting runtime against candidate-space size on logarithmic scales. The figure reveals a clear separation between the sketch classes that encode only structural and functional information and those that additionally include dynamical constraints.

Table 7.1 summarises the main qualitative tendencies observed for each sketch class. Rather than reporting exact numerical aggregates, which would require a dedicated statistical table, it highlights the relative role of each class in terms of ambiguity reduction and computational cost.

The first four classes, namely `S0_exact`, `S1_monotonicity`, `S2_essentiality`, and `S3_canalization`, are concentrated toward the right-hand side of the figure. This means that they typically admit very large candidate spaces, often spanning several orders of magnitude. At the same time, their runtimes remain comparatively low, usually within fractions of a second. This indicates that

structural and functional constraints alone are generally inexpensive to process, but are often insufficient to narrow the solution space substantially.

Among these four classes, the addition of monotonicity, essentiality, and canalization produces only moderate visible shifts relative to the baseline `S0_exact` sketches. Although these constraints introduce additional knowledge, the overall point cloud remains in the region of high ambiguity. The dominant pattern is therefore not a qualitative collapse of the candidate space, but rather a limited refinement of an already large family of compatible networks.

In contrast, the dynamically enriched classes `S4_reachability` and `S5_biolqm` occupy a markedly different region of the plot. Their candidate spaces are much smaller, often lying several orders of magnitude to the left of the structural-only classes. This shift shows that temporal and attractor-related information contributes substantially stronger pruning power than purely structural annotations.

However, this reduction in ambiguity is accompanied by a clear increase in computational cost. The runtimes for `S4_reachability` and `S5_biolqm` are distributed much higher on the vertical axis, with several runs extending into the multi-second range and some outliers approaching the highest values observed in the experiment. This behaviour suggests that dynamic constraints are significantly more expensive for the solver to enforce, even when they succeed in shrinking the compatible candidate space.

The reachability-based sketches `S4_reachability` appear to show the strongest concentration of high-runtime outliers. This is consistent with the fact that reachability properties require explicit reasoning about possible state transitions and paths through the dynamics of the candidate networks. The `S5_biolqm` sketches also increase runtime noticeably, but they appear somewhat more dispersed, suggesting that attractor-related constraints can be restrictive while exhibiting varying computational impact depending on the specific sketch instance.

Overall, the results show a trade-off that is visible across the full experimental dataset. Less informative sketches are easier to solve but leave a very large set of compatible models, whereas more informative sketches sharply reduce the candidate space at the cost of more demanding inference. The most important qualitative outcome of the experiment is therefore the emergence of two distinct regimes: a low-cost, high-ambiguity regime for structural and functional sketches, and a high-cost, low-ambiguity regime for dynamically enriched sketches.

The table makes the progressive change in inferential behaviour easier to read. The first four classes primarily refine the structural search space while preserving low computational cost. The last two classes, which encode dynamical information, provide the strongest reduction in ambiguity but also shift the inference task into a computationally heavier regime.

Table 7.1: Qualitative summary of the observed behaviour of the six sketch classes.

Sketch class	Information type	Candidate space	Runtime
`S0_exact`	Partial structure + exact rule revelation	Very large	Low
`S1_monotonicity`	Signed regulatory constraints	Very large to large	Low
`S2_essentiality`	Regulator relevance constraints	Very large to large	Low
`S3_canalization`	Dominant input constraints	Large	Low
`S4_reachability`	Trace-based dynamic constraints	Small to moderate	High
`S5_biolqm`	Fixed points and trap spaces	Small	High

7.3 Evaluation

The experimental results provide direct evidence for the central hypothesis stated in Section 7.1.1. Increasing the amount of knowledge encoded in a sketch generally makes the inference problem more informative, but also more computationally demanding. The extent of this effect, however, depends strongly on the type of information being added.

7.3.1 Effect of Structural and Functional Information

With respect to **RQ1**, the results indicate that increasing structural and functional information alone has a limited effect on reducing the candidate network space. The classes `S1_monotonicity`, `S2_essentiality`, and `S3_canalization` remain broadly close to the baseline `S0_exact` class in the runtime-versus-candidate-space plot. This suggests that such information is useful for refining the domain of admissible models, but is often not sufficient to isolate a small set of candidates on its own.

