CONTROL OF PARTIALLY SPECIFIED BOOLEAN NETWORKS

EVA ŠMIJÁKOVÁ

MUNI
FI

PhD Thesis

Faculty of Informatics
Masaryk University

2026

SUPERVISORS: doc. RNDr. David Šafránek, Ph.D.,
prof. RNDr. Luboš Brim, CSc.

“I have yet to see any problem, however complicated, which, when you looked at it in the right way, did not become still more complicated.”

— Poul Anderson

DECLARATION

Hereby I declare that this thesis is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during the elaboration of this work are properly cited and listed with complete references to the original sources. During the preparation of this thesis, I used the following AI tools to improve the clarity and style of my writing:

- Grammarly
- ChatGPT
- GitHub Copilot

Additionally, I used the following tool to assist with software development:

- GitHub Copilot

I declare that I used these tools in accordance with the principles of academic integrity. I checked the content and took full responsibility for it.

Eva Šmijáková

ABSTRACT

In various fields, a compelling challenge is to influence systems to align with desired behaviors. Formal methods offer ways to address this challenge by utilizing abstractions of real-world systems to compute the solutions. This thesis focuses on the control of biological systems, which contributes to solving cell reprogramming.

Biological systems are modeled using a variety of approaches, including Boolean networks (BNs), which, thanks to their simplicity and expressiveness, effectively capture interactions at the cellular level. While control problems for fully specified BNs have been extensively studied, the corresponding problem has received much less attention when the Boolean functions are only partially known.

Partially specified Boolean networks (PSBNs) accommodate model uncertainty by representing unknown Boolean update functions with function symbols. However, this introduces complexities in the control problem, including variations in the state transition graph between varying interpretations and optimization challenges in minimizing perturbations while ensuring target reachability.

This thesis deals with these challenges and brings the control problem to the partially specified setting of Boolean networks. The state perturbations are used to exercise control on the models since their implementation in practice can be realized via gene knockouts or overexpressions. The main difference in PSBN control is that, instead of computing only a minimal perturbation, the methods compute a set of possible perturbations together with their robustness with respect to model uncertainty.

We present and solve two flavors of the PSBN control problem – source-target control and phenotype control. Two methods were developed to deal with the source-target problem – a semi-symbolic parallel method, which uses just one-step perturbations and a symbolic method, which is able to employ one-step, temporary and permanent perturbations. The weakness of source-target control lies in the necessity to know the target attractor beforehand, which can be highly inconvenient for PSBN. This weakness is mitigated by the phenotype control, where it is sufficient to specify only characteristic traits of the target, and control works globally regardless of the source state. A method based on permanent perturbations with symbolic state space exploration was developed to solve this problem. Finally, software encompassing all these methods is described, and the applicability of the methods is demonstrated on real-world case studies.

PUBLICATIONS

This thesis encompasses and extends results from my previously published works. Here, I describe how these publications relate to the thesis as a whole. The detailed structure of the thesis is summarized in Chapter 1. The groundwork for several concepts examined here was laid in my PhD thesis proposal [Šm21].

My first publication [CMSB'20] is based on the results that I have achieved in my master thesis [Šm19]. Since all subsequent work is done on formal methods applied to systems biology, this publication gave me a solid foundation in the field. Moreover, it introduced me to the topic of colored model checking based on BDDs, which is related to the methods I have developed in this thesis. The context of these methods is described in Chapter 6.

Then I focused on the topic of control of partially specified Boolean networks, which is the backbone of this thesis. The two published works [MATHEM'21, BIOSYS'23] focus on source-target control, and their results are demonstrated in Chapter 4.

The findings from these works inspired me to investigate phenotype control of partially specified Boolean networks because it alleviates the limitations of source-target control in the partially specified setting. The results achieved with this method were published in [CMSB'23] and here are described in Chapter 5.

The methods I developed in [BIOSYS'23, CMSB'23] were then included in the AEON.py tool. This tool was also presented as the Application note [BIOINF'23]. Later, the control methods were incorporated into the frontend of the AEON tool, as presented in [CMSB'25]. AEON tool set is described in Chapter 6.

Finally, I proposed a novel framework to design control experiments for refinement of the partially specified Boolean networks. This framework and relevant case studies are presented in a journal article, that is accepted but not yet published at the time of the submission of this thesis [TOAPPEAR]. The framework along with other case studies for control of partially specified Boolean networks, is summarized in Chapter 7.

The following list chronologically enumerates the publications I coauthored. For each publication, I describe my involvement in their preparation, and an approximate percentage indicating my contribution.

[TOAPPEAR] *Control-Guided Refinement of Partially Specified Boolean Networks: Applications to RTK Signalling* (E. Šmijáková, L. Brim, S. Pastva, D. Šafránek)

An original journal paper. At the time of this thesis submission, the paper was accepted and is going to appear in the Bioinformatics journal. I have conceptualized the novel framework for control-guided refinement of partially specified Boolean networks and methods for controlling partially specified networks to oscillatory phenotypes. I conducted experiments to demonstrate the methods as well as wrote the major part of the paper. (70%)

[CMSB'25] *AEON 2025: Robust Control of Partially-Specified Boolean Networks* (V. Veselý, E. Šmijáková, S. Pastva, N. Beneš, D. Šafránek) [Ves+25]

A tool conference paper for Computational Methods in Systems Biology conference. The main author is a student whom I supervised. The student developed an extension of the AEON tool frontend based on the methods I have previously developed. I have provided consultations to the student, contributed to the implementation and testing of the tool, and written a part of the paper. (30%)

[CMSB'23] *Phenotype Control of Partially Specified Boolean Networks* (L. Brim, S. Pastva, D. Šafránek, E. Šmijáková) [Ben+23b]

A regular conference paper for Computational Methods in Systems Biology conference. I have conceptualized the novel type of control problem with better usability than the previous ones. I have proposed and implemented the methods. I designed and conducted case studies case studies that demonstrated the applicability of the developed methods. I contributed substantially to the writing. (70%)

[BIOSYS'23] *Temporary and Permanent Control of Partially Specified Boolean Networks* (L. Brim, S. Pastva, D. Šafránek, E. Šmijáková) [Bri+23]

A full-length journal article in Biosystems journal. I proposed, developed, and implemented the novel methods introduced in the paper. I designed and executed the experiments demonstrating the capabilities of the methods and did an extensive part of the writing. (70%)

[BIOINF'23] *AEON.py: Python library for attractor analysis in asynchronous Boolean networks* (N. Beneš, L. Brim, O. Huvar, S. Pastva, D. Šafránek, E. Šmijáková) [Ben+22a]

An application note published in the Bioinformatics journal. I was the author of the control methods included in the published tool paper. I have described the technical details of these methods. (30%)

[MATHEM'20] *Parallel One-Step Control of Parametrised Networks* (L. Brim, S. Pastva, D. Šafránek, E. Šmijáková) [Bri+21]

An article published in a special issue on Boolean Networks Models in Science and Engineering in Mathematics journal. I have conceptualized the novel problem of the control of partially specified Boolean networks. I have proposed the algorithm that solves this problem and collaborated on its implementation and wrote a major part of the manuscript. (70%)

[CMSB'20] *Parallel Parameter Synthesis for Multi-affine Hybrid Systems from Hybrid CTL Specifications* (E. Šmijáková, S. Pastva, D. Šafránek, L. Brim) [Šmi+20]

A regular conference paper for Computational Methods in Systems Biology conference. The article presents the results achieved in my master's thesis [Šm19]. I have proposed and implemented a method based on a previous work of the Sybila research group. I significantly participated in the manuscript writing. (60%)

ACKNOWLEDGEMENTS

As no man is an island, no thesis is completed alone. I would like to express my sincere gratitude to my supervisor, Luboš Brim, for giving me the opportunity to work on this topic and for his continuous support and guidance throughout my PhD. I am equally grateful to David Šafránek for taking up the baton and for his help and support during my studies.

I would also like to thank all members of the Sybila group for their collaboration and for keeping me inspired. A special thanks go to Samuel Pastva and Matej Troják for sharing the hardships of doctoral studies with me and for their mutual support, and to Vojtěch Veselý for development of the AEON tool user interface.

My deepest thanks go to my significant other, Michal, who has been my inspiration throughout the whole journey. Enrolling in the PhD program at the same time and reaching the finish line sooner than I did, he has been a constant source of motivation, support and understanding.

Finally, I would like to thank my family for their unwavering support and encouragement throughout my studies, and my friends and colleagues for their understanding, support, and for resisting the urge to ask about the progress of my thesis.

CONTENTS

DECLARATION v
ABSTRACT vii
PUBLICATIONS ix
ACKNOWLEDGEMENTS xiii
1 INTRODUCTION 1
1.1 Contribution . . . 3
1.2 Thesis structure . . . 6
2 PRELIMINARIES 9
2.1 Regulatory networks . . . 9
2.2 Boolean networks . . . 10
2.2.1 Notation . . . 11
2.2.2 Definition . . . 11
2.2.3 Updating schema . . . 12
2.2.4 State space structures . . . 14
2.2.5 Summary . . . 15
2.3 Control of Boolean networks . . . 15
2.3.1 Control objective . . . 16
2.3.2 Perturbations . . . 17
2.3.3 Temporal properties of perturbations . . . 18
2.3.4 Summary . . . 23
2.4 Partially specified Boolean networks . . . 23
2.4.1 Interpretations and instances . . . 24
2.4.2 Normalization of PSBN . . . 25
2.4.3 Colored state transition graph . . . 26
2.4.4 Well-known problems on PSBNs . . . 27
2.5 Summary . . . 29
3 STATE OF THE ART 31
3.1 Control of discrete systems . . . 31
3.2 Control of partially specified systems . . . 33
3.3 Control of synchronous Boolean networks . . . 34
3.4 Control of asynchronous Boolean networks . . . 36
3.4.1 Refined strong basins search . . . 36
3.4.2 Stable motifs identification . . . 37
3.4.3 Trap spaces . . . 38
3.4.4 Model checking . . . 39
3.4.5 Other techniques . . . 39
3.5 Control of most-permissive Boolean networks . . . 40
3.6 Leveraging control for model refinement . . . 41
3.7 Related tools . . . 42
3.8 Summary . . . 44
4 SOURCE-TARGET CONTROL OF PSBN 45
4.1 Problem definition . . . 45

4.1.1 Perturbations, perturbed state transition graph and runs . . . 46
4.1.2 Control types and goal . . . 48
4.1.3 Robustness . . . 50
4.1.4 Control problem . . . 50
4.1.5 Summary . . . 51
4.2 Semi-symbolic method for one-step control . . . 51
4.2.1 Semi-symbolic labelled strong basin search . . . 52
4.2.2 Control computation workflow . . . 54
4.2.3 Results . . . 56
4.2.4 Summary . . . 58
4.3 Symbolic method for one-step, temporary and permanent control . . . 58
4.3.1 Symbolic computation model . . . 59
4.3.2 Symbolic operations . . . 60
4.3.3 Supporting algorithms . . . 61
4.3.4 Control algorithms . . . 62
4.3.5 Evaluation . . . 65
4.3.6 Summary . . . 71
5 PHENOTYPE CONTROL OF PARTIALLY SPECIFIED BOOLEAN NETWORKS 73
5.1 Problem definition . . . 73
5.2 Methods . . . 75
5.2.1 Symbolic computation model . . . 75
5.2.2 Control algorithm . . . 77
5.2.3 Phenotype control with oscillations . . . 79
5.3 Evaluation . . . 82
5.3.1 Performance . . . 82
5.3.2 Method validation . . . 84
5.4 Summary . . . 85
6 SOFTWARE 87
6.1 Software overview . . . 87
6.2 Core packages . . . 89
6.3 AEON web application . . . 92
6.4 AEON.py . . . 94
6.5 Summary . . . 95
7 APPLICATIONS 97
7.1 Motivation: cell reprogramming . . . 97
7.2 Myeloid case study . . . 98
7.3 Control workflow for therapy and model refinement . . . 103
7.3.1 Control-driven therapy design . . . 104
7.3.2 Control-guided model refinement . . . 104
7.4 MAPK case study . . . 109
7.4.1 Isolated FGFR3 pathway model . . . 110
7.4.2 Grieco MAPK . . . 117
7.4.3 Extended FGFR3 pathway MAPK . . . 120

7.5 Summary 125
Conclusions 127

A Notation 131
B Digital attachments 133

Bibliography 135

1 INTRODUCTION

A compelling challenge for many scientific fields is finding ways to influence systems, so they exhibit a desired behavior. Biology is no exception. Finding a cure to a disease is essentially finding a way to control the system of the human body to get rid of the disease. Of course, today, we are not yet able to make a single complete model of the human body, but we still can make models of some of its parts, such as gene-protein interactions in a cell [Car05].

Cell models are therefore valuable tools for studying and designing disease-related interventions. For example, we can try to find a way to reprogram a cancer cell into a healthy one or to reprogram a stem cell into a phenotype (a specific cell type) that is needed for regenerative medicine. The problem of controlling a cell is called *cell reprogramming*, and it is one of the most significant challenges of regenerative medicine [CD12]. In practice, reprogramming strategies often rely on targeted perturbations of gene expression, including forced over-expression, or knockout, implemented through methods such as CRISPR-based technologies [DC14] and RNA interference [Fir+98].

To study and design such interventions, we rely on field of *systems biology* that aims to apply formal methods developed by computer science to biological models. Systems biology uses a variety of models ranging from discrete (e.g. Boolean networks [Sch+20]), through continuous (e.g. ODE [GIP13]), to hybrid models [Gro+93]. A very efficient model to capture numerous interactions at the cellular level is *Boolean network* (BN) because this model is both simple and expressive. Moreover, BNs have applications in many other areas, including circuit theory and behavioral science [Rol+15; KS14].

BNs are composed of two parts: a finite set of Boolean *variables* representing genes or other biochemical substances and *update functions* which specify the way variables dynamically change their value based on influences from other variables. BNs generate a finite discrete state-space which makes them convenient for computational analyses. When starting from any initial state, the BN eventually establishes in some single state or a set of states. These components of BN state space are called *attractors*. They can be viewed as an abstraction for phenotypes of the modeled biological systems [Che+16].

A computational model is considered controllable if we can assure that from some initial state, it reaches a desired final state in a finite amount of steps. When controlling BNs, we are interested in BN reaching the desired attractor. This property is well-studied in the context of BNs without unknown parts [ZA15; Man+19; SP20a; CFTS20].

However, the problem has not received a lot of attention in the partially specified setting yet, that allows to capture the uncertainty in the model typical for biological systems [Zou13; Ben+19].

The definition of control is rather broad and allows many variations of the problem for BNs. To change the normal behavior of BN, understandably, we need to interfere with the model in some way. We call these interventions *perturbations*. However, we can consider many types of perturbations. They vary in three aspects. Firstly, we can either fix variable values to some fixed value or change the rules of their evolution. Secondly, we can apply the perturbation just as a single transition step (e.g. a brief exposure to a signalling molecule), or we can hold the perturbations for more transition steps (e.g. a prolonged drug treatment), but even permanent perturbations (e.g. a CRISPR-mediated gene knockout) are considered. Lastly, the perturbation can be applied only once, or we consider applying perturbations and releasing perturbations at any time.

Apart from the different kinds of perturbations, control problems also vary in terms of the objective. Four common control goals are: (i) driving the system from a specified source state to a specified target attractor (*source-target control*) [SPP19b], (ii) achieving a single desired target attractor irrespective of the current state (*target control*) [KPC13], (iii) achieving control between every pair of attractors (*full control*) [Fie+13], and (iv) reaching the set of states satisfying given conditions (*phenotype control*) [CFTS20]. For instance, source-target control may consist in driving a proliferating cell to an apoptotic attractor; target control would require driving the system to the same apoptotic attractor from *any* initial cellular state; full control may correspond to the ability to switch the system among all stable cell fates represented by attractors, such as proliferation, quiescence, and apoptosis; and phenotype control may correspond to steering the network to any state satisfying a prescribed pattern of marker activity, for example states in which stemness genes are inactive, and differentiation markers are active.

In practice, the exact Boolean functions of the model may not be precisely known due to various uncertainties such as insufficient experimental knowledge [Gri+13], inconsistent observations [GGC16], genetic mutations [Mar+16], or any other ambiguities. In contrast, there is typically good evidence of the fact that a variable regulates another one. This issue can be addressed by allowing BNs to contain function symbols. *Partially specified Boolean networks* (PSBNs) [Zou13; Ben+19] allows one to specify only regulators of a variable and thus capture multiple variants of the possible actual behavior of variables without conducting many more expensive experimental observations. Every interpretation of the unknown parts can be understood as an instance of a standard BN. The number of these instances can be doubly exponential in the worst case [WSA12].

The partially specified setting brings many new challenges to the control problem. First, the nature of BN state transition graph may change dramatically between the interpretations [Ben+19; Ben+20]. Even the original goal of the standard control – to stabilize in a target attractor – might not be feasible in some instances as the given attractor is not present in them. Secondly, the BN control problem is optimizing – we are trying to minimize the number of perturbations done to a network, while we must guarantee that the target attractor is reached in all BN non-deterministic runs. It was shown that for many biological models, controlling values of only a few variables is sufficient to drive the network into any attractor of the BN [ZA15]. On the contrary, this is usually not the case for the PSBNs. Due to the uncertainties, we would often need to fix the value of many variables to guarantee PSBN to reach the given attractor in all network instances. Such an approach is not applicable in practice as each perturbation can be very costly in the real environment. Therefore, the optimization criteria should be adjusted.

In our current research line, we decided to treat the interpretations solely as unknown knowledge and consider only the state perturbations that fix the value of variables. To cope with minimization criteria, the output of the control problem for PSBN is a complete *mapping* from the domain of perturbations to the domain of relevant interpretations. This way, it is possible to manually explore the available perturbations and consider both the cost of perturbation and the *robustness* of perturbation – the proportion of interpretations for which the perturbation ensures PSBN stabilization in the given attractor with all considered interpretations.