This observation is important from a modeling perspective. Structural and functional properties can be extracted relatively naturally from a known network and encoded compactly in a sketch, but their inferential power appears weaker than that of dynamic observations. In other words, they help shape the candidate space, but do not usually collapse it.

7.3.2 Impact of Dynamic Constraints

With respect to **RQ2**, the experimental evidence shows that dynamic constraints have the strongest effect on the inference problem. Both trace-based reachability information and attractor-related information shift the results toward much smaller candidate spaces. This means that dynamical knowledge carries significantly greater discriminatory power than static structural annotations.

At the same time, the results also show that this gain in precision is not computationally free. Dynamic constraints lead to substantially higher runtimes, especially for sketches based on reachability. The data therefore support the view that dynamical information is the most effective mechanism for reducing ambiguity, but also the main source of increased solver effort.

7.3.3 Influence of Exact Function Revelation

With respect to **RQ3**, the present aggregate figure does not isolate reveal levels explicitly, but it is consistent with the expected trend that exact revelation should simplify the inference problem by removing uncertainty from local update rules. As more local functions are revealed exactly, the symbolic search space becomes smaller, and fewer candidate networks should remain compatible with the sketch.

The effect of reveal level should therefore be interpreted as complementary to the effect of additional constraints. Exact revelation directly removes degrees of freedom from the model space, whereas monotonicity, essentiality, canalization, and dynamical properties constrain the remaining degrees of freedom indirectly. Together, these two mechanisms determine how restrictive a sketch becomes.

7.3.4 Scalability and Overall Trade-Off

With respect to **RQ4**, the experiments show that inference runtime does not depend only on the size of the candidate space. In fact, Figure 7.1 shows several cases in which smaller candidate spaces correspond to higher runtimes. This demonstrates that runtime is driven not only by how many solutions exist, but also by how difficult it is to verify the constraints that define them.

This point is especially visible for the dynamic sketch classes. Even when the compatible space is sharply reduced, the solver must reason about reachability or attractor-related behaviour over large families of networks, which introduces a substantial computational burden. Scalability must therefore be understood as the result of two competing forces: richer information reduces ambiguity, but richer constraints also make each inference instance more expensive to solve.

Taken together, the results support a nuanced conclusion. If the primary goal is to obtain fast inference, structural and functional sketches offer a lightweight option, but they leave substantial uncertainty unresolved. If the primary goal is to narrow the candidate space as much as possible, dynamic constraints are much more effective, but they incur a significantly higher computational cost. The experimental study therefore confirms that the design of a useful sketch is not only a question of how much information to include, but also of which kind of information is most beneficial for the intended inference task.

Chapter 8

Future Work

8.1 Potential Extensions **84**
8.2 Open Research Directions **85**

This chapter outlines promising directions for extending the methods, tools, and experimental analysis presented in this thesis. The proposed directions aim to enhance the expressiveness, applicability, and theoretical understanding of sketch-based Boolean network modeling.

8.1 Potential Extensions

A natural extension of the work presented in this thesis concerns the incorporation of uncertainty into the modeling and inference process. The current pipeline assumes deterministic Boolean dynamics and exact observations, whereas in many real-world applications, especially in systems biology, data are often noisy or incomplete. A well-established extension in this direction is the framework of Probabilistic Boolean Networks (PBNs), introduced by Shmulevich and Dougherty [41], which generalizes classical Boolean networks by associating probabilities with alternative update functions. In such models, the system evolves as a stochastic process rather than a deterministic one. Integrating probabilistic information into the sketch-based framework would allow the representation of uncertain observations and noisy traces, and would enable a systematic study of how uncertainty affects the inference process, the size of the candidate space, and the robustness of the resulting models.

Another important extension concerns the expressiveness of temporal specifications used within the pipeline. In the current approach, temporal properties are treated as fixed constraints that must be satisfied by all candidate models. However, in many realistic scenarios, the available knowledge about system behavior may itself be partial or uncertain. Temporal logic has been widely used for specifying and verifying dynamical properties of systems [24, 35], including biological regulatory networks. Extending the framework to support partial or flexible temporal specifications would significantly enhance its applicability. For instance, instead of enforcing

strict satisfaction of a given temporal formula, one could consider incomplete specifications, alternative behavioral constraints, or soft requirements that allow multiple admissible behaviors. Such an extension would align naturally with the notion of sketches as partial models and would enable reasoning under incomplete dynamical knowledge.

A further direction for extending the pipeline is the generalization beyond purely Boolean models to multi-valued logical networks. While Boolean networks restrict each variable to two possible states, many real systems exhibit multiple levels of activity that cannot be adequately captured by binary abstractions. Multi-valued models provide a more expressive framework for representing such systems, particularly in biological contexts where gene expression levels may vary continuously or across multiple discrete states [42]. Extending the current pipeline to support multi-valued networks would require adapting both the structural representation and the inference mechanisms, but would significantly improve the realism and applicability of the approach.

Finally, an additional extension concerns the automation of the pipeline configuration process. In the present work, the selection of structural and dynamical constraints is controlled manually, which enables systematic experimentation but may limit scalability. Future work could investigate methods for automatically selecting or optimizing the set of constraints based on the characteristics of the input data or desired performance criteria. Such approaches could draw on techniques from optimization and machine learning, enabling adaptive and data-driven configuration of the pipeline.

8.2 Open Research Directions

Beyond practical extensions, the work presented in this thesis opens several broader research directions related to the theoretical foundations of Boolean network inference and sketch-based modeling.

A fundamental question concerns the relationship between the amount of information provided in a sketch and the resulting inference complexity. In particular, it is important to understand how different types of constraints structural, functional, and dynamical affect the size of the candidate model space and the computational effort required for inference. Investigating the minimal sets of constraints required to uniquely characterize a Boolean network, as well as identifying redundancies among different forms of information, constitutes an important direction for future research.

Another key research direction is the study of identifiability in Boolean network inference. Given partial observations and constraints, it is often the case that multiple networks are consistent with the available information. Understanding when a unique solution can be obtained,

and when ambiguity is unavoidable, is essential for both theoretical analysis and practical applications. This problem is closely related to the inherent combinatorial complexity of Boolean network reconstruction, which has been shown to be computationally challenging [6].

The connection between temporal logic and Boolean network dynamics also presents opportunities for further investigation. While temporal logic provides a powerful language for specifying system behavior, it remains an open question how expressive such specifications are in fully characterizing dynamical phenomena such as attractors, reachability, and stability. Exploring the limits of temporal logic in capturing these properties, as well as identifying classes of properties that can be efficiently enforced or verified, is an important direction for future work.

In addition, extending sketch-based modeling to real-world data remains a significant challenge. Biological and other real-world datasets are often noisy, incomplete, and heterogeneous, requiring robust methods for integrating multiple sources of information. Developing techniques that can handle such data while maintaining interpretability and computational efficiency would significantly enhance the practical impact of the framework.

Finally, the interpretability of inferred models represents an important and often overlooked research direction. While the use of sketches allows for partial and flexible representations, it is important to ensure that the resulting models remain understandable and meaningful to domain experts. Studying the trade-off between model expressiveness, accuracy, and interpretability, and developing methods for generating concise and explainable models, constitutes a valuable direction for future research.

BIBLIOGRAPHY

[1] L. Mendoza, D. Thieffry, and E. R. Alvarez-Buylla, "Genetic control of flower morphogenesis in *Arabidopsis thaliana*: a logical analysis," *Bioinformatics*, vol. 15, no. 7, pp. 593–606, 1999.

[2] A. Saadatpour and R. Albert, "Boolean modeling of biological regulatory networks: A methodology tutorial," *Methods*, vol. 62, no. 1, pp. 3–12, 2013.