1.1 CONTRIBUTION

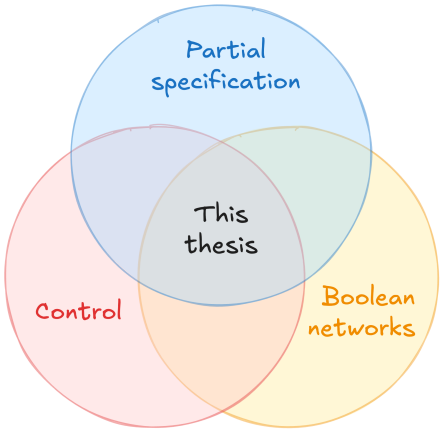


Figure 1: The confluence of concepts studied in this thesis.

The contribution of this thesis lies in its integration of three distinct concepts: control, asynchronous Boolean networks and systems

with partial specification. Such a confluence has not yet been explored previously. The asynchronous Boolean networks are a well-studied model of system biology. Also, control is a commonly employed formal method in systems biology and the topic of controlling asynchronous BNs was already thoroughly studied as we describe in Section 2.3. On the contrary, the promising model of partially specified Boolean networks is just beginning its rise. Other formal methods such as attractor search [Ben+21; RP22], bifurcation [Ben+22c], or parameter synthesis [Ben+23a] are also being investigated only relatively recently.

The partial specification of BNs thus makes this work truly novel, although the specific approaches are often inspired by previously investigated intersections of the topics as we will see in the thesis. The objectives of this thesis are four-fold:

1. Formally define control problem for partially specified Boolean networks
2. Develop efficient methods that solve the control problem for partially specified Boolean networks
3. Bundle the developed methods to a software toolkit
4. Demonstrate applicability for biological systems

Now let us have a look at these goals in detail and describe in what parts of this thesis they are fulfilled.

GOAL 1 – PROBLEM DEFINITION. Problem definition is an essential contribution of this work since the problem of PSBN control has not been formalized prior to our work. Several crucial aspects need to be considered for the problem definition. Firstly, attractors of the PSBN vary according to the interpretation. Therefore, searching for non-permanent perturbations that drive the network into a target attractor would be impossible for interpretations of unknown parts, where the given attractor is not present in the first place. Secondly, the practical realization of perturbations requires non-trivial effort; therefore, in a normal BN setting the number of perturbations is sought to be minimal. However, there is rarely a solution with a few perturbations in the partially specified setting, which would work for a big portion of interpretations. Thus, optimization criteria should be readjusted accordingly.

The achievement of this goal is presented in Section 4.1 where we define *source-target* control for PSBN which result is a relation of perturbations with interpretations for which they work as opposed to standard control goal where only minimal perturbation sets are returned. However, the landscape of attractors may change between interpretations [Ben+22c]. That might cause a shift of target attractors that is not impeding from the cell reprogramming objective, but it

is technically not allowed by a source-target control. This caveat is resolved by a more general notion of control – *phenotype* control which is defined in Section 5.1 where only target traits are necessary to be specified.

GOAL 2 – EFFICIENT METHODS DEVELOPMENT. As described before, the control problem for fully specified asynchronous BNs has been well-studied already. Therefore, after control for PSBN is formally defined, in theory, we could use the existing methods for asynchronous BN control to solve the problem for every interpretation distinctly. However, the number of these interpretations can be doubly exponential, meaning even a very efficient method for BN control would need to be run too many times to be handled in a realistic time. That is why seeking a novel, efficient approach for the partially specified setting is crucial.

Three methods were developed to solve the posed variants of the problem.

1. Parallel one-step source-target control method for PSBN with semi-symbolic state space exploration and symbolic result which is described in Section 4.2 [Bri+21]
2. One-step, temporary, and permanent source-target control for PSBN with symbolic state space exploration and symbolic result which are described in Section 4.3 [Bri+23]
3. Permanent phenotype control method for PSBN with symbolic state space exploration and result requiring explicit enumeration which is described in Chapter 5 [Ben+23b]

The aforementioned sections present algorithms, and simple (i.e., quantitative) demonstrations of them.

GOAL 3 – SOFTWARE TOOLKIT FOR METHODS. In order to make the developed methods publicly available, they were incorporated to the BioDivine toolkit [Bar+09]. A library in Rust based on AEON tool [Ben+20] was developed to make methods fast and performant. Later, the library was wrapped into Python language as part of a tool AEON.py [Ben+22a]. This helps to make the methods more convenient to work with for the wider scientific audience because the language is easier to learn and offers popular notebook environments such as Jupyter [Bar21]. The Python version also has become part of the CoLoMoTo Interactive Notebook tools [Nal+15; Lev+18]. Finally, the control functionality was also embedded into AEON web application [Ves+25]. The software technical specification and solved challenges are in detail described in Chapter 6.

GOAL 4 – APPLICATIONS TO BIOLOGICAL SYSTEMS. Finally, since Boolean network control is closely connected to cell reprogramming, we demonstrate that the developed methods address the original

biological motivation of this thesis. In addition, we show that control can also serve as a tool for model refinement. Both applications, together with case studies illustrating their utility, are presented in Chapter 7.

1.2 THESIS STRUCTURE

As mentioned before, this thesis covers multiple interlaced themes. Thus, it is important for the reader to recognize what aspects are considered while reading this thesis, especially distinguishing where only fully specified Boolean networks are discussed and where their partial specification is assumed. To help with that, a switch from the context be indicated by accompanying figure in the margin of the text, similar to one in Figure 1. Also, in the following text we provide a brief guidance on the structure of the thesis and what is covered in each chapter.

Chapter 2 describes all necessary preliminaries to understand the basic problematic of the fully specified Boolean networks. Regulatory networks, which are an essential basis for BNs and also PSBNs are described at first (Section 2.1). Then, all motivation and formal definitions related to fully specified BNs are introduced (Section 2.2). After that, the control problem for BNs, is described, including technical differences between all variants of the problem (Section 2.3). Finally, the notion of partially specified Boolean networks is introduced (Section 2.4), including the motivation for this model and the technical details of it.

Chapter 3 is about the state of the art in the field of control of Boolean networks. It covers the methods for control from the most general classes of the systems to the systems most similar to the PSBNs. First, it describes roots of the control theory in terms of discrete systems (Section 3.1) and incompletely specified systems (Section 3.2). Then, it describes the methods for control of fully specified BNs from the simplest synchronous BNs (Section 3.3) to the more complex asynchronous (Section 3.4) and most-permissive BNs (Section 3.5). Finally, it describes how the control problem for BNs can aid the problem of model refinement (Section 3.6) and a comparison of the software tools for control of BNs is given (Section 3.7).

Chapter 4 provides the problem definition of source-target control problem for PSBN (Section 4.1) and then it also shows two methods that solve this problem. The first one, semi-symbolic employs only the most trivial one-step perturbation (Section 4.2). The latter developed fully symbolic method is capable of one-step, temporary and permanent perturbations (Section 4.3). Finally, these two methods are discussed and compared (Section 4.3.5).

Chapter 5 describes the problem of phenotype control for PSBN (Section 5.1) and a method to solve it (Section 5.2). The method is

based on the same symbolic state space exploration as the one for source-target control. Finally, the method is evaluated on several middle-sized PSBNs (Section 5.3) in terms of performance and the quality of the results.

Chapter 6 describes the software implementation of the presented methods. First, the software architecture is described (Section 6.1), including the main libraries and their functionalities (Section 6.2). Then, the web application for control of PSBN is presented (Section 6.3), followed by the description of the Python package (Section 6.4).

Chapter 7 is about the applications of the developed methods for control of PSBN. Firstly, it described the primary motivation for this work – the problem of cell reprogramming in regenerative medicine (Section 7.1). Then, it shows an example of PSBN control application for cell reprogramming on a specific example of a myeloid cell (Section 7.2). After that, a more general workflow of how the control of PSBN can be used to design a therapy or to refine the model is described (Section 7.3). Finally, this workflow is shown on the second case study that is focused on the MAPK signalling pathway and demonstrates the control-guided model refinement (Section 7.4).

Finally, Conclusions summarizes the main contributions of this thesis and discusses possible future directions for research in this area.

2

PRELIMINARIES

This chapter provides the necessary background to understand the concepts and models used throughout the thesis. First, we introduce the concept of regulatory networks as abstract representations of interactions between biological components (Section 2.1). Building on this foundation, we formally define Boolean networks in Section 2.2, which serve as the core modeling framework throughout this thesis. With this model in place, we turn to the control of Boolean networks and explore various control strategies in Section 2.3. Finally, we extend the classical framework to partially specified Boolean networks in Section 2.4, a novel model variant that incorporates uncertainty and incomplete knowledge.

2.1 REGULATORY NETWORKS

The model designers of biological systems typically begin their work by identifying connections and influences among proteins, genes, and other biochemical substances. Such an approach is often chosen because gaining evidence for these kinds of interconnections is usually the most accessible. We may observe that in the presence or a lack of some substance, another is behaving differently. On the other hand, translating such interaction into an explicit mathematical form is more complex.

The model that captures just simple relationships between biochemical substances (or other entities) is called regulatory networks (RNs) [DL05]. Consider an example in Figure 2. It compactly represents a relationship between three enzymes and proteins. For example, we can observe transitive feedback between network components. Influencing FGFR3 could impact the whole rest of the network, while by contrast, perturbing ERK would not affect FGFR3 at all.

Figure 2: **An example of a simple regulatory network.** The RN contains three nodes and four regulations (two activations, one inhibition and one non-essential self-regulation). The network is a small extract of a MAPK signaling pathway from [Gri+13].

Definition 1. *Let $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ denote a directed graph representing the regulatory network, where $\mathbb{V}$ is the set of vertices representing the*

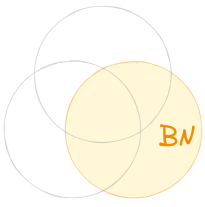
molecular components and $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ *is the set of directed edges representing the regulatory interactions between the components.*

We call the influence of one substance (regulator) to another (target) a *regulation*. These regulatory effects typically fall into two categories: upregulation (activation), which denotes a positive influence (usually represented in green), or downregulation (inhibition), indicating a negative effect (depicted in red). These types of regulations are *monotonous*; since their impact is consistent. Non-monotonous regulations appear in biological models rarely. When they occur or if the monotonicity is not required, they are depicted in a black color.

Apart from the monotonicity, another aspect we consider of regulation is *observability*. Observable (essential) regulations were proven by experimentation that the regulator surely influences the target. I.e., in some settings where the regulator is present the regulated substance behaves differently compared to the environment with the lack of the regulator. On the contrary, in case we do not have this certainty, the regulation is *non-essential*. The non-essential regulations should not be over-used. Hypothetically, we could mark all possible RN edges with this type of regulation resulting in a complete, dense graph giving us very little information. Nonetheless, if these regulations occur, they are labelled by a question mark. All other regulations are considered to be observable. We give more formal details about different regulation types in Section 2.2.2.

Notice, that the influences captured by RNs are highly abstract and that there is very little modeling knowledge present in this model. That is why more specific models are using RNs as a basis. Among these models are ODEs, stochastic gene networks, or Boolean networks, which are described next.

2.2 BOOLEAN NETWORKS



The traditional differential equation-based (ODE) models often struggle to capture the intricate interactions and feedback loops present in biological systems due to their complexity. This shortcoming was addressed by Boolean networks (BNs) that were introduced by Kauffman in the 1960s [Kau69]. Kauffman represented system components as binary variables connected by logical rules. Such an abstraction facilitated the exploration of system dynamics while avoiding the computational complexity of traditional models.

Since then, it has become a widely used model to study the dynamics of complex biological systems [Bar+20; Alb04]. The Boolean representation also allows researchers to focus on the regulatory relationships and dynamics without getting hindered by detailed biochemical mechanisms. This simplification facilitates efficient analysis of regulatory networks, enabling the discovery of crucial pathways, pre-

diction of system behavior, and identification of pivotal components driving cellular processes.

2.2.1 *Notation*

Here, the notation used in the context of Boolean networks is briefly summarized. For a complete overview of the used notation, refer to Appendix A.

We consider 0 and 1 as interchangeable with *false* and *true*, respectively. We then define $\mathbb{B} = \{0, 1\}$ to be the set of Boolean values and $\mathbb{B}_* = \{0, 1, *\}$ to be an extension of $\mathbb{B}$ which also admits a *free* value $*$ (i.e. neither *true* nor *false*). We write $\mathbb{B}^n$ to denote the set of all n-element vectors over $\mathbb{B}$. For each $x \in \mathbb{B}^n$, x_i then denotes the i-th element of x. Finally, for $x \in \mathbb{B}^n$, index $i \in [1, n]$, and a Boolean value $b \in \mathbb{B}$, expression $x[i \mapsto b]$ denotes a substitution of the i-th element in x for the value b. Formally, the result is $x' = x[i \mapsto b]$ s.t. $x'_j = b$ for $j = i$, and $x'_j = x_j$ otherwise.

2.2.2 *Definition*

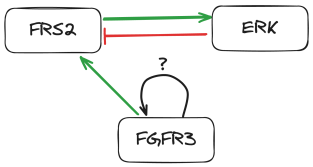
We can formally define Boolean networks, as follows:

Definition 2. *Let n be the number of system variables. A* Boolean network *is a collection $\mathcal{N} = \{f_1, \ldots, f_n\}$ with each $f_i : \mathbb{B}^n \to \mathbb{B}$ being the Boolean* update function *(sometimes called* local function*) of the network's i-th variable.*

A running example of a Boolean network based on the RN from Figure 2 can be seen in Figure 3. Note that, while it is mathematically more convenient to denote variables by their indices, oftentimes we will use their actual names (e.g. FGFR instead of f_1) to present context of the biological system comprehensively.

In the context of a specific Boolean network, the set $\mathbb{B}^n$ is the network's *state space*, and the vectors $x \in \mathbb{B}^n$ are its *states*. Note that although the input of each f_i is a full state $x \in \mathbb{B}^n$, the output of f_i does not typically depend on all network variables, but rather on a smaller subset of variables which we say *regulate* the i-th variable. These regulations are typically given by a corresponding regulation network. For a BN to respect the regulations defined by its regulation network, the function should comply with the regulation type in following ways:

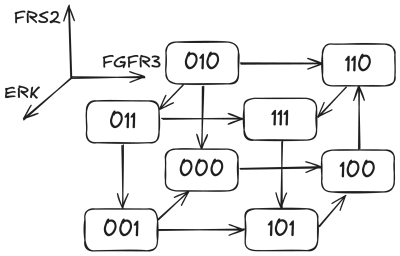
- If a variable j is *essential* (observable) for a variable i, denoted as $\mathcal{E}(j, i)$ then:
  $\exists x \in \mathbb{B}^n . f_i(x[j \mapsto 1]) \neq f_i(x[j \mapsto 0])$. We sometimes denote this simply as $j \to i$.



(a) Regulatory network

$$\begin{aligned} \text{FGFR} &= 1 \\ \text{FRS2} &= \text{FGFR} \wedge \neg \text{ERK} \\ \text{ERK} &= \text{FRS2} \end{aligned}$$

(b) Boolean network



(c) State transition graph

Figure 3: **An example of a Boolean network and related concepts.** In Figure 3a, the same RN as in Figure 2 is displayed for a reader's convenience. This RN is used as a basis for BN defined in Figure 3b. Finally, state-transition graph of the BN is shown in Figure 3c where the labeling of states has the same order as variables order in Figure 3b.

- If variable j *upregulates* (activates) variable i, denoted as $\mathcal{U}(j, i)$: $\forall x \in \mathbb{B}^n . f_i(x[j \mapsto 0]) \Rightarrow f_i(x[j \mapsto 1])$
- If variable j *downregulates* (inhibits) variable i, or $\mathcal{D}(j, i)$: $\forall x \in \mathbb{B}^n . f_i(x[j \mapsto 1]) \Rightarrow f_i(x[j \mapsto 0])$

If a variable has no regulators (i.e. its update function is a constant), we also call it an *input* of $\mathcal{N}$. Symmetrically, a variable that does not regulate any other variable is called an *output*. For example, when we consider Figure 3a, if the self-regulation of FGFR3 was not present, FGFR3 would be considered an input.

Additionally, we call the members of $\mathbb{B}_*^n$ the *subspaces* of $\mathbb{BN}$. Intuitively, each subspace $S \in \mathbb{B}_*^n$ describes a hypercube in the state space $\mathbb{B}^n$. This hyper-cube consists of states $x \in \mathbb{B}^n$ such that $x_i = S_i$ for all i where $S_i \in \mathbb{B}$. We can thus treat each subspace S as a set of states. Furthermore, to denote a specific subspace, we will often simply use a string of values from $\mathbb{B}_*$ instead of the full vector notation (e.g. $S = 11*0$ instead of $S = (1, 1, *, 0)$).

2.2.3 *Updating schema*

In our work, we focus on *asynchronous* semantics for Boolean networks. To give a reader a better overview of the differences between the various semantics, we briefly discuss other common semantics and their trade-offs. A more formal and detailed guide to BN semantics can be found in [PAU23].

In Boolean network modeling, the semantics used to define the update rules of node states significantly influence the system's dynamics and interpretability. *Synchronous* semantics, as introduced in [Kau69], operate under the assumption that all nodes in the network update their states simultaneously in discrete time steps. While this approach simplifies analysis and yields deterministic behavior, it imposes an unrealistic constraint on systems such as gene regulatory networks, where simultaneous updating is biologically implausible.

In contrast, *asynchronous* semantics, introduced in [Tho73], allow for only one node to update at each time step, with a nondeterministic choice of which node updates. This reflects the inherent variability in timing and activity across components in real-world systems, such as the stochastic firing of genes. Asynchronous semantics maintain a more biologically and physically faithful representation of dynamics without the need to introduce explicit timing or probability.

Probabilistic semantics introduce randomness directly into the update process. A variable has assigned more than a single update function, while each update function has assigned probability of triggering. This allows the modeling of noise and uncertainty but requires a probabilistic framework and often obscures causal interpretability.

Lastly, the *most permissive* semantics that was recently introduced in [Pau+20], define all logically consistent transitions as simultaneously valid, capturing the broadest possible behavior. This is achieved by considering a transient state between 0 and 1 for each variable, allowing any combination of variables to update in a way that is consistent with their logical functions. While useful for exhaustive exploration, this framework might be hard to use for quantification of state-space components since these properties are lost to the abstraction via transient states.

Asynchronous semantics offer a compelling middle ground. They provide a natural and realistic modeling framework for systems where the order and timing of updates are nondeterministic. Nonetheless, they preserve the causal structure of networks and allow for quantifiable tractable analysis while avoiding the oversimplification of synchronous updates.

To formally reason about the evolution of the network's state, we consider asynchronous state-transition graph of BNs:

Definition 3. *Given a Boolean network* $\mathcal{N} = \{f_1, \ldots, f_n\}$*, the* state-transition graph $\mathsf{STG}(\mathcal{N}) = (\mathbb{V}, \mathbb{E})$ *is a directed graph with* $\mathbb{V} = \mathbb{B}^n$ *and* $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ *given as follows:*

$$(u, v) \in \mathbb{E} \Leftrightarrow (u \neq v \wedge \exists i \in [1, n].\, v = u[i \mapsto f_i(u)])$$

The *state-transition graph* $\mathsf{STG}(\mathcal{N})$ captures the dynamics of the Boolean network by representing its states and transitions. An example is graphically depicted in Figure 3c.

The behavior of a network is then studied through its *maximal strongly fair runs*. Here, a maximal run $\pi : \mathbb{N}_0 \to \mathbb{B}^n$ represents an infinite sequence of states where for every $t \in \mathbb{N}_0$, we have either $\pi(t) = \pi(t+1)$, or $\pi(t) \to \pi(t+1)$. That is, the state either remains constant, or updates according to $\mathrm{STG}(\mathcal{N})$.

Furthermore, we are only concerned with *strongly fair* runs [AL95; Man+19]. These are runs which do not delay any available transition indefinitely. Formally, let π be a run and $x \in \mathbb{B}^n$ a state that appears infinitely often in π, such that $x \to y$ for some $y \in \mathbb{B}^n$. Then π is fair only if y also appears infinitely often in π as a successor to x. To denote the set of such states that appear infinitely often in a run π, we write $\mathrm{inf}(\pi) = \{x \in \mathbb{B}^n \mid \forall k \in \mathbb{N}_0, \exists t > k.\ \pi(t) = x\}$. Finally, we may write $x \in \pi$ as a shorthand for $\exists t.\ \pi(t) = x$ and a set of all possible runs in $\mathcal{N}$ is denoted as $\Pi(\mathcal{N})$.

2.2.4 *State space structures*

The focus on strongly fair runs naturally leads to the notions of trap set, trap space, and attractor, which dictate the long-term behavior of the corresponding BN:

Definition 4. *Let $\mathcal{N} = \{f_1, \ldots, f_n\}$ be a Boolean network and $X \subseteq \mathbb{B}^n$ a set of network states. We say that X is a* trap set *when for all $x \in X$ and $y \in \mathbb{B}^n$ we have that $x \to y$ implies $y \in X$ (i.e., X cannot be escaped). A subspace $X \in \mathbb{B}^n_*$ which is also a trap set can be also called a* trap space. *A trap set X is an* attractor *when X is strongly connected within* $\mathrm{STG}(\mathcal{N})$.

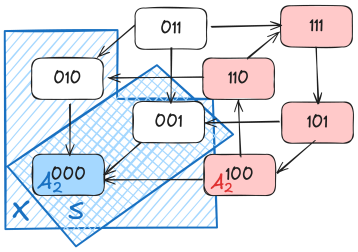


Figure 4: **Difference between a trap set and a trap space.** A state space of a BN with $n = 3$ is shown. The BN contains two attractors: a steady state A_1 (a blue state 000) and an oscillatory attractor A_2 (red states $1{*}{*}$). The trap set S (double-shaded are) is also a trap space as it can be described by a subspace $00{*}$. On the other hand, the trap set X (shaded area) is not a trap space as it cannot be described by a subspace. Notice, that also a set of all states ${*}{*}{*}$ is a trap space (set), even though it contains multiple attractors.

For a visualization of the concepts, see Figure 4. Equivalently, we can define an attractor A as the inclusion-minimal trap set of $\mathcal{N}$.

Notice, that an attractor of a run is exactly the set of states that appear infinitely often in the run $\mathrm{inf}(\pi) = A$. We write $\mathbb{A}(\mathcal{N})$ to de-

note the set of all attractors of $\mathcal{N}$. An attractor containing only a single state is called a *steady state* or *fixed point*. On the other hand, we will refer to attractors containing multiple states as *oscillatory attractors**.

*Some works also distinguish oscillatory (where period of states repeating in the sequence is strictly regular) and chaotic attractors [Pas22]. In this work we call both these types as oscillatory.

Finally, observe that due to our fairness assumption, all runs admitted by STG($\mathcal{N}$) eventually reach some attractor. As such, for each strongly fair π, the set of states that appear infinitely often in π is always an attractor. For example, if we consider the BN from Figure 3, it has a single attractor; the oscillation in subspace $0**$. All runs of this BN eventually reach this attractor.

Given an attractor, we also define its *basin of attraction* as the set of all states that can reach the attractor. The larger the basin of attraction, the attractor is more likely to be biologically relevant [KB05]. Due to the non-deterministic nature of asynchronous BNs, individual states may reach multiple attractors depending on the selected successor states. To that end, we recognize two types of basins of attraction [Kla+18; Sch+20]:

Definition 5. *The* weak basin of attraction *comprises all states which can lead to the given attractor:*

$$WB(\mathcal{N}, A) = \{x \in \mathbb{B}^n \mid \exists \pi \in \Pi(\mathcal{N}) : x \in \pi \wedge (\forall s \in A : s \in \pi)\}$$

In contrast, the strong basin of attraction *includes all states from which the given attractor is eventually surely reached.*

$$SB(\mathcal{N}, A) = \{x \in \mathbb{B}^n \mid \forall \pi \in \Pi(\mathcal{N}) : x \in \pi \Rightarrow (\forall s \in A : s \in \pi)\}$$

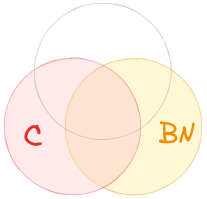
If Boolean network $\mathcal{N}$ is clear from the context, it can be omitted from the notation. States which belong to a strong basin of attraction lead to only one possible attractor. In contrast, states in the weak basin of attractor may lead to a certain attractor but also another one. This concept is also shown in Figure 5. As we will see later in Chapter 4, these notions are particularly useful in the context of BN control.

2.2.5 *Summary*

In this section, we summarized the key concepts related to Boolean networks and their dynamics. We introduced the notions of states and updating schemas. We have introduced state space structures, such as trap sets and attractors, which are crucial for understanding the long-term behavior of Boolean networks Building on these concepts, we now turn our attention to the control of Boolean networks.

2.3 CONTROL OF BOOLEAN NETWORKS

In this section we zoom into the *control* of Boolean networks. We first discuss different control objectives that have been studied in the lit-



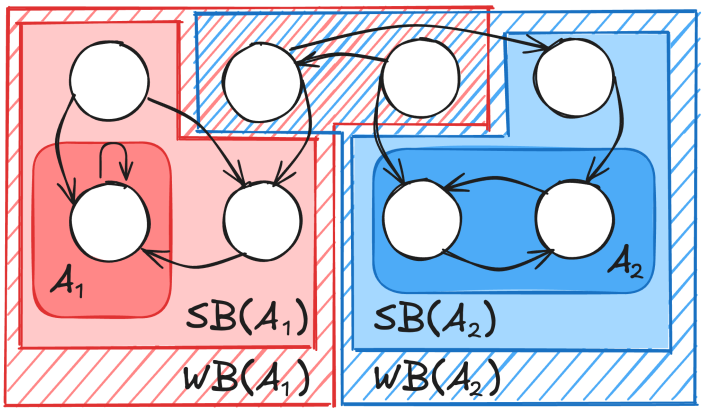


Figure 5: **Difference between a strong and a weak basin.** The figure shows a simple STG with two attractors: a steady state A_1 (dark red) and an oscillatory attractor A_2 (dark blue). Strong basins of A_1 and A_2 are shown in light red and light blue, respectively and are disjoint. Finally, the weak basin of A_1 and A_2 are shown in red and blue stripes, respectively. The weak basins overlap as some states can reach multiple attractors.

erature. Then, we introduce the concept of perturbations, which are the means of controlling a Boolean network. Finally, we summarize the aspects in which the problem of control of Boolean networks differ.

2.3.1 *Control objective*

The first aspect in which control problems differ is "what kind of stabilization we are trying to achieve". Since the only component stable in a BN are its attractors, the control objectives typically focus on navigating a system to reach a certain attractor or a set of attractors.

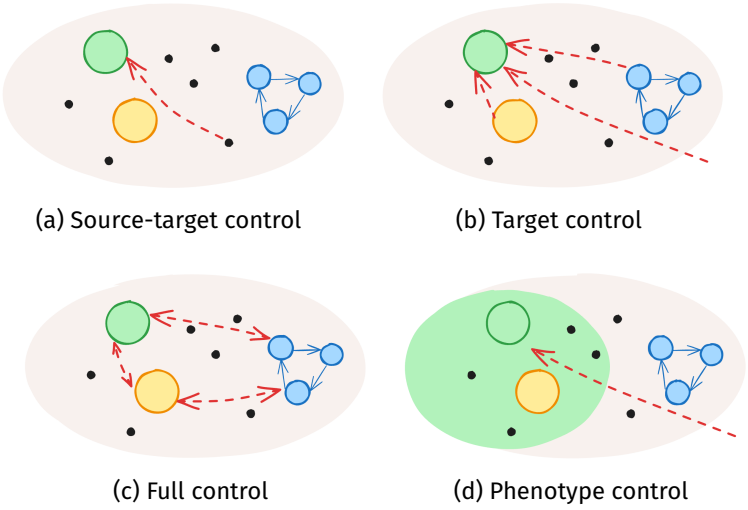
There are two base types of control objectives: *existential* and *inevitable* control [PPS18b; Man19]. Existential control aims to find a strategy that makes for a system *possible* to reach the target. On the other hand, universal control seeks a strategy that *ensures* convergence to the target attractor from any initial state. In this work, we consider only the inevitable control objectives.

We consider five different control objectives based on the previous lines of research [Bau+19; CFTS20]. The differences between control objectives are also illustrated in Figure 6:

1. *Source-target control*: Given a *source state* $s \in \mathbb{B}^n$, and a *target attractor* $T \in \mathbb{A}$, ensure, that the BN converges from the state s to the attractor T (see Figure 6a).

2. *Target control*: Given a *target attractor* $T \in \mathbb{A}$, ensure that for *any* initial state $s \in \mathbb{B}^n$ the BN converges from s to T (see Figure 6b).

3. *Full control*: Find a set of control strategies such that for all pairs of distinct attractors $S, T \in \mathbb{A}$, applying a strategy from this set guarantees that the BN converges from S to T (see Figure 6c).
4. *All-pairs control*: Given a subset of source attractors $\mathcal{S} \subseteq \mathbb{A}$ and target attractors $\mathcal{T} \subseteq \mathbb{A}$, find a set of perturbations $\mathcal{Q}$ such that for every pair $S \in \mathcal{S}$ and $T \in \mathcal{T}$, there exists a subset perturbations $\mathcal{Q}' \subseteq \mathcal{Q}$ that guarantees that the BN converges from S to T (similar to full control in Figure 6c).
5. *Phenotype control*: Given a subset of states $X \subseteq \mathbb{B}^n$, ensure that from any initial state, the BN converges to an attractor $A \subseteq \mathbb{A}$ such that $A \subseteq X$ (Figure 6d).



(a) Source-target control (b) Target control

(c) Full control (d) Phenotype control

Figure 6: **Difference between control objectives.** The black points are non-attractor states of BN, the colorful points are attractors, and the red arrows shows where we are trying to stabilize by applying the control. The green area in the Figure 6d represents a target phenotype.

The control objectives as defined above are rather high-level since they do not specify using what means are allowed to achieve the goals. In particular, they do not clarify what types of perturbations can be applied to the system in order to guide it towards the desired attractors. Therefore, we discuss the perturbations next.

2.3.2 *Perturbations*

Perturbations are the means of controlling a Boolean network. In order to ensure that a network always reaches some attractor which it does not reach in some of its normal runs, logically, we need to interfere with the system in some way. This is exactly what perturbations are – a forced intervention to the dynamics of a system. There

are several types of perturbations, but in this work we focus on *variable perturbations*, which override the value of some variables in the network by constant values.

Other types of perturbations are *function perturbations* which can override the update functions completely and in a more complex way. However, due to their complexity, they were so far considered only within a context of synchronous Boolean networks [Liu+17; XD07]. Somewhat less abstract is a concept of *edge perturbations* which can disable or enable certain regulations in the network [FLTS22]. Both these two notions can be considered within the framework of partially specified Boolean networks, however due to their complexity both in regard to formalization and practical application, they were not included in this work, and we leave their exploration for future research.

The perturbations are formally defined as follows:

Definition 6. *Given a Boolean network* $\mathcal{N} = \{f_1, \ldots, f_n\}$*, a* variable perturbation *is a vector* $Q \in \mathbb{B}_*^n$*. For every* $Q_i = *$*, we say that the* i*-th variable is* unperturbed*, whereas for* $Q_i = 0$ *or* $Q_i = 1$*, we say that the variable is* perturbed *(to either* 0 *or* 1*). The* size *of* Q *is the number of perturbed variables: Size*$(Q) = |\{i \mid Q_i \neq *\}|$*. A perturbation can again be seen as a* subspace *of the* $\mathcal{N}$ *state space* $\mathbb{B}^n$*. The set of all considered perturbations is denoted as* $\mathbb{Q}$ *and typically equals to* $\mathbb{B}_*^n$*.*

We further recognize two special types of perturbations based on their size; an *empty perturbation* is a perturbation of size 0 (i.e., $Q = **\ldots*$) which does not perturb any variable, for short, denoted as $\emptyset$. In contrast, a *trivial perturbation* is a perturbation of size n (i.e., $Q \in \mathbb{B}^n$) which perturbs all variables in the network to a value of the target state. Finally, if a perturbation is very small (i.e one or two variables being perturbed), we can denote it as an assignment of perturbed variables instead of a full vector. For example, for $n = 4$, instead of writing $Q = 1*0*$, we can write $\{v1 = 1, v3 = 0\}$.

Usually, the perturbations are expensive. Therefore, we want to keep the amount of performed perturbations as small as possible. Thus, control is an optimization problem and is typically measured by a perturbation size (number of perturbed variables), but other criteria might be considered as well.

It can be seen, that the definition of perturbation is rather abstract and does not specify how the perturbation is applied. To make perturbations applicable, we need to consider also *temporal* properties of perturbations.

2.3.3 *Temporal properties of perturbations*

The perturbations differ based on their temporal characteristics. The first classification is based on the length of the applied perturbation.

We consider *one-step*, *temporary*, and *permanent* perturbations. Figure 7 shows an intuition on the differences between these types of perturbations.

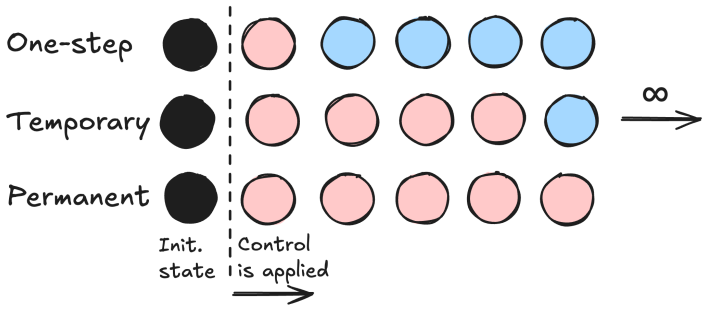


Figure 7: **Difference between one-step, temporary and permanent control.** The black markers represent initial states. Red markers represent states which were reached during the time while BN was perturbed and finally, blue markers represent states to which BN transitioned with its natural behavior. The run of BN would continue in the same nature as its last transition.

One-step perturbation (OSP) overrides values of controlled variables once, and then the network is left to evolve in its natural dynamics. Intuitively, for target control, the OSP can be viewed as a restriction of the initial states of all BN runs. Technically, the BN run may be at any state before the time step in which the perturbation is applied. Nonetheless, the very next step would bring the network to a state which is part of the OSP.

The perturbations applied for multiple time-steps are temporary (TP) and permanent perturbations (PP). As the name suggest, when we use temporary perturbation, there exists a finite amount of time steps after which the perturbation is released, and BN is left to evolve in its natural dynamics. Temporary perturbation applied for only a single time step is equivalent to OSP.

Finally, the most intrusive perturbations are permanent. This kind of perturbation might be harder to achieve in practice, depending on the specific perturbation context and techniques. Moreover, the permanent perturbation might disrupt the network's original long-term behavior and change the network's attractor landscape; even cause a target attractor to be no longer present in the BN.

To formally define application of perturbations, we differentiate between two types of perturbation applications. The first type is an application to a specific source state, as is the case of source-target control.

Definition 7. *Given a BN* $\mathcal{N} = \{f_1, \ldots, f_n\}$, application of a perturbation $Q \in \mathbb{Q}$ to a source state $s \in \mathbb{B}^n$ *is a* forced *state transition (i.e.,*

might not follow the BN standard dynamics), $\delta(s, Q) = s \to s'$ *such that:*

$$\delta(s_i, Q) = s'_i = \begin{cases} s_i & \textit{iff } Q_i = * \\ Q_i & \textit{iff } Q_i \neq * \end{cases}$$

where s' *is the state reached after applying the perturbation.*

**What we call here an application of perturbation (δ) is in some literature called state perturbations and a perturbed BN or runs of perturbed BN are referred to as function perturbations [Man19]. This is because constants are seen to be an override of variable update function. However, we avoid function perturbation term to avoid confusion with line of work on synchronous Boolean networks where more complex than constant functions are used to perturb networks [Liu+17].*

After application of a perturbation to a state*, we obtain a state which is *compliant* with the perturbation. This means that the state does not contradict any of the perturbed variables. For example, if we apply perturbation $1*0$ to state 001 (contradicting the perturbation at the first variable), we obtain state 101 which is compliant with the perturbation.

The aforementioned definition is effectively capturing the concept of one-step perturbation for the source-target control. Nonetheless, if we wanted to consider OSP for target or phenotype control, we need to generalize the definition to a whole set of states. To that end, we can adapt the definition above to a set of (all) states and as a result, we obtain a subspace equivalent to the given perturbation as the viable set of initial states.

Another important concept we need to define to understand the application of temporary and permanent perturbations is a *perturbed state transition graph* for a BN:

Definition 8. *Given a BN* $\mathcal{N} = \{f_1, \ldots, f_n\}$ *where* $\mathsf{STG}(\mathcal{N}) = (\mathbb{V}, \mathbb{E})$*,* application of a perturbation $Q \in \mathbb{Q}$ to a Boolean network $\mathcal{N}$ *results in a* perturbed Boolean network $\mathcal{N}_Q$ *with* perturbed state transition graph $\mathsf{STG}_Q(\mathcal{N}) = (Q, \mathbb{E}_Q)$ *where vertices are a subspace of the unperturbed state space* $\mathbb{V} = \mathbb{B}^n$ *given by the perturbation* Q *and a set of perturbed edges* $\mathbb{E}_Q \subseteq \mathbb{E}$ *is given as follows:*

$$\mathbb{E}_Q = \{(u, v) \mid (u, v) \in \mathbb{E} \wedge u \in Q \wedge v \in Q\}$$

We can see, that by definition, the perturbed state transition graph is a subgraph of the original state transition graph, containing only the states and transitions that are not violating the perturbation. On the other hand, the perturbed STG might contain different attractors than the original STG. For example trivially, if a BN contains a steady-state attractor, and we apply a perturbation that contradicts some variable of the attractor, the attractor may no longer be present in the perturbed STG. Another example can be seen in Figure 8 where due to perturbation the former attractor $1**$ is no longer present and is replaced by the steady-state attractor 110.

Now, we can define a perturbed run of a BN:

Definition 9. *Let us assume a BN* $\mathcal{N} = \{f_1, \ldots, f_n\}$*, source states* $\mathcal{S} \subseteq \mathbb{B}^n$*, a perturbation* $Q \in \mathbb{Q}$*, a time period* $z \in \mathbb{N}_\infty$*, after which the perturbation is withheld and a natural run* $\pi \in \mathsf{STG}(\mathcal{N})$*, such that*

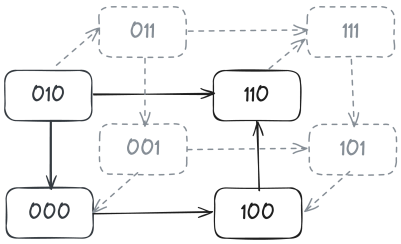


Figure 8: **A perturbed state transition graph of a Boolean network.** The depicted STG is a perturbed version of the STG from Figure 3c under perturbation $**0$. The black solid vertices and edges mark the states and transitions that are preserved in the perturbed STG while the gray dashed vertices and edges represent the states and transitions that have been removed due to not being consistent with the perturbation.

there exists a time-step t' *where* $\pi(t') \in \mathcal{S}$. A perturbed run π_Q *is a run which satisfies the following conditions:*

$$\pi_Q(t) = \begin{cases} \pi(t) & \textit{iff } t \leqslant t' \\ \delta(t, Q) & \textit{iff } t = t' + 1 \\ s' \textit{ s. t. } \pi(t-1) = s'' \wedge (s'', s') \in \mathbb{E}_Q & \textit{iff } t' + z \geqslant t > t' + 1 \\ s' \textit{ s. t. } \pi(t-1) = s'' \wedge (s'', s') \in \mathbb{E} & \textit{iff } t > t' + z \end{cases}$$

A set $\Pi_Q(\mathcal{N}, \mathcal{S}, z)$ *is a collection of all perturbed runs* π_Q *for a given perturbation* Q*, applied to any state from* $\mathcal{S}$ *with perturbation being withheld after* z *time-steps.*

Intuitively, a perturbed run consists of four "stages". First, the run is evolved “normally" in a given BN. When the BN reaches a state in $\mathcal{S}$ (does not need to be the first occurrence of an $s \in \mathcal{S}$ in the run), we apply a perturbation to that state. As a result, we obtain the nearest state which is compliant with the perturbation. From this state, the BN evolves according to the perturbed STG. If z is not infinite, after z time-steps, the perturbation is withheld, and the BN continues to evolve according to its natural dynamics.

Example 1. *An example of a valid perturbed run for a BN from our running example (Figure 3) under perturbation* $**0$ *applied to a state* 101 *and withheld after* $z = 5$ *time-steps is the following:*

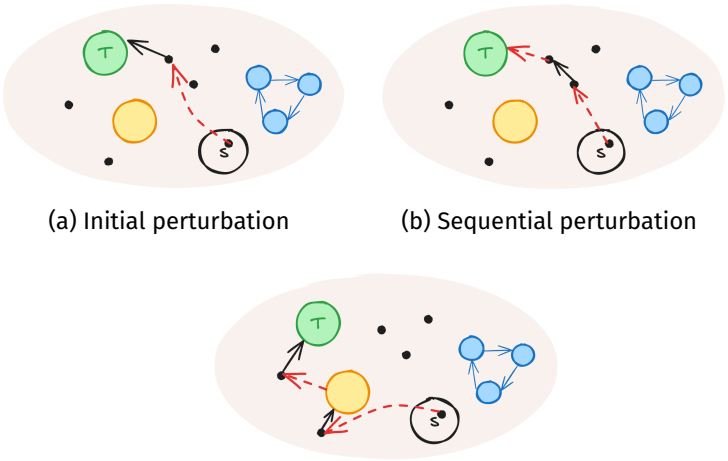
- $t = 0 \ldots 2$*: The BN evolves naturally, e.g., traverses the states* $011 \to 001 \to 101$.
- $t = 3$*: The perturbation* $**0$ *is applied, resulting in state* 100.
- $t = 4 \ldots 8$*: The BN evolves according to the perturbed STG, reaching state* 110*. There are no further viable transitions in the perturbed STG, therefore the BN remains in state* 110.

- $t = 9$: *The perturbation is withheld, and the BN continues to evolve naturally, stabilizing in the attractor* $1**$.

Table 1: **Types of perturbations.** The table summarizes the differences between one-step, temporary, and permanent perturbations when applied to a source-target control, or target types of control (target/full/all-pair/phenotype).

	one-step	temporary	permanent
source-target	$\delta(s, Q)$ or $\Pi_Q(\mathcal{N}, \{s\}, 1)$	$\Pi_Q(\mathcal{N}, \{s\}, z)$ s.t. $z \in \mathbb{N}$	$\Pi_Q(\mathcal{N}, \{s\}, \infty)$
target	$\delta(\mathbb{B}^n, Q)$ or $\Pi_Q(\mathcal{N}, \mathbb{B}^n, 1)$	$\Pi_Q(\mathcal{N}, \mathbb{B}^n, z)$ s.t. $z \in \mathbb{N}$	$\Pi_Q(\mathcal{N}, \mathbb{B}^n, \infty)$ or $\mathcal{N}_Q$

The Table 1 summarizes the different types of perturbations and how they can be implemented in the context of different control objectives using the concepts of perturbation application, perturbed STG, and perturbed runs. We can notice, that a one-step control can be seen as a special case of temporary control with time steps sustaining perturbation $z = 1$. Moreover, permanent control can be implemented as a perturbed Boolean network.



(a) Initial perturbation

(b) Sequential perturbation

(c) Sequential attractor-based perturbation

Figure 9: **Difference between initial, sequential, and sequential attractor-based perturbations.** The figure is only illustrative and does not show precise transitions or perturbations done to the BN. The black points are non-attractor states of BN. The colorful points are attractors, the black arrows are transitions conducted by natural BN dynamics, and the red dashed arrows are transitions done under the effect of perturbations. The initial state is marked as s while the target attractor is the blue one marked with T.

Some perturbation techniques also consider *when* the perturbations are applied. In *initial* or also referred to as *immediate* control, the perturbations are applied to the given initial state.

Alternatively, during *sequential* control, (possibly different) perturbations can be applied multiple times at different states. This way, we can potentially control the system using fewer perturbations. However, this approach brings new issues, such as dealing with the nondeterminism of the BN (different perturbations may be required for different branches of the non-deterministic network's behavior).

Moreover, we would need to be able to precisely observe the current state of the BN, which can be very problematic to obtain in practice. These issues can be addressed by using *attractor-based sequential* approach [Man+19]. Differences between these approaches are depicted in Figure 9.

In this work, we focus only on the initial control approach, since attractor landscape can be very complex in context of partially specified Boolean networks. On the other hand, the sequential control approach may offer smaller perturbation sizes in certain scenarios that might arise in a future research.

2.3.4 *Summary*

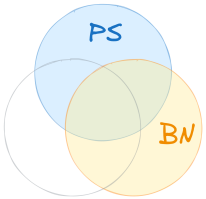
In summary, the control problems for Boolean networks differ in the following aspects:

- *What do we want to control; goal*: What is the initial state of the BN? Where we want to end? Do we want to control only one scenario or multiple scenarios?
- *What type of perturbations we apply*: We can either perturb states functions of variables.
- *For how long is perturbation applied*: We can perturb BN only during a single computational step, or we can hold it temporarily, or even forever.
- *When we apply the perturbations*: Only once at an initial state in contrast with applying control to an arbitrary state and any amount of times.

Since in this work, we focus on the control of Boolean networks, in the context of partially specified Boolean networks, we introduce them in the next section.

2.4 PARTIALLY SPECIFIED BOOLEAN NETWORKS

A shortcoming of classical Boolean networks as defined above is that to study network dynamics, all update functions must be fully speci-



fied. Moreover, the exact form of the update function might have little impact on observable behavior of the network (i.e., the attractor landscape).

Therefore, it was suggested to observe the dynamics of the Boolean network in context of of Boolean networks ensembles first [Kau04; Sch+20]. In this approach, we consider a set of Boolean networks that share some common properties (e.g., the same network structure or the same values of certain variables in their attractors). The goal is to study the dynamics of the whole ensemble instead of a single network. This way, we can obtain more robust results that are not dependent on a specific choice of update functions.

The ensemble approach described as above, however, has its limitations, as it is not formalized very well and is usually limited to simulations. To address this problem, we consider the notion of *partially specified Boolean networks* (PSBNs) [Ben+19; Ben+22b] (sometimes also termed parametrized Boolean networks or colored Boolean networks).

In a PSBN, we use *function symbols* as stand-ins for unknown (fixed but arbitrary) parts of the network's unspecified dynamics. Therefore, the dynamics of a variable are either denoted by a standard (fully specified) function or by its function symbol and input arguments.

Definition 10. *Let n be the number of variables, and $\mathbb{G} = \{g_1, \ldots, g_m\}$ a set of* function symbols. *A* partially specified Boolean network $\mathcal{M} = \{E_1, \ldots, E_n\}$ *consists of function symbol expressions given by the following grammar:*

$$E ::= 0 \mid 1 \mid x \mid \neg E \mid E \wedge E \mid E \vee E \mid g^{(a)}(E, \ldots, E)$$

Here, x ranges over the network variables and g over the function symbols of $\mathbb{G}$ (superscript $a \in \mathbb{N}_0$ denotes the arity of g). Other Boolean operators (e.g. $\Rightarrow$ or $\Leftrightarrow$) can be implemented as syntactic abbreviations using $\vee$, $\wedge$, and $\neg$.

In other words, a PSBN is defined using standard Boolean constants (0 and 1), variable state propositions (x_i) and Boolean connectives ($\neg$, $\wedge$, $\vee$), but it can also use function symbols from $\mathbb{G}$ as a way of expressing unknown behavior. This makes it possible to idiomatically describe systems whose dynamics are not fully known (see Example 2).

2.4.1 *Interpretations and instances*

To give meaning to a particular $\mathcal{M}$, we rely on the term *interpretation*. Formally, an interpretation I is a set of $\{i_1, \ldots, i_m\}$ Boolean functions such that for each $g^{(a)} \in \mathbb{G}$, there is a corresponding Boolean function $i^{(a)} \in I$ of the matching arity a.

Intuitively, an interpretation associates a specific Boolean function to every function symbol. Consequently, we can substitute all symbols $g \in \mathbb{G}$ for functions $i \in I$ in a $\mathbb{G} = \{g_1, \ldots, g_n\}$ of a $\mathcal{M}$, and as a result we obtain a collection of n fully specified functions. The result is a *Boolean network instance* which we denote as $\mathcal{M}(I)$ and which has the same semantics as a standard BN. The set of all possible interpretations of $\mathbb{G}$ is denoted as $\mathbb{I}(\mathbb{G})$ or $\mathbb{I}$ for short (i.e., when the context is clear). Some subset of interpretations is then referred to as $\mathcal{I} \subseteq \mathbb{I}$.

2.4.2 *Normalization of partially specified Boolean networks*

Function symbols are practical as a human-friendly specification language of unknown network behavior. However, it is not entirely clear how algorithms should represent such functions symbolically, as is our goal in this paper. To alleviate this issue, we note the following: Assuming that all m function symbols in $\mathbb{G}$ have arity zero, possible interpretations $\mathbb{I}$ of $\mathbb{G}$ correspond to the vectors $\mathbb{B}^m$. This is because a nullary function symbol is necessarily a constant and can be thus interpreted as a Boolean value. We call such symbols *zero-arity symbols* (we can also simply write g instead of $g^{(0)}()$ when using such symbols). In some literature, these are also referred to as network inputs [KH11].

Given any $\mathcal{M}$, we can produce a *normalized* $\widehat{\mathcal{M}}$ that only uses a fresh set of logical nullary function symbols $\widehat{\mathbb{G}}$ instead of general function symbols $\mathbb{G}$, but its interpretations are the same fully specified BNs as for the original $\mathcal{M}$. To implement this normalization, we use the following expansion rule (here, $E_1, \ldots, E_a$ are arbitrary Boolean expressions):

$$g^{(a)}(E_1, \ldots, E_a) \equiv (E_1 \Rightarrow g_P^{(a-1)}(E_2, \ldots, E_a)) \\ \wedge (\neg E_1 \Rightarrow g_N^{(a-1)}(E_2, \ldots, E_a))$$

This rule transforms any expression using a function symbol g of arity a into an expression using two fresh function symbols of arity $a-1$ (g_P and g_N). Interpretations of the original and the transformed expression result in the same concrete Boolean functions. By recursively applying this rule, we can transform any $g^{(a)}$ into 2^a nullary function symbols, each corresponding to one row of the truth table of the original g. Without the loss of generality, we can thus assume that the interpretations of a particular $\mathcal{M}$ are identified by the members of $\mathbb{B}^m$ for some $m = |\widehat{\mathbb{G}}|$. The set of all interpretations is then exactly equivalent to $\mathbb{I} = \mathbb{B}^m$. An example of a normalized $\widehat{\mathcal{M}}$ is given in Example 2.

We must not forget, that the basis of BNs (and PSBNs) are the regulatory networks as described in Section 2.1. Therefore, the interpretations should respect the constraints on the functions imposed by

the regulations. For example, if we have a definition of an expression for x_1 given as $E_1 \equiv g(x_1, x_2)$, and we know, that x_2 has a down-regulating effect on x_1, the functional symbol g should *not* be replaced by a function $x_1 \vee x_2$.

To address this, we will consider the set $\mathbb{I}$ to contain only *consistent* interpretations*. An interpretation I is consistent with a given regulatory network if for every variable x_i and its corresponding expression E_i, the function f_i obtained by substituting all function symbols in E_i with their corresponding functions from I respects the regulatory structure of the network. The restrictions on regulations can be found in Section 2.2.2.

*More on the topic of consistent interpretations can be found in a line of work on Boolean networks refinement [Ben+23a].

2.4.3 *Colored state transition graph*

With this knowledge in mind, we can define a *colored* state-transition graph that collectively encodes the behavior of all possible interpretations of a particular $\mathcal{M}$ using only Boolean values*. In this representation, the interpretations of the normalized $\widehat{\mathcal{M}}$ (the members of $\mathbb{B}^m$) each define a potentially different transition relation over the shared state space $\mathbb{B}^n$:

*Note, that update schema of PSBNs is different from the probabilistic BNs mentioned in Section 2.2.3. In particular, the probabilistic semantics use multiple update functions for the same variable within a single run while PSBN considers only a single instance (interpretation) for each run.

Definition 11. *Let $\widehat{\mathcal{M}} = \{E_1, \ldots, E_n\}$ be a normalized partially specified Boolean network such that $\widehat{\mathcal{M}}$ admits m nullary function symbols $\mathbb{G} = \{g_1, \ldots, g_m\}$. The* labelled state-transition graph $\mathsf{STG}(\widehat{\mathcal{M}}) = (\mathbb{V}, \mathbb{I}, \mathbb{E})$ *is a directed graph where $\mathbb{V} = \mathbb{B}^n$, $\mathbb{I} = \mathbb{B}^m$ and $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{I} \times \mathbb{V}$ is given as $(u, I, v) \in \mathbb{E} \Leftrightarrow (u, v) \in \mathbb{E}(\mathsf{STG}(\widehat{\mathcal{M}}(I)))$. We write a state transition under I as $u \to_I v$. The set of all interpretations enabling a transition from u to v is denoted as $\mathcal{I}(u, v) = \{I \mid (u, I, v) \in \mathbb{E}\}$.*

In other words, the labelled state-transition graph is a unifying structure that incorporates the STG of all instances of $\mathcal{M}$: A transition from a state u to state v is enabled for an interpretation I if the same transition appears in the STG of $\mathcal{M}(I)$. Similarly, we consider PSBN runs in the context of a specific interpretation I (or instance) denoted as Π_I. An example of a PSBN instance and its STG is depicted in the following Example 2.

Example 2. *Consider a PSBN $\mathcal{M}$ with $\mathbb{G} = \{g^{(2)}\}$ and given by three expressions:*

$$E_1 \equiv g(x_1, x_2)$$
$$E_2 \equiv x_1 \wedge x_3$$
$$E_3 \equiv x_1 \vee \neg x_3$$

After normalization, we have $\widehat{\mathbb{G}} = \{g_{00}, g_{01}, g_{10}, g_{11}\}$ *and expressions:*

$$\begin{aligned}
\widehat{E}_1 &\equiv ((\neg x_1 \wedge \neg x_2) \Rightarrow g_{00}) \\
&\wedge ((\neg x_1 \wedge x_2) \Rightarrow g_{01}) \\
&\wedge ((x_1 \wedge \neg x_2) \Rightarrow g_{10}) \\
&\wedge ((x_1 \wedge x_2) \Rightarrow g_{11}) \\
\widehat{E}_2 &\equiv x_1 \wedge x_3 \\
\widehat{E}_3 &\equiv x_1 \vee \neg x_3
\end{aligned}$$

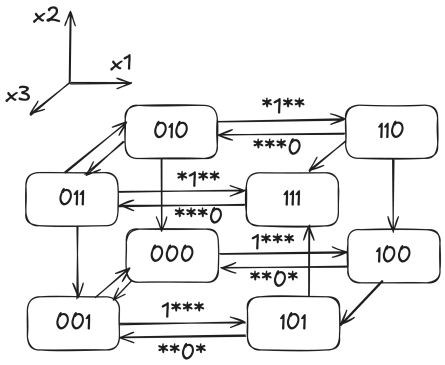


Figure 10: **State-transition graph of a PSBN.** Figure shows the state-transition graph of a PSBN from Example 2. The labels of edges represent all interpretations for which the edge is admissible. For brevity, edges admissible in all instances are drawn without a label while edges admissible for no instance are not depicted in the figure at all.

Given the notions of labelled STG and a PSBN instance, we can transfer the state space structures previously introduced for BNs to the PSBN setting. We will not focus on the control and perturbation notions, since they are part of the contribution of the thesis and are described in Chapter 4 and Chapter 5.

The notions which are independent on edges (transitions) of a STG such as character, trait and phenotype do not depend on the actual dynamics of the network (only on its variables) are identical for BNs and PSBNs.

Definitions of trap set, attractor, attractor's weak and strong basins for fully specified BNs are lifted to PSBNs per individual instances (interpretations). For example, we write that a set $A \subseteq \mathbb{V}$ is an attractor for an instance $\mathcal{M}(I)$ (i.e. there are no attractors of $\mathcal{M}$, only attractors of its instance).

2.4.4 *Well-known problems on partially specified Boolean networks*

Now that we have defined PSBNs, we can also consider well-known problems solved for this type of models, in order to understand the

challenges they present and give us a better understanding of the tools we can use to solve the control problems. Or vice versa, we can also consider how the control problems can be used to solve the well-known problems on PSBNs.

2.4.4.1 *Attractor search*

Attractor search in PSBNs involves identifying sets of states (attractors) that the system tends to evolve towards, regardless of the initial state. This is crucial for understanding the long-term behavior of the networks. As opposed to fully specified BNs, PSBNs can have a significantly larger number of attractors due to the uncertainty in their dynamics. Moreover, the attractors can differ across instances, leading to a more complex attractor landscape.

For PSBNs, usually the approaches from fully specified BNs are applied, but they are either highly parallelized or adjusted to work with the labelled STG. For example, one can use symbolic methods to represent and manipulate the state space efficiently, allowing for the exploration of multiple interpretations simultaneously [Ben+22b; PAU23].

In context of control, attractor search is often a preliminary step to identify potential target states or behaviors that we want to achieve through control interventions. It also serves as a good last step to verify if the control objectives have been met.

2.4.4.2 *Bifurcation*

As described before, bifurcation analysis studies how changes in parameters (in the case of PSBNs, interpretations) can lead to qualitative changes in the system's dynamics. In PSBNs, this involves examining how different interpretations affect the attractor landscape and identifying critical interpretations that lead to significant changes in behavior [Ben+22c; Pas22].

This problem is particularly relevant in biological contexts, where certain gene regulatory networks may exhibit different behaviors under varying conditions or mutations. Bifurcation analysis can help identify these critical points and understand the robustness of the network's behavior.

In context of control of PSBNs, bifurcation analysis again can be used as an aid for attractor search and deciding suitable target attractors.

2.4.4.3 *Model checking*

Model checking in PSBNs involves verifying whether certain properties or specifications hold across all possible interpretations of the network. This is particularly challenging due to the combinatorial explosion of possible interpretations, but it is essential for en-

suring that the network behaves as expected under various conditions [Ben+23a].

Model checking can be used to verify safety properties (e.g., certain undesirable states are never reached), liveness properties (e.g., certain desirable states are eventually reached), and other temporal properties of interest. In the context of control, model checking can be used to verify desired dynamic properties of the perturbed models.

2.4.4.4 *Model refinement*

All the methods above are also often used for the grand objective of *model refinement*, which aims to reduce the uncertainty in a PSBN by eliminating interpretations that are inconsistent with observed data or desired properties. This process helps in narrowing down the set of possible network behaviors, making the model more precise and reliable [Ben+23a]. Such a model is more likely to yield accurate predictions and insights into the underlying biological processes.

In the context of control, model refinement can help in identifying more effective control strategies by omitting spurious network behaviors. But also vice versa, as we describe in Chapter 7, we suggest control to be used as another tool for model refinement.

2.5 SUMMARY

In this chapter, we have introduced the necessary background for understanding the control of Boolean networks. We started by defining regulatory networks, and Boolean networks and their dynamics, including the concept of attractors and basins of attraction.

We then discussed various control objectives, types of perturbations that can be applied to control the network, including one-step, temporary, and permanent perturbations, as well as their temporal properties.

Finally, we introduced partially specified Boolean networks, which allow for the representation of uncertainty in network dynamics, and discussed their normalization and the concept of a colored state transition graph.

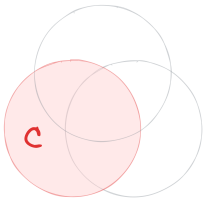
This foundational knowledge provides the basis for the following chapters. We first review existing control strategies and algorithms for fully specified Boolean networks, and then introduce the methods developed in this thesis for the partially specified setting.

3

STATE OF THE ART

In this chapter, we will discuss the state of the art related to control of Boolean networks. We will start with a brief overview of the history of control theory and control of partially specified systems. We will then focus on control of synchronous, asynchronous and most-permissive BN which served as an inspiration of development of the PSBN control methods presented in the following chapters (Chapter 4 and Chapter 5). Next, we discuss overlap and applicability of control to the problematic of model inference. Finally, we exhaustively compare the existing control methods and tools in the field of BN control and discuss the limitations of the existing methods and tools.

3.1 CONTROL OF DISCRETE SYSTEMS



The field of control theory has an extensive historical background, tracing back to the 19th century when James Clerk Maxwell formalized the principles of feedback through his seminal work on governors [Max68]. These mechanical regulators, such as the centrifugal governor used to maintain windmill speed [FCZ03], embodied some of the earliest notions of automatic control, though their application was predominantly focused on systems governed by continuous dynamics.

As engineering advanced into the early 20th century, a paradigm shift towards discrete systems began with the rise of relay logic and switching circuits. Pioneered by Claude Shannon in his groundbreaking master's thesis [Sha38; Sha40], the application of Boolean algebra to electrical circuits laid the foundation for the control of discrete systems. This marked the transition from purely continuous control to symbolic, state-based reasoning in control design.

The 1940s brought a unifying conceptual breakthrough with the emergence of cybernetics. Led by Norbert Wiener, who was himself inspired by Maxwell's early insights, cybernetics established a theoretical framework for feedback, communication, and control across mechanical, biological and combined systems [Wie48]. This new perspective helped shape the modern view of control as a universal system principle, leading to its widespread application in fields such as engineering, economics, cognitive science, and biology.

From this foundation, the control of discrete systems evolved into more specialized subfields. One of the most prominent is the control of discrete event systems (DES), introduced by Ramadge and Wonham [RW87]. In DES, the processes under control are modeled as asyn-

chronous, discrete, and potentially nondeterministic systems. These systems are often described by formal languages, where the plant is represented as a language generator and the controller (or supervisor) as a recognizer of a target language. The central problem becomes the synthesis of the largest controllable sub-language that satisfies the desired specifications

One of the major advancement in the discrete systems theory is the development of model checking [CE81; Cla97]. Model checking is a formal verification method used to determine whether a system model (e.g. a Kripke structure [Kri59; Kri63]) satisfies a temporal logic specification (e.g. a LTL [Pnu77]). It has been instrumental in verifying properties such as safety, liveness, and reachability. A *parametric* model checking or *parameter synthesis* then seeks to find the maximal set of parameters for which the property holds [Hun+02; Daw04; Šmi+20].

Logical formulas expressing properties such as a system "eventually reaches a desired state" can be also viewed as a direct reduction of the control problem. This brings us to a provocative question – what is the relationship between control and parametric model checking? Both problems involve identifying choices under which a system satisfies a desired specification, but the nature of these choices differs: control selects interventions, whereas parametric model checking selects parameter valuations. Model checking typically operates within a well-defined domain of parameters and verifies whether a selected property (drawn from a wide variety of types) holds for a given system.

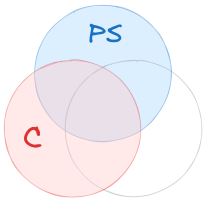
Control theory, on the other hand, tends to focus on a narrower set of property types (i.e., reachability) but allows for greater flexibility in choosing parameters, applying perturbations, or designing other interventions to guide system behavior. Specifically, parameter synthesis typically infers a static system properties, whereas control might also apply temporal interventions. Therefore, narrowing the focus to a specific property type enables more efficient optimization for the problem at hand.

The control of discrete systems eventually extended to the realm of BN theory thoroughly introduced in the previous chapter. Initially, Boolean *control* networks (BCN) were considered as an object of the control [Aku+07; CQL10; Hoc+13; ZGC15; MGFA19]. BCNs are characteristic by containing a set of control (input) nodes and a set of controlled (output) nodes. The control problem lies in the assigning Boolean values to the given control inputs. This problem was extensively studied and successfully solved e.g. using algebraic and semi-tensor product methods [CQL10] or model checking [LJ09].

The BCN control problem can be viewed as being closer to parameter synthesis than to the control of discrete-event systems. This is because the available control inputs are specified in advance, whereas

the control framework considered in this thesis allows greater flexibility in the choice of perturbations. Nevertheless, as discussed later, our approach also employs BN inputs for control, but in a more tailored manner. The control of BCNs is also discussed in more detail in Section 3.3, as it is closely related to the control of synchronous BNs.

3.2 CONTROL OF PARTIALLY SPECIFIED SYSTEMS



The notion of *partial specification* plays a central role in this thesis, and it appears in various forms within discrete system modeling. Broadly speaking, partial specification refers to situations in which some aspects of a system's structure or behavior are unknown, undefined, or only partially observable. This incompleteness poses a unique challenge for control, as it requires strategies that remain effective despite the inherent uncertainty. In the following paragraphs, we discuss several interpretations of partial specification and their relevance to Boolean network control.

Non-observability is considered to be one of the forms of the partial specification, since in such systems, the current state of the system cannot always be determined precisely [Kal60]. This challenge has long been recognized in control theory [Flo62; Sor68], as we can not optimize control for the source state (because it is unknown). Non-observability is particularly common in biological systems where the full internal state is often hidden, or changing too rapidly to measure. [LSB13; Vil19] In the context of Boolean networks (BNs), we adopt a similar view: the internal dynamics are considered unobservable, and only attractors (i.e., stable, long-term behaviors) are assumed measurable. This abstraction is essential in modeling of biological systems, where stable phenotypes correspond to observable steady states (or other attractors). Nonetheless, this thesis is concerned with an even broader and more complex notion of partial specification.

Another important source of uncertainty is *non-determinism*, especially relevant in asynchronous BN. In such systems, the exact transition path from one state to another is not deterministic — multiple successor states may exist. This lack of transition determinism can be viewed as a form of partial specification, as it obscures the system's exact evolution over time. However, in this thesis, we treat non-determinism as an intrinsic property of the system rather than as an imperfection in its specification. The control task must therefore account for all possible transitions, ensuring control guarantee across all potential execution paths.

Nondeterminism also inspired a game theoretic interpretation of control. In this perspective, the system and its uncertain evolution are viewed as an adversarial player, and the controller must identify a winning strategy to guide the system to a desired state [Man19]. This view is particularly useful in multistep control problems, where

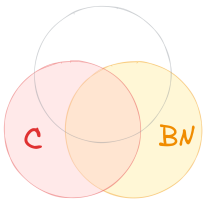
strategies unfold over multiple transitions and must remain effective despite adversarial behaviors. While the game theoretic approach provides valuable conceptual insight, the focus of this thesis lies in simpler control settings (e.g., single-step or permanent control), where the full strategy does not need to be explicitly constructed.

A more structural form of partial specification arises in models such as *probabilistic Boolean networks* (PBNs) [Shm+02], where uncertainty is introduced by allowing each variable to be governed by multiple update functions, each with an associated selection probability. In such models, not only is the update time of variables uncertain, but so is the logic guiding those updates. Thus, control must operate over a probability-weighted ensemble of behaviors, further complicating the synthesis of effective system perturbations.

In contrast, the partially specified Boolean network framework central to this thesis allows each variable to have multiple possible update functions, but crucially assumes that only one "true" function per variable governs the evolution throughout the system's runtime. That is, while the identity of the function is unknown (and remains hidden due to non-observability), it is assumed fixed. The resulting dynamics are therefore equivalent to asynchronous BN under a hidden function instance, though unknown from the observer's perspective. This hybrid interpretation combines partial structural knowledge, non-determinism, and limited observability, creating a novel challenging control problem that this thesis aims to address.

The control of partially specified systems is also closely related to the broader framework of discrete event systems (DES), where uncertainties arise in timing, event ordering, and transition logic. Similar challenges are present in stochastic models such as Markov chains and probabilistic transition systems. These models share structural affinities with PBNs, particularly in how transitions are defined over distributions rather than deterministic rules. These are a few examples which were inspiration for development of the BN. Now let us focus on the control of BN and methods closer to our approach.

3.3 CONTROL OF SYNCHRONOUS BOOLEAN NETWORKS



Synchronous Boolean networks are characterized by the *simultaneous* update of all variables at each time step. This deterministic update scheme ensures that the system's evolution is fully determined by its initial state. Despite this determinism, research on control for synchronous BNs remains highly relevant, especially because it provides foundational insights and inspiration for tackling more complex cases, such as asynchronous or partially specified networks.

As discussed earlier, control of Boolean control networks (BCNs) represents the most basic form of BN control. In BCNs, control inputs can be modeled as direct perturbations of network variables or up-

date rules. Therefore, if one enumerates all possible perturbations as BN inputs and adjusts update functions accordingly, BCN-based control can trivially solve the control problem for synchronous BNs. Nonetheless, this naive approach is exponentially complex. Moreover, despite the deterministic underlying dynamics, the control problem for synchronous BN remains computationally challenging as it was proven to be NP-hard [Aku+07].

Given the Boolean nature of BN, an interesting problem reduction to the *answer set programming* (ASP) was explored in [Kam+13]. The encoding of the problem here is quite straightforward, as the BN can be represented as a logic program, the control problem is formulated as a query, and even perturbations can be expressed as additional rules very conveniently by using three-valued logic. The approach was proven efficient, being handle to solve BN with up to 200 variables and perturbation size 10. A slightly generalized approach was later developed, allowing to specify general changes in long-term behavior (as opposed to output behavior only) [Vid14]. Finally, ASP was also used to not just control the BN but also to infer input-output experiments and design therapeutic targets also packaged as a tool caspo [Vid+17].

Beyond basic control problem, several more refined problem variations have emerged over time. Notably, Kim et al. [KPC13] introduced the notion of a *control kernel*, which is defined as the minimal set of variables that need to be set *initially* in order to drive the system toward a desired attractor. This formulation corresponds to a one-step control problem and is closely aligned with the concept of perturbation strategies discussed in Section 2.3.2.

Another perspective is provided by Choo et al. [Cho+18], who generalized the target of control from a specific attractor to a broader *phenotype*, defined as a set of target states sharing functional or biological characteristics. This abstraction reduces the granularity of control targets and allows for more flexible solution spaces. While computationally simpler than controlling asynchronous or partially specified networks, phenotype-based control provides valuable conceptual tools. Indeed, as discussed in Chapter 5, a similar generalization is central to our own contributions in control of PSBN.

A particularly relevant concept introduced for synchronous BNs is the control of *Boolean network ensembles* [GR16; Cor+18; Gat+21; Vid+17]. In this setting, one considers a family of networks with identical topology but differing update functions. The control problem becomes finding a perturbation that guarantees convergence to the same attractor across *all* network instances. This notion aligns closely with the challenges posed by partial specification as we will see later in this work.

Finally, synchronous dynamics can also serve as a useful tool for analyzing asynchronous networks. Specifically, attractors identified

under synchronous updates can act as approximations or partial indicators of stable attractors in the asynchronous setting [Zhe+13]. While this correspondence is limited to certain classes of attractors (i.e., stable fixed points), it provides valuable structural insights that can guide further analysis of asynchronous dynamics.

3.4 CONTROL OF ASYNCHRONOUS BOOLEAN NETWORKS

Asynchronous BN offer a more biologically realistic modeling framework, as they incorporate asynchronous variable updates [CAS05]. This added expressiveness, however, introduces significant challenges for control, since the system's evolution is no longer deterministic and may follow multiple trajectories from the same initial state. To address this complexity, a broader variety control techniques were developed. In the following subsections, we survey key approaches to the control of asynchronous BN, highlighting the methods employed, the types of perturbations considered, and the formulation of control objectives.

3.4.1 *Refined strong basins search*

Another branch of methods solving the BN control problem is based on the efficient identification of the strong basin of a target attractor (see Section 2.2.4). By definition, a strong basin is the maximal set of states from which only the given attractor is reachable. Once the strong basin is identified, the control task reduces to finding the minimal perturbation that drives the BN into this basin. This is effectively a solution to the source-target control problem using one-step perturbations, since the solution there is as trivial as finding a state from strong basin with minimal Hamming distance to the source state. However, when considering other types of perturbations (e.g., temporary or permanent), additional computational steps are needed to determine appropriate control strategies.

A trivial approach to identifying the strong basin is through a fixed-point algorithm: nodes are iteratively removed from the weak basin if they have transitions that lead outside the basin. When no such nodes remain, the resulting set constitutes the strong basin. While conceptually simple, this method does not scale well for large biological networks. To address this limitation, a more efficient algorithm based on a decomposition of the state-transition graph (STG) into blocks was proposed [Pau+18].

The core idea of this block-based method is to partition the STG into basic blocks, where each block consists of a strongly connected component and its parent states. These blocks are considered in partial projections of the BN (subsets of the full variable set). Once identified, the blocks are topologically sorted and then unfolded, starting

from the block containing the attractor, until all states in the strong basin are identified.

Compared to the stable motif approach that was discussed previously, strong basin identification must be performed separately for each target attractor when solving more complex variants of the control problem, such as full-control or all-pair control. Nonetheless, for smaller BN, the decomposition-based method has shown better performance even in full-control scenarios [PPS18a], whereas for larger networks, the stable motif-based method tends to remain more efficient.

The strong basin identification approach has been successfully applied to source-target control with a variety of perturbation types: one-step initial perturbations [Pau+18; PPS18a], temporary and permanent perturbations [SPP19a], and sequential (attractor-based) variants [Man+19; SP20b]. More recently, the method has been extended to support target control under all types of perturbations [SP20a; Su20]. All of these techniques have been integrated into the software toolkit CABEAN [SP21], facilitating broader adoption and experimentation.

3.4.2 *Stable motifs identification*

Some techniques address complexity of asynchronous BN by analyzing only RN along with the update functions at first. The obtained information can be used to dramatically reduce the size of explicit state space, which needs to be explored to solve the BN control problem.

One such approach is based on identification of *stable motifs* [ZA13; ZA15]. A stable motif is a subset of variables and their corresponding values that, once reached, remain unchanged indefinitely. By identifying all stable motifs, one can construct a so-called *succession diagram* that captures all possible nondeterministic runs of a BN. This diagram provides a structured way to navigate through the system's dynamics and determine the appropriate interventions needed to reach a desired attractor.

The strength of this method lies in its global topological analysis, which enables efficient performance. Once stable motifs are identified, they can be reused to compute control strategies for any attractor within the BN. This facilitates a solution to the full-control variant of the problem and offers valuable insights into the system robustness. However, the method is not complete as it may miss some viable control strategies. Additionally, stable motifs primarily serve as guidance for control design and must be validated through simulation to confirm their effectiveness. This validation also determines whether the required perturbations should be applied transiently (temporarily) or permanently.

To support broader adoption, the method was later implemented as a Python package pystablemotifs, significantly improving its performance as well as accessibility for the research community [Roz+22].

3.4.3 *Trap spaces*

Another approach based observing partially stabilized components is based on observing so-called *value percolations*. This phenomenon is observed if we generalize successor function from a single state to subspaces. This way we can very quickly trace how the reachable subspaces from a given perturbation. Such an approach was first employed by aforementioned approach for the synchronous BN [Kru+11; Vid+17].

The downside of potential percolation-only approach for asynchronous BN is that it is not complete, since it does not guarantee to discover all perturbation strategies. Therefore, it was extended later by combining percolations with trap spaces exploration [CFTS20]. Trap spaces are spaces of the BN which can not be left once reached (see Section 2.2.4).

Compared to the strong basin, however, if we find a trap space in a *synchronous* BN, the trap spaces are also contained in network's *asynchronous* dynamics. Therefore, we can transfer the knowledge obtained from analysis of a simpler dynamics of synchronous BNs and use it for computing control of asynchronous BNs. The perturbations used in this approach are temporary state perturbations.

Since the target of the method is a trap space, the target of the control is also generalized from a single attractor to an arbitrary subspace. This way the method is able to solve the control problem for a wider range of targets (attractors), and allow for more flexible control strategies.

Given the nature of percolations and trapsaces, similar to the synchronous BN approach which inspired this work [Kru+11], methods for control of asynchronous BNs also use ASP to compute the control strategies.

What is also interesting is that logical programming allowed to express the other types of perturbations than just variable perturbation, but also *edge* perturbations, which disable a regulation of one variable to another [FLTS22; CF23].

It is worth mentioning, that there is also a significant body of work focusing solely on efficient computation of trap spaces. Therefore, tools and methods such as PyBoolNet [Kla+18], BioLQM [Nal18], trappist [THB22], or tsconj [Tri+24] can inspire future research of control methods.

3.4.4 *Model checking*

The methods based on percolation and trap spaces were limited to a specific subspace target (i.e., not a combination of multiple independent subspaces). Moreover, they needed to be combined with other approaches in order to obtain a complete solution to the control problem. These caveats inspired the development of a new method based on the model checking approach [CFTS22].

The model-checking approach is using CTL to easily express the control problem. Nonetheless, the general model-checking approach tends to be slow since it explores the entire state space to make sure, that all states fulfill the properties.

Therefore, the control method itself is based on the idea of using a model checking as a "backup" approach if we are unable to assess whether a control strategy works using the trap-space-based method introduced previously. This way we obtain both, high performance for the majority of the cases and a complete solution for the rest.

The approach based on model checking for the phenotype control was implemented based on the Python interface of the BoolNet tool [Kla+18].

3.4.5 *Other techniques*

Finally, we describe other techniques to control asynchronous BN which do not fit into the previous categories, because they use less conventional or a combination of multiple methods.

Kali is a tool for performing in silico therapeutic target discovery using network attractors [PB14; Por17; PG18]. The tool was in its last version updated to handle asynchronous and multivalued BNs.

In this approach, the input is a physiological variant of the network and a pathological variant of the network. The goal is to find perturbations that drive the pathological variant to behave like the physiological one, i.e., to reach the same or at least a subset of the same attractors.

The approach is based on a combination of several techniques. First, the tool identifies all attractors of the BN using random walks. Then, so-called *therapeutic bullets* candidates are generated, using uniform distribution of all possible options. These are combinations of "bullet targets" (states where perturbation is applied) and "bullet values" (the values to which the targets are perturbed).

The candidates are then evaluated by simulating the BN with applied bullet targets. The evaluation is based on the proportion of initial states reaching a physiological attractor or a portion of reachable physiological attractors in case of synchronous BN.

An interesting aspect of the method is, that not only universal perturbations are computed, but also *existential* perturbations evalu-

ated and quantified. Also, the method is focused on refining a model with the same behavior as opposed to driving it towards a specific static set of states. On the other hand, the method is not complete, since it relies on random walks to identify attractors and evaluate perturbations.

Another recently developed approach relies on semi-tensor products [Su+24]. This is another example of successful generalization of a technique working on synchronous BNs to asynchronous BNs. Again, given the nature of semi-tensor products, the control is achieved using periodical impulsive control. The viable controllers are defined along with the model, similarly to the synchronous BCNs mentioned in Section 3.3.

Nonetheless, in this approach, the controllers do not need to be just static Boolean values, but can be also dynamic components that are gaining the value periodically. Moreover, this approach is able to quantify probability of reaching the target states as opposed to inevitably reach the state after applying the control.

3.5 CONTROL OF MOST-PERMISSIVE BOOLEAN NETWORKS

The previously described variants of BN semantics are limited in their ability to capture certain biologically significant behaviors [Pau+20]. For example, none of the earlier models can accurately represent the incoherent feed-forward loop of type 3 (I3-FFL) [MA03], a well-known motif in gene regulatory networks.

Most-permissive semantics enhances the expressiveness of BNs by allowing transient dynamics during state transitions from 0 to 1 and vice versa. This feature enables the representation of intermediate states, capturing subtleties in biological processes that are not strictly binary.

A major motivation for the development of most-permissive semantics lies in its application to model inference. The aim of model inference is to narrow down a broad set of candidate models to those that accurately reflect observed biological behavior. Under more restrictive Boolean semantics, correct models may be incorrectly discarded due to the absence of valid trajectories leading to experimentally observed attractors. As the name implies, most-permissive semantics allow the widest range of behaviors, thereby reducing the risk of false negatives during model selection.

An additional advantage of most-permissive Boolean networks is that despite their increased expressiveness they remain analytically tractable. In particular, Answer Set Programming (ASP) has proven to be well-suited for this framework, as it naturally supports the declarative encoding of Boolean logic and allows efficient searching for trajectories that either confirm or refute specific hypotheses [Pau+20]. However, a notable limitation of this approach is the difficulty in ana-

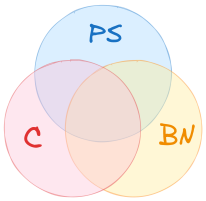
lyzing quantitative properties, such as the size of basins of attraction. For such purposes, simulation-based methods may be used, though they typically trade off completeness and performance [RP22].

The tool BoNesis integrates various techniques for the analysis of most-permissive Boolean networks, including simulation, attractor detection, model synthesis, and control [Pau23]. For symbolic reasoning it leverages ASP, enhanced with counterexample-guided abstraction refinement (CEGAR) [Riv+23], while simulations of network dynamics are performed using an iterative depth-bounded approach.

Control within BoNesis is guided by specifying biological markers, which semantically correspond to the phenotypes introduced in Section 3.4.4. The perturbations computed by the tool are permanent, and the control is universal, meaning that the desired properties must hold across all possible runs of the Boolean network. To minimize the size of the required perturbation set, users can optionally specify an initial configuration, thereby avoiding over-optimization across all possible initial states.

BoNesis supports different control strategies for steady states (fixed points), and all attractors, enabling optimization tailored to task complexity, since reprogramming steady states is typically less computationally intensive [Pau23]. Another notable feature of BoNesis is its ability to control ensembles of Boolean networks, which is especially relevant since model inference usually produces a set of plausible models rather than a single one. When controlling BN ensembles, users can choose between existential and universal control strategies. Existential control identifies perturbations that succeed in at least one model in the ensemble, while universal control yields perturbations that are guaranteed to work across all models.

3.6 LEVERAGING CONTROL FOR MODEL REFINEMENT



Traditionally, the control of Boolean networks has been viewed primarily as a means to modify system's behavior, most notably for applications such as cellular reprogramming. However, recent research has begun to explore control as a valuable asset in the model refinement process. In this context, control can serve several important purposes:

1. Handling uncertainty: Model inference may produce many network models that all fit the data. A good control strategy should work across these variations. If control leads all plausible models to the same desired behavior, it increases our confidence in both the models and the control itself [Che+25].

2. Model verification: Control does not need to be just a downstream consumer of inferred models; it can also act as a verification tool. By applying perturbations and comparing the sys-

tem's behavior to experimental observations, we can check that the model behaves in accordance with observations.

3. Guiding experiments: When experimental data is limited or if there is an abundance of plausible models, control can help design better experiments. It can identify the most informative perturbations; those that produce system behaviors revealing how the network works.

A growing body of work focuses on integrating perturbation data such as gene knockouts and over-expression experiments into the model inference pipeline [Gje+20; Yor+16; PRA25]. These efforts often rely on analyze model ensembles to explore the impact of perturbations. For instance, logic-based approaches can use Satisfiability Modulo Theories (SMT) to impose constraints on system dynamics, enabling the prediction of perturbation effects and facilitating model refinement based on experimental evidence [Yor+16].

Control can also help prioritize new experiments by running in-silico simulations to predict which experiments would be most useful for refining the model. This idea originally came from continuous models, where system analysis has been used to design experiments that help fine-tune model parameters [KT09; Mel+10]. Later, similar strategies were applied to regulatory networks, where perturbation experiments were designed specifically to uncover the network's underlying structure [UD+16].

Similar strategies have been developed for BNs, though with varying degrees of generality and scalability. Early works addressed experiment design for synchronous Boolean networks [Vid+15]. However, the only refined parts are inputs and outputs of the network, thus all available perturbations need to be included into model inputs beforehand.

Another line of work concerned with perturbations experiment design uses a maximum entropy framework to choose experiments that provide the most information [Ati+14]. Still, these methods rely heavily on simulations and are expected to be hard to scale; especially when dealing with asynchronous dynamics, where the number of possible system behaviors grows rapidly.

3.7 RELATED TOOLS

Many of the previously introduced methods were encompassed as software tools in order to make them accessible to a wider audience. Given the numerous nuances between control problems it is not an easy task to exhaustively compare the tools between them. Moreover, some features combinations might not be available.

The first factor which we can compare the tools is the type of dynamics they support. Asynchronous dynamics is more general and bi-

Table 2: **Comparison of related software tools for Boolean network control.** The columns indicate whether the tool supports asynchronous dynamics, the type of control problems it can solve (source-target, target, phenotype), whether it can handle partial specification (e.g., ensembles), the types of perturbations supported (one-step (O), temporary (T), permanent (P)), completeness of the method, availability of a graphical user interface, and the main computational method employed. The checkmarks in gray mean, that features were implemented as part of this thesis. * indicates that control can be achieved indirectly through trap space analysis. ** indicates that control is focused on driving a pathological model towards an archetype model rather than specific phenotypes. *** indicates that for phenotype control, only permanent perturbations are supported.

Tool	Asynchronous	Source-targ.	Target	Phenotypes	Partial spec.	Pert. types	Completeness	Graphical UI	Main method
CABEAN [SP21]	✓	✓	✓			OTP	✓		Strong basin search, ASP
BoolNet [KSS17]	✓	✓*	✓*	✓		OP	✓		Trap spaces, model checking
KALI [PG18]	✓			✓**		P			Simulations
CASPO [Vid+17]			✓	✓	✓	P	✓		ASP
BoNesis [Pau23]	✓	✓	✓	✓	✓	P	✓		ASP
AEON [Ben+20]	✓	✓	✓	✓	✓	OTP***	✓	✓	Symbolic BDDs

ologically relevant, but also more computationally challenging. The majority of the tools support asynchronous dynamics, or were extended to support it. Only CASPO tool does not support asynchronous dynamics.

Secondly, we compare the goal of the control. Note, that BoolNet does not support any specific control per se, but since it supports the identification of trap spaces, it can be easily used to design one-step control strategies based on trap spaces. The tool was also used for methods of phenotype control in [CF23], however to the best of our knowledge, the control methods are not embedded to the tool directly. As for the KALI tool, it is focused on controlling a "pathological" BN based on an "archetype" BN, thus we categorized it as a special case of phenotype control.

As for the partial specification, CASPO and BoNesis tools support Boolean network ensembles, although CASPO is based only on synchronous BNs. Regarding the perturbation types, most of the tools support only permanent perturbations. CABEAN on the other hand, supports the widest range of perturbation types, including one-step, temporary, and permanent perturbations and even sequential perturbations.

Finally, we can also compare the tools based on their completeness and user-faced tools availability. Completeness is a crucial aspect of control methods, as it ensures that all viable control strategies are

identified. The methods based on simulations from KALI tool can not guarantee this property.

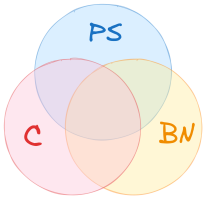
In terms of user-faced availability, the presence of a graphical user interface (GUI) can significantly enhance accessibility for researchers who may not be familiar with command-line tools or programming languages. This capability is present only in AEON, which provides a user-friendly interface for designing partially specified Boolean networks and analyzing their bifurcation and attractor landscapes. Prior to this work, AEON did not support any control methods. Due to its good support of partially specified Boolean networks, we have decided to implement the control methods on top of this tool, thus making it the only tool supporting control of partially specified Boolean networks with a graphical user interface. The details of this implementation are described in Chapter 6.

3.8 SUMMARY

In this chapter, we have provided an overview of the state-of-the-art in the control of Boolean networks, with a particular focus on asynchronous dynamics and partial specification. We have discussed various methods for identifying control strategies, including strong basin search, stable motif identification, trap space analysis, and model checking. Additionally, we have highlighted the importance of control in the context of model refinement and experiment design. Finally, we have compared several software tools that implement these methods, noting their capabilities and limitations. This overview provides the context for the contributions of this thesis, which extend Boolean network control to the partially specified setting.

4

SOURCE-TARGET CONTROL OF PARTIALLY SPECIFIED BOOLEAN NETWORKS



In our work, the source-target control problem provides a natural starting point for the development of control methods and for understanding the challenges arising from the partial specification of Boolean networks. Since the source state is fixed, it is not necessary to analyze the effect of a perturbation on the entire state space; instead, the search for suitable perturbations can be optimized for the given initial state.

We begin with one-step perturbations, which constitute the simplest class of interventions. As will be shown later, computing one-step perturbations is significantly easier than computing permanent or temporary perturbations, because it does not require modifying the transition system of the network to account for sustained intervention.

However, one-step perturbations are often less efficient than perturbations applied over a longer time horizon. Therefore, we also develop methods for permanent and temporary perturbations, which are computationally more complex but may yield smaller perturbation sets.

In this chapter, we first define the source-target control problem in Section 4.1, including the notions of one-step, permanent, and temporary perturbations. Next, in Section 4.2, we present a semi-symbolic method for computing one-step source-target control in partially specified Boolean networks. Finally, in Section 4.3, we extend this method by representing the state space of perturbed networks fully symbolically and to handle permanent and temporary perturbations as well.

4.1 PROBLEM DEFINITION

The cardinal difference between classical Boolean networks and partially specified Boolean networks is that the latter can be interpreted in many ways, depending on the chosen interpretation. As a consequence, every interpretation induces a different instance of the PSBN, which in turn induces a different state transition graph.

On the other hand, the definition of a perturbation is independent of the network's interpretation. A perturbation simply fixes the values of certain variables to constants. Therefore, the definition of a perturbation is very similar to the one used in classical Boolean networks.

What we need to keep in mind, however, is the effect of a perturbation on the network's transition system. In a classical Boolean network, a perturbation can be applied to the whole state space, and it simply removes all states that are not admissible by the perturbation. In a PSBN, however, the effect of a perturbation can differ between different interpretations. This is because the unperturbed variables can still evolve freely. As a result, a perturbation can achieve control for some interpretations, while failing for others.

4.1.1 *Perturbations, perturbed state transition graph and runs*

Here, we build on the definitions of perturbations and their applications from the previous chapter. Most notably, we will leverage the concepts of perturbation (Definition 6), application of a perturbation to a source state (Definition 7), perturbation application to a state-transition graph (Definition 8) and a perturbed run of a Boolean network (Definition 9).

Now, let us remind the definitions of a perturbation and its application to a source state, but in the context of a partially specified Boolean network (although the definitions themselves remain very similar, since they are independent of the network's dynamics):

Definition 12. *Assume having a given PSBN $\mathcal{M} = \{E_1, \ldots, E_n\}$. A* variable perturbation *is then a vector $Q \in \mathbb{B}_*^n$. For every $Q_i = *$, we say that the i-th variable is* unperturbed*, whereas for $Q_i = 0$ or $Q_i = 1$, we say that the variable is* perturbed*. The* size *of Q is the number of perturbed variables: Size$(Q) = |\{i \mid Q_i \neq *\}|$. The set of all considered perturbations is denoted as $\mathbb{Q}$ and typically equals to $\mathbb{B}_*^n$.*

Definition 13. *Given a PSBN $\mathcal{M} = \{E_1, \ldots, E_n\}$,* application of a perturbation $Q \in \mathbb{Q}$ to a source state $s \in \mathbb{B}^n$ *is a* forced *state transition, $\delta(s, Q) = s \to s'$ such that:*

$$\delta(s_i) = s_i' = \begin{cases} s_i & \text{iff } Q_i = * \\ Q_i & \text{iff } Q_i \neq * \end{cases}$$

where s' is the state reached after applying the perturbation.

To facilitate the discussion of temporary and permanent perturbations, we also need to define the effect of a perturbation on the state transition graph of PSBN. Since the dynamics of a PSBN depend on the chosen interpretation, we also take it to the context for the definitions of the perturbed PSBN state transition as well as a perturbed PSBN run:

Definition 14. *Let $\widehat{\mathcal{M}} = \{E_1, \ldots, E_n\}$ be a normalized partially specified Boolean network such that $\widehat{\mathcal{M}}$ admits m nullary function symbols $\mathbb{G} = \{g_1, \ldots, g_m\}$. Given an $\widehat{\mathcal{M}}$, and an interpretation $I \in \mathbb{I}$,* application of a

perturbation $Q \in \mathbb{Q}$ to a PSBN instance $\widehat{\mathcal{M}}(I)$ *with* $\mathsf{STG}(\widehat{\mathcal{M}}(I)) = (\mathbb{V}, \mathbb{E})$ *results in a* perturbed partially specified Boolean network instance $\widehat{\mathcal{M}}(I)_Q$ *with* perturbed state transition graph $\mathsf{STG}_Q(\widehat{\mathcal{M}}(I)) = (Q, \mathbb{E}_Q)$ *where vertices are a subspace of the unperturbed state space* $\mathbb{V} = \mathbb{B}^n$ *given by the perturbation* Q *and a set of perturbed edges* $\mathbb{E}_Q \subseteq \mathbb{E}$ *is given as follows:*

$$\mathbb{E}_Q = \{(u, v) \mid (u, v) \in \mathbb{E} \wedge u \in Q \wedge v \in Q\}$$

The perturbed STG of a PSBN is illustrated in Figure 11.

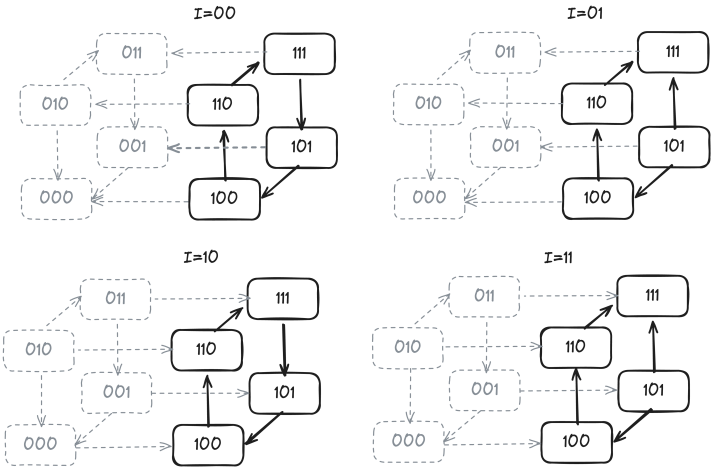


Figure 11: **Perturbed STG of a PSBN.** An example of a PSBN with four interpretations perturbed with a perturbation x1$**$.

Definition 15. *Let us assume an interpretation* $I \in \mathbb{I}$ *of a PSBN* $\widehat{\mathcal{M}} = \{E_1, \ldots, E_n\}$ *with* m *nullary function symbols* $\mathbb{G} = \{g_1, \ldots, g_m\}$ *where* $\mathsf{STG}(\widehat{\mathcal{M}}(I)) = \mathbb{V} \times \mathbb{E}$*. Also, have a given source state* $s \in \mathbb{B}^n$*, a perturbation* $Q \in \mathbb{Q}$*, a and a time period* $z \in \mathbb{N}_\infty$*, after which the perturbation is withheld.* A perturbed run π_Q *is a run which satisfies the following conditions:*

$$\pi_Q(t) = \begin{cases} s & \textit{iff } t = 0 \\ \delta(s, Q) & \textit{iff } t = 1 \\ s' \textit{ s. t. } \pi(t-1) = s'' \wedge (s'', s') \in \mathbb{E}_Q & \textit{iff } t' + z \geqslant t > t' + 1 \\ s' \textit{ s. t. } \pi(t-1) = s'' \wedge (s'', s') \in \mathbb{E} & \textit{iff } t > t' + z \end{cases}$$

A set $\Pi_Q(\mathcal{M}(I), s, z)$ *is a collection of all perturbed runs* π_Q *for a given perturbation* Q*, applied to a state* s *with perturbation being withheld after* z *time-steps.*

Notice, that again the definitions of perturbation and its application to a source state remain almost the same as in the Definition 9. The only differences are that we now also consider the interpretation of the PSBN instance. Moreover, since this chapter focuses on

source-target control, the perturbation is always applied just to a given source state.

4.1.2 *Control types and goal*

Finally, we can define the source-target control problem goal in the context of partially specified Boolean networks. Here, we consider three types of perturbations: one-step, permanent, and temporary perturbations. The definitions of these perturbation types are similar to those used in classical Boolean networks [SPP19b; SP20b], but adapted to the context of partially specified Boolean networks.

Definition 16. *Assume having a PSBN* $\mathcal{M} = \{E_1, \ldots, E_n\}$ *with* m *nullary function symbols* $\mathbb{G} = \{g_1, \ldots, g_m\}$*, an interpretation* $I \in \mathbb{I}$*, a source state* $s \in \mathbb{B}^n$*, and a target state* $t \in \mathbb{B}^n$ *such that* $t \in \bigcup \mathbb{A}(\mathcal{M}(I))$*. Then, we say that a perturbation* $Q \in \mathbb{Q}$ controls $\mathcal{M}(I)$ *from* s *to* t *via:*

- One-step perturbation*: if for all* $\pi \in \Pi_Q(\mathcal{M}(I), s, 1)$, $t \in \inf(\pi)$*. That is, after performing the perturbation for a single time-step (*$\delta(s, Q)$*),* t *is always visited infinitely often without further restricting the network in any way.*
- Permanent perturbation*: if for all* $\pi_Q \in \Pi_Q(\mathcal{M}(I), Q(s), \infty)$, $t \in \inf(\pi_Q)$*. That is,* t *is always visited infinitely often assuming the perturbed variables remain constant to their perturbed value.*
- Temporary perturbation*: if for all* $\pi_Q \in \Pi_Q(\mathcal{M}(I), Q(s), z + k)$*, there exist such time period* z *that for every* $k \in \mathbb{N}_0$ *we have* $t \in \inf(\pi_Q)$*. Intuitively, the system can evolve for an arbitrary number of time-steps while assuming the perturbed variables are constant. But there must exist some time period* z *after which if the perturbation is withheld, run reaches (or remains in)* t *infinitely often.*

Here, note several observations regarding the presented types of PSBN control. The one-step and permanent perturbations are rather straightforward since they either do not require any further adjustment of the network's transition system (one-step perturbation) or they require a simple restriction of the transition system to the subspace of states admissible by the perturbation (permanent perturbation). The temporary perturbation, however, is more complex since the network transition system needs to be adjusted only for a finite number of time-steps. The number of these time-steps is not know in advance or specific since the Boolean network is not considered to be observable. Typically, we assume, that network reaches some "intermediate" attractor within $\mathbb{E}_Q$ an in any of these states, the perturbation can be withheld. Oftentimes, this intermediate attractor

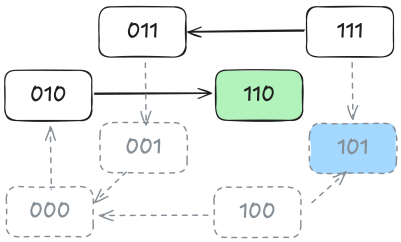


Figure 12: **A specific case of a perturbed STG.** An example of an STG where a temporary perturbation control the network while one-step or permanent perturbations do not guarantee control.

contains the target state t itself. In such case, the exact same perturbation works with both temporary and permanent applications.

Moreover, as already noted, the assumption that target state t is a member of an attractor guarantees that if visited once in an unperturbed system, it is in fact visited infinitely often. Second, permanent control could force a particular t to be visited infinitely often even though it is not a member of any attractor in the original $\mathcal{M}(I)$. This is because the perturbed variables remain fixed indefinitely, as opposed to one-step and temporary perturbations where the restrictions are eventually lifted. We allow for such case in the phenotype control as we see in the following (Chapter 5). Here, to maintain the comparability between all three perturbation types, we strictly restrict ourselves to the interpretations where target is an attractor state.

Note that under this assumption, any successful permanent perturbation is also successful as a temporary perturbation, since the control can be released once target is reached. If we consider the cases where permanent control can introduce the target attractors, this property no longer holds. Nevertheless, a permanent perturbation can also potentially cause target to not be visited infinitely often by disturbing the attractor in which target resides, or it can cause the attractor to be unreachable within the perturbed state space. In this case, such permanent perturbation simply cannot control the network towards the desired target. An example of such scenario (target being unreachable due to permanent perturbation) is shown in Example 3.

Example 3. *Consider STG from Figure 12, source* $s = 101$ *(blue) and target* $t = 110$ *(green) with a perturbation* $Q = *1*$*. By applying the perturbation, the network transitions from* $s = 101$ *to* $\delta(s, Q) = 111$*. In the shown STG, only solid arrows are admissible while the perturbation is applied. The whole (unperturbed) STG then also admits the dashed arrows. As a result, under permanent perturbation, there is no way of reaching* 110 *from* 111*. Meanwhile, in the context of a temporary perturbation,* 011 *corresponds to a state when the control would be released. Once* 011 *is reached, lifting the perturbation guarantees*

that $\mathrm{target} = 110$ *is eventually reached as well. Finally, note that* $Q = *1*$ *is also unsuccessful as a one-step perturbation. This is because by immediately lifting the perturbation,* $\delta(s, Q) = 111$ *can nondeterministically return to* $s = 101$*—reaching target* t *is therefore possible, but not guaranteed.*

4.1.3 *Robustness*

In practice, our goal is to not only find perturbations that successfully control the network, but that are efficient and biologically feasible. Indeed, for a target state t a trivial perturbation $Q = t$ successfully controls the network towards t, but is most likely not realistic. When the network is fully known, we typically focus on finding perturbations that are minimal in terms of size [SPP19b; SP20b]. This search can be further restricted to network variables where the perturbation is known to be biologically feasible if such information is available.

However, if the network is not fully known, there are many possible interpretations, potentially resulting in very different sets of fair runs. In turn, most perturbations correctly control the network only for some subset of these interpretations. Since we do not know which interpretation(s) of the PSBN actually occurs in the wild, we may want to quantify such uncertainty. This allows us to search for perturbations that are not only small (in terms of perturbed variables), but that also work for a reasonable portion of the PSBN interpretations. Formally, this is captured by the metric of perturbation *robustness* [Bri+21]:

Definition 17. *The* robustness ρ *of a perturbation* $Q \in \mathbb{B}^n_*$ *for a PSBN* $\mathcal{M}$ *and a target state* t *is defined as follows:*

$$\rho(Q) = \frac{|\{I \in \mathbb{I} \mid Q \text{ controls } \mathcal{M}(I)\}|}{|\{I \in \mathbb{I} \mid t \in \bigcup \mathbb{A}(\mathcal{M}(I))\}|}$$

Depending on context, the term controls *refers to either one-step, permanent, or temporary perturbation as defined above.*

4.1.4 *Control problem*

As we can see, both size and robustness are critical factors when selecting viable perturbations for a PSBN. To facilitate the search for such perturbations and to compare different perturbations, we define control problem for PSBN as computation of the *control relation* of all perturbations that successfully control the network from a given source to a given target:

Definition 18. *Given a PSBN* $\mathcal{M} = \{f_1, \ldots, f_n\}$ *with* m *nullary function symbols* $G = \{g_1, \ldots, g_m\}$*, a source state* $s \in \mathbb{B}^n$*, and a target state* $t \in \mathbb{B}^n$ *such that there exists an interpretation* $I \in \mathbb{I}$ *for which* $t \in$

$\bigcup \mathbb{A}(\mathcal{M}(I))$, *the goal of the* source-target control problem *is to compute a* control relation $\mathbb{C}_X$ *– the set of all pairs* $\mathbb{C}_X = \{Q, I\} \subseteq \mathbb{B}_*^n \times \mathbb{I}$ *such that* Q *controls* $\mathcal{M}(I)$ *from* s *to* t *assuming a type of perturbations* $X \in \{O, P, T\}$ *as one-step (*$\mathbb{C}_O$*), permanent (*$\mathbb{C}_P$*), or temporary (*$\mathbb{C}_T$*) control.*

The robustness measure gives us the ability to compare the "quality" of a single perturbation. To also compare the "quality" of different control relations (or control approaches in general), we extend the notion of robustness to be measurable on a control relation. Specifically, we define *maximal robustness* $\rho_{max}(\mathbb{C}_X)$ to be the maximal robustness achievable by a single perturbation from $\mathbb{C}_X$, and *union robustness* $\rho_\cup(\mathbb{C}_X)$ to be the collective robustness achievable by all available perturbations:

$$\rho_{max}(\mathbb{C}_X) = \max_{Q \in \mathbb{Q}} \rho(Q)$$

$$\rho_\cup(\mathbb{C}_X) = \frac{|\bigcup_{I \in \mathbb{C}_X} I|}{|\{I \in \mathbb{I} \mid t \in \bigcup \mathbb{A}(\mathcal{M}(I))\}|}$$

Intuitively, the maximal robustness ρ_{max} gives us the best case scenario that can be achieved using a single perturbation from $\mathbb{Q}$. Meanwhile, union robustness $\rho_\cup$ is the measure of how well a system can be controlled *overall*. In particular, if union robustness is less than one, there are PSBN interpretations that cannot be controlled by any perturbation in $\mathbb{Q}$. Both measures are thus worth exploring, as each represents a slightly different optimization goal: While a control relation $\mathbb{C}_X$ with high ρ_{max} is generally more likely to provide a practically viable control strategy, another control relation with a higher $\rho_\cup$ may achieve control even for cases that are not feasible by the former $\mathbb{C}_X$.

4.1.5 *Summary*

To summarize, in this section we defined the source-target control problem for partially specified Boolean networks. We defined three types of perturbations: one-step, permanent, and temporary perturbations. Then, we defined the goal of the source-target control problem as computation of the control relation containing all perturbations that successfully control the network from a given source to a given target. Finally, we introduced the robustness metric to quantify the uncertainty of a perturbation in terms of the portion of PSBN interpretations it can control.

4.2 SEMI-SYMBOLIC METHOD FOR ONE-STEP CONTROL

Now we describe our computational framework for solving the one-step state perturbation control of PSBN. We start by introducing our approach for finding strong basins in a PSBN. Then we explain the

framework for exploring PSBN STG and for manipulation with PSBN interpretations. Next, a concise workflow for PSBN control that employs the proposed algorithms is demonstrated. Finally, we present an evaluation of the method on a collection of PSBN models.

4.2.1 *Semi-symbolic labelled strong basin search*

The foundation of our method is based on binary decision diagram (BDD [Bry86]) representation of interpretation sets of a PSBN. The decision variables of the BDD are the nullary function symbols of the network meaning that every path from the root to a leaf in such a BDD represent a set of interpretations of the PSBN while states are represented explicitly.

Common logical operations on such BDDs (and, or, negation, ...) then correspond to set operations (intersection, union, complement, ...). Furthermore, static regulation constraints (activation, inhibition, observability) can be formalized using Boolean formulae over interpretations, and we can, therefore, create a BDD which enforces all constraints imposed by the regulatory network and represents the set of only relevant interpretations.

As described in Section 2.4.3, PSBN dynamics $\mathrm{STG}(\mathcal{M})$ are represented as an edge-labelled state-transition graph, where each transition $s \to t$ has an associated set of interpretations $\mathcal{I}(s,t) \subseteq \mathbb{I}$ represented as a BDD. A *labelled state set* is a mapping $\mathbb{V} \to 2^{\mathbb{I}}$ assigning to each state a set of interpretations. Furthermore, we suppose that the state space is represented *explicitly*, meaning that all operations on states are performed element-wise (typically in parallel).

We consider basic procedures over an STG of a PSBN. Given a source state s and a viable interpretation set $\mathcal{I}$ of a PSBN $\mathcal{M}$, procedures Pre and Post yield a set of labelled states reachable in one step backward/forward from s under interpretations from $\mathcal{I}$:

$$\textsc{Pre}(s,\mathcal{I}) = \{t \mapsto \mathcal{I}_t \mid \forall I \in \mathcal{I}_t : I \in \mathcal{I} \wedge s \to_I t\}$$
$$\textsc{Post}(s,\mathcal{I}) = \{t \mapsto \mathcal{I}_t \mid \forall I \in \mathcal{I}_t : I \in \mathcal{I} \wedge t \to_I s\}$$

These procedures can be then used (e.g. using a fixed-point computation) to compute a maximal labelled state set of *all* forward/backward reachable states from the source state s. The set contains all reachable states t where each t is associated with a maximal set of interpretations $\mathcal{I}_t$ for which t is reachable from s:

$$\textsc{Fwd}(s,\mathcal{I}) = \{t \mapsto \mathcal{I}_t \mid \forall I \in \mathcal{I}_t : I \in \mathcal{I} \wedge s \to_I^* t\}$$
$$\textsc{Bwd}(s,\mathcal{I}) = \{t \mapsto \mathcal{I}_t \mid \forall I \in \mathcal{I}_t : I \in \mathcal{I} \wedge t \to_I^* s\}$$

This representation (explicit state space and symbolic interpretations) allows computing the reachability procedures in parallel. For

the underlying implementation, we worked with internal libraries of the tool AEON [Ben+20] which provides most of the necessary functionality, including a convenient format for specifying PSBNs and parallel reachability procedures.

For control of PSBN, we first need to determine the relevant interpretations $\mathcal{I}_A(t)$ for the target state t, where t is a part of some attractor. This process is described in Algorithm 1. The algorithm is following: we first compute all reachable states, but only the one from which we can also reach back the target state itself are a part of a TSCC. Otherwise, it is possible to reach some other component of the system from the target state, and that contradicts the notion of attractor. In the partially specified setting, we obtain a mapping of states to interpretations in which it is not possible to return to the target again. Therefore, in all these interpretations the target state is not a part of any attractor, and so we discard them from the full parameter space $\mathbb{I}$ to obtain $\mathcal{I}_A(t)$.

Algorithm 1: Computation of attractor interpretations $\mathcal{I}_A(t)$.

```
1 Fn AttInterp(t ∈ 𝕍)
2     F ← Fwd(t, 𝕀);
3     B ← Bwd(t, 𝕀);
      /* For every state, compute the BDD difference   */
4     NA ← F \ B;
5     return 𝕀 \ ⋃_{s∈𝕍(ℳ)} NA(s);
```

The key observation for controlling PSBNs is that an attractor is always reached from its strong basin (as explained in Definition 5). From all other states, it is possible to reach also some other attractor; therefore, reaching the target attractor is not guaranteed. When the attractor's strong basin for all interpretations is known, we can compute the control mapping $\mathbb{C}_O$ from a source state to the target attractor by considering Hamming differences between the source and the states of the strong basin.

Our algorithm is based on the fixed-point approach for strong basin computation in fully specified BNs [PPS18a]. The premise is that only states from which it is possible to reach the attractor (weak basin) can be part of the strong basin. The weak basin of the attractor is computed using standard reachability algorithms, for example, BFS. Then states are iteratively removed if it is possible to leave the basin from them (and therefore not reach the given attractor). Finally, if there are no states left to remove, the fixed-point is achieved, and the strong basin is found.

In Algorithm 2, we extend the approach from [PPS18a] onto PSBN. For each state we remember under which interpretations it is in the basin, starting with the labelled weak basin (backward reachability from the target state). Then we iterate over the basin's states. For each

Algorithm 2: Parallel strong basin computation.

```
1  Fn StrongBasin(𝓜, t ∈ 𝕍(𝓜), ℐ_A(t))
2      SB ← Bwd(t, ℐ_A(t));
3      toUpdate ← {s ∈ 𝕍(𝓜) | SB(s) ≠ ∅};
4      while toUpdate ≠ ∅ do
5          updated ← ∅;
6          toRemove ← ∀s ∈ 𝕍(𝓜) ↦ ∅;
7          parallel for s ∈ toUpdate do
8              for u ∈ Post(s) do
                   /* Recompute interpretations leading
                      outside of basin                    */
9                  toRemove(s) ← toRemove(s) ∪
                    (SB(s) ∩ ℐ(s,u) ∩ (ℐ_A(t) \ SB(u)));
10             if toRemove(s) ≠ ∅ then
11                 updated ← updated ∪ {s};
12         toUpdate ← ∅;
13         for s ∈ updated do
               /* Update strong basin                    */
14             SB(s) ← SB(s) \ toRemove(s);
               /* Only predecessors of updated states might
                  be updated in the next iteration       */
15             toUpdate ← toUpdate ∪ {s} ∪ Pre(s);
16     return SB;
```

state, we consider its successors, and we compute the interpretations for which some successor is *not in the strong basin*. In these interpretations, it is possible to leave the basin, and therefore these interpretations are not part of the strong basin and so we remove them for this state.

We can process the states in parallel and compute for them the interpretations which are not part of the strong basin. To avoid writing to the strong basin structure while it is intensively read from, we store the interpretations which should be removed separately, and after it is computed, we update the strong basin structure sequentially. The procedure is repeated until nothing more can be removed from the strong basin.

4.2.2 *Control computation workflow*

Now we can describe a complete workflow for computing the source-target control in PSBNs, depicted in Figure 13. The workflow consists of three inputs and three computation steps, resulting in the control mapping $\mathbb{C}_O$.

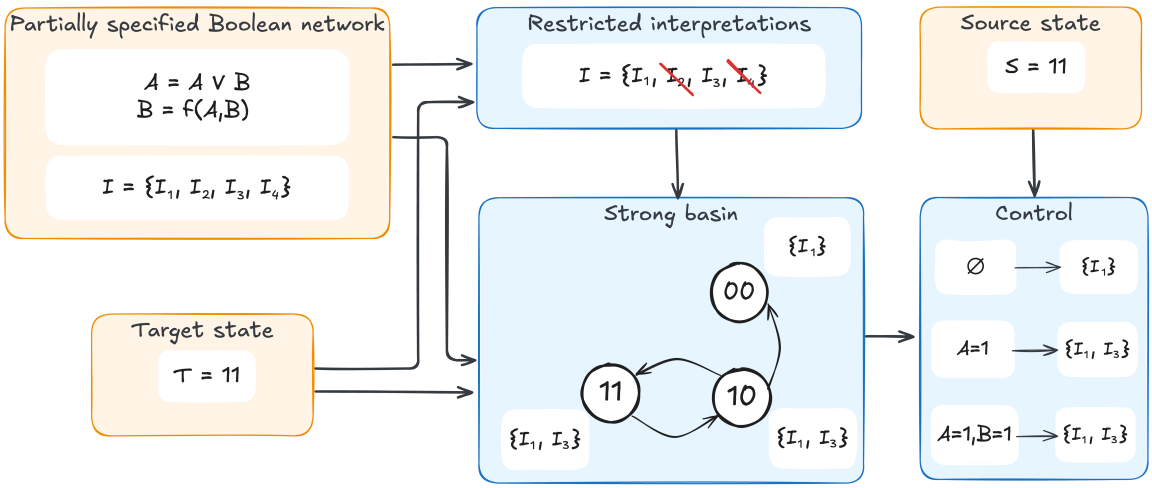


Figure 13: **Workflow of computing the source-target control problem.** Parts in the orange boxes represent inputs and blue boxes represent (intermediate) results.

Given an input PSBN and a target attractor state t, we start by computing valid interpretation set $\mathcal{I}_{\mathrm{A}}(\mathrm{t})$ using Algorithm 1. Then, the labelled strong basin is computed from the PSBN with target attractor state t and its valid interpretations $\mathcal{I}_{\mathrm{A}}(\mathrm{t})$ using Algorithm 2. After that, from the strong basin and the source state, we obtain the complete control mapping. Notice that we do not need to know the source state for computing the strong basin and that we can re-use the target's strong basin for obtaining control for different sources.

To compute the control mapping, observe that viable controls correspond exactly to the Hamming differences (the variables with opposite values) between the source state and the states of the strong basin; yielding one viable perturbation Q for every state s in the strong basin SB. Any other control Q' does not reach the strong basin and therefore does not guarantee to reach the target attractor (for these, $\mathbb{C}_{\mathrm{O}}(\mathrm{Q}) = \emptyset$). A perturbation Q is then viable only for interpretations for which s appears in the strong basin, we thus set $\mathbb{C}_{\mathrm{O}}(\mathrm{Q}) = \mathrm{SB}(s)$.

Finally, we can compute the size and robustness of each control as we have all knowledge regarding the variables which need to be controlled and interpretations for which the controls work. If it is desired, we can construct a *witness* PSBN instance for a perturbation Q (a fully specified BN where the given control works) by fixing the parameters from $\mathbb{C}_{\mathrm{O}}(\mathrm{Q})$.

In complex and highly underspecified models, the empty perturbation $\emptyset$may satisfy the control objective for some interpretations. This occurs when, under a given interpretation, the source state already lies in the strong basin of the target, and therefore no intervention is required. If such interpretations are considered unrealistic, for example because they contradict the assumption that an intervention is

needed, the set $\mathcal{I}_A(t)$ can be replaced by a custom set of admissible interpretations.

The resulting set of all available controls can be then used for example for cell reprogramming. However, we might obtain many possible controls, and we need to decide which one should be applied. To do that, we can decide based on the size of control or its robustness. The control set can be arbitrarily filtered discarding controls with size bigger than the trivial control (which always has 100% robustness) or bigger than some set size. Similarly, the control relation can be pruned based on too low robustness. It is left to the actual application to decide which control would best suit its needs. Namely, whether it is more important for the control set to be small or to be robust, as typically, there is rarely available perturbation which would be optimal in both these factors.

4.2.3 *Results*

We evaluate our approach on two real-life BN models. We compare the performance of our approach using a different number of parameters implanted into the models, resulting in different size of relevant parameter space. We conducted all measurements using a machine equipped with AMD Ryzen Threadripper 2990 WX 32-Core Processor and 64 GB of memory.

The first model is a *cell-fate decision model* [Cal+10]. The model provides a high-level view of possible different cell fates such as pro-survival, necrosis or apoptosis. We used an adapted version of this model, where only regulatory network is provided and all the functions are left unspecified. Then, we selected seven biologically relevant attractors and computed strong basins only for these attractors in several model instances differing in the number of unknown parameters (see Table 3).

The second model, a *myeloid differentiation network* [Kru+11], was designed to model a bone marrow tissue cell differentiation from common myeloid cell to specialised blood cells (megakaryocytes, erythrocytes, granulocytes, and monocytes). The original network has eleven nodes and six attractors. We derived several partially specified versions of the model by arbitrarily substituting update functions of the model for function symbols. Similarly to the previous case, we used only attractors of the original network when computing strong basins for partially specified versions of the model. The results are again shown in Table 3.

Next, we “virtually” compare our approach to a naïve approach based on an interpretation enumeration (i.e., computing results for each interpretation separately). In [Bau+19], a strong basin of (fully specified) asynchronous BNs is computed using a block decomposition method with 4 ms needed to finish the computation for the

Table 3: Results of strong basin computation. The values are stated as ranges because we computed strong basins of all attractors considered in the given models. The second column shows the number of model's parameters. The third column shows count ranges of interpretations which contain the given attractor. The fourth and fifth columns display ranges of the number of states in the weak (resp. strong) basins. The last column contains ranges of times needed to compute strong basins.

Model	$\lvert\mathbb{G}\rvert$	$\lvert\mathfrak{I}_A(t)\rvert$	**# WB States**	**# SB States**	**Time**
Cell-Fate	1	1	258,000–491,184	32–352	4.4–9.19 s
	8	1–4	258,000–491,464	21–79	4.63–13.42 s
	20	7–56	258,048–491,520	1632–262,144	5.82–26.59 s
Myeloid	1	1	128–1152	64–384	8–30 ms
	32	63–2052	224–1984	64–1472	14–214 ms
	70	5.9×10^4–1.8×10^7	1512–2048	256–2048	147–1717 ms
	94	3.4×10^6–3.7×10^9	2008–2048	1024–2048	0.6–15.38 s

Table 4: **Scalability of strong basin computation.** The strong basins are computed on attractors of myeloid model [Kru+11].

CPUs	A_1	A_2	A_3	A_4	A_5	A_6
1	6.13 s	99.31 s	71.32 s	130.17 s	45.84 s	136.65 s
2	3.34 s	54.32 s	38.87 s	71.31 s	24.95 s	74.29 s
4	1.86 s	31.3 s	21.83 s	40.31 s	13.71 s	42.26 s
8	1.11 s	19.34 s	13.31 s	24.68 s	8.4 s	26.49 s
16	0.87 s	14.03 s	9.32 s	17.56 s	5.77 s	18.86 s
32	0.6 s	10.98 s	6.87 s	13.22 s	4.54 s	15.38 s

myeloid model. Even if the reported hardware was slower than in our case, and we assume that the strong basin computation for one interpretation would last only 1 ms, the fully unspecified myeloid model contains an attractor which is present in 3.7×10^9 interpretations. Therefore, the expected time for computing a strong basin for all interpretations with 32-fold parallelization is more than a day compared to less than 27 s achieved using our semi-symbolic approach.

We evaluate the scalability of our approach on a fully unspecified myeloid model. The results are shown in Table 4. The computation was restricted to the specified amount of CPUs. The final speed-up achieved on our machine, when using 32 CPUs compared to a non-parallel CPU usage was about 10-fold.

Now let us have a look at an example of how the results might be interpreted and a suitable control selected. Suppose that we want to reprogram an erythrocyte cell (attractor having factors `EKLF=1` and `GATA-2=0`) into a monocyte cell (attractor having factors `cJun=1` and

EgrNab=1) of the myeloid model. First, we obtain a strong basin of the monocyte attractor having 1472 states yielding us 1472 possible control sets. We can observe that trivial, i.e., the most robust control strategy (setting variables so that we reach the attractor right after applying the control), has size 8. We can discard all controls with size 8 and larger as they are less optimal in all aspects than the trivial control. In our case, there are 1259 controls with a smaller size than the trivial one.

Resulting smallest control sets have the size 1. However, the best robustness among these controls is 46%. Therefore, it is quite likely that it will not work in practice. If we allow the control to have size 2, we can use the control with 76.8% robustness. Increasing the size further provides control with the size 3 and the robustness 92% and so is highly likely to reprogram the cell's phenotype.

To achieve only slightly more robustness (93.8%), we may use a control of size 5. In this highly unspecified model, there is no control with 100% robustness being smaller than the trivial one. It is not a rule of the thumb that with bigger size of control we obtain better robustness, for example, the control with the size 11 (the only one, with all variables, reversed compared to the original one) has the robustness of only 50%.

It can be seen that the unknown properties of PSBN make selection of one particular "best" control complicated. The smallest control has low chance to work in reality and while control of size 3 works has significantly better chance to work, the success of the control is not guaranteed and might be difficult to implement. This is why a careful in vitro experimentation is needed to verify the correctness of the control in reality. Nonetheless, the obtained control set still can help and highly reduce the exponential number of potential transcription factor combinations that could be theoretically tested in vitro.

4.2.4 *Summary*

In this section, we presented a semi-symbolic method for computing the source-target control of PSBN using one-step perturbations. The method is based on a fixed-point computation of the labelled strong basin of the target attractor. We demonstrated that our method is capable of controlling highly unspecified models in seconds, and is considerably faster than the naïve interpretation scan approach.

4.3 SYMBOLIC METHOD FOR ONE-STEP, TEMPORARY AND PERMANENT CONTROL

In the previous section, we presented a method for computing the source-target control of PSBN using one-step perturbations. We have seen, that control size for one-step perturbations could be improved.

To address this issue, we can consider more general types of perturbations, such as permanent or temporary perturbations. Moreover, in the meantime, the tool AEON was advanced to support symbolic representation of states as well [Ben+22b], which allows us to explore the whole perturbation space in parallel. Therefore, we can now propose a fully symbolic method for computing the source-target control of PSBN using one-step, temporary, and permanent perturbations.

4.3.1 *Symbolic computation model*

Similarly to the previous method, we employ symbolic representation using binary decision diagrams (BDDs) [Bry86]. However, in this case, we extend the use of BDDs to represent not just interpretation space symbolically, but also the sets of states. The main advantage of BDDs is that a logical operation (e.g. a conjunction) can be performed directly on two operand diagrams, maintaining the compact representation