[3] S. Chevalier and L. Paulevé, "Boolean network synthesis from single-cell data with BoNesis," *Bioinformatics*, 2023.

[4] Axons, "Using the event-b method for critical systems," https://medium.com/axons/using-the-event-b-method-for-critical-systems-c8a7beb38214, 2020, accessed: 2026-04-17.

[5] N. Beneš, L. Brim, O. Huvar, S. Pastva, and D. Šafránek, "Boolean network sketches: A unifying framework for logical model inference," *Bioinformatics*, vol. 39, no. 4, p. btad158, 2023.

[6] T. Akutsu *et al.*, "Identification of genetic networks under the boolean model," *Pacific Symposium on Biocomputing*, pp. 17–28, 1999.

[7] C. Müssel, M. Hopfensitz, and H. A. Kestler, "Boolnet—an r package for generation, reconstruction and analysis of boolean networks," *Bioinformatics*, vol. 26, no. 10, pp. 1378–1380, 2010.

[8] L. Paulevé, "Bonesis: Synthesis of most permissive boolean networks," 2020. [Online]. Available: https://github.com/bioasp/bonesis

[9] R. Albert and H. G. Othmer, "The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in *Drosophila melanogaster*," *Journal of Theoretical Biology*, vol. 223, no. 1, pp. 1–18, 2003.

[10] R. Zhang, M. V. Shah, J. Yang, S. B. Nyland, X. Liu, J. K. Yun, R. Albert, and T. P. Loughran, "Network model of survival signaling in large granular lymphocyte leukemia," *Proceedings of the National Academy of Sciences*, vol. 105, no. 42, pp. 16 308–16 313, 2008.

[11] S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of Theoretical Biology*, vol. 22, no. 3, pp. 437–467, 1969.

[12] R. Thomas, "Boolean formalization of genetic control circuits," *Journal of Theoretical Biology*, vol. 42, no. 3, pp. 563–585, 1973.

[13] J. Saez-Rodriguez, L. Simeoni, J. A. Lindquist, R. Hemenway, U. Bommhardt, B. Arndt, U.-U. Haus, R. Weismantel, E. D. Gilles, D. L. Bhatt, and B. Bhatt, "A logical model provides insights into T cell receptor signaling," *PLOS Computational Biology*, vol. 3, no. 8, p. e163, 2007.

[14] R. Samaga, J. Saez-Rodriguez, L. G. Alexopoulos, P. K. Sorger, and S. Klamt, "The logic of EGFR/ErbB signaling: theoretical properties and analysis of high-throughput data," *PLOS Computational Biology*, vol. 5, no. 8, p. e1000438, 2009.

[15] A. Naldi, J. Carneiro, C. Chaouiya, and D. Thieffry, "Diversity and plasticity of Th cell types predicted from regulatory network modelling," *PLOS Computational Biology*, vol. 6, no. 9, p. e1000912, 2010.

[16] M. Lu, M. K. Jolly, H. Levine, J. N. Onuchic, and E. Ben-Jacob, "Microrna-based regulation of epithelial–hybrid– mesenchymal fate determination," *Proceedings of the National Academy of Sciences*, vol. 110, no. 45, pp. 18 144–18 149, 2013.

[17] N. Beneš, L. Brim, O. Huvar, S. Pastva, and D. Šafránek, "Supplementary material for boolean network sketches," 2023.

[18] L. Calzone, L. Tournier, S. Fourquet, D. Thieffry, B. Zhivotovsky, E. Barillot, and A. Zinovyev, "Mathematical modelling of cell-fate decision in response to death receptor engagement," *PLOS Computational Biology*, vol. 6, no. 3, p. e1000702, 2010.

[19] A. Saadatpour, R.-S. Wang, A. Liao, X. Liu, T. P. Loughran, I. Albert, and R. Albert, "Dynamical and structural analysis of a T cell survival network identifies novel candidate therapeutic targets for large granular lymphocyte leukemia," *PLOS Computational Biology*, vol. 7, no. 11, p. e1002267, 2011.

[20] E. Remy, S. Rebouissou, C. Chaouiya, A. Zinovyev, F. Radvanyi, and L. Calzone, "A modelling approach to explain mutually exclusive and co-occurring genetic alterations in bladder tumorigenesis," *Cancer Research*, vol. 75, no. 19, pp. 4042–4052, 2015.

[21] J. G. T. Zañudo and R. Albert, "Cell fate reprogramming by control of intracellular network dynamics," *PLOS Computational Biology*, vol. 11, no. 4, p. e1004193, 2015.

[22] A. Mochizuki, B. Fiedler, G. Kurosawa, and D. Saito, "Dynamics and control at feedback vertex sets. II: a faithful monitor to determine the diversity of molecular activities in regulatory networks," *Journal of Theoretical Biology*, vol. 335, pp. 130–146, 2013.

[23] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[24] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 2008.

[25] D. Cheng and H. Qi, “Controllability and observability of boolean control networks,” *Automatica*, vol. 45, no. 7, pp. 1659–1667, 2009.

[26] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabási, “Controllability of complex networks,” *Nature*, vol. 473, pp. 167–173, 2011.

[27] M. Aldana, “Boolean dynamics of networks with scale-free topology,” *Physica D: Nonlinear Phenomena*, vol. 185, no. 1, pp. 45–66, 2003.

[28] A. Wagner, “Does evolutionary plasticity evolve?” *Evolution*, vol. 50, no. 3, pp. 1008–1023, 1996.

[29] Y. Dimopoulos, W. Dvořák, and M. König, “Connecting abstract argumentation and boolean networks,” in *Computational Models of Argument (COMMA 2024)*. IOS Press, 2024.

[30] R. Albert and R. Robeva, “Signaling networks: Asynchronous boolean models,” lecture notes.

[31] T. Mori and T. Akutsu, “Attractor detection and enumeration algorithms for boolean networks,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E87-A, no. 5, pp. 1133–1140, 2004.

[32] A. Naldi, “BioLQM: a Java toolkit for the manipulation and conversion of logical qualitative models of biological networks,” *Frontiers in Physiology*, vol. 9, p. 1605, 2018.

[33] A. Pnueli, “The temporal logic of programs,” in *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS)*. Providence, Rhode Island, USA: IEEE, 1977, pp. 46–67.

[34] T. Hafer and W. Thomas, “Computation tree logic ctl* and path quantifiers in the monadic theory of the binary tree,” in *Proceedings of the 14th International Colloquium on Automata, Languages and Programming (ICALP)*, ser. Lecture Notes in Computer Science, vol. 267. Springer, 1987.

[35] G. Bernot *et al.*, “Application of formal methods to biological regulatory networks,” *Journal of Theoretical Biology*, vol. 229, no. 3, pp. 339–347, 2004.

[36] J. Fisher and N. Piterman, “Model checking in biology,” *STTT*, vol. 13, pp. 501–513, 2011.

[37] P. Traynard, “Model building by temporal logic constraint solving,” Ph.D. dissertation, Université Grenoble Alpes, 2016.

[38] S. Muñoz, M. Carrillo, E. Azpeitia, and D. A. Rosenblueth, “Griffin: A tool for symbolic inference of synchronous boolean molecular networks,” *Frontiers in Genetics*, vol. 9, p. 39, 2018.

[39] J. A. Goldfeder and H. Kugler, "Bre:in – a backend for reasoning about interaction networks with temporal logic," in *Computational Methods in Systems Biology*, ser. Lecture Notes in Bioinformatics, vol. 11773. Springer, 2019, pp. 289–295.

[40] A. Naldi, "Biolqm toolkit for logical models," *Frontiers in Physiology*, vol. 9, p. 1605, 2018.

[41] I. Shmulevich and E. R. Dougherty, "Probabilistic boolean networks: A rule-based uncertainty model," *Bioinformatics*, vol. 18, no. 2, pp. 261–274, 2002.

[42] G. Karlebach and R. Shamir, "Modelling and analysis of gene regulatory networks," *Nature Reviews Molecular Cell Biology*, vol. 9, no. 10, pp. 770–780, 2008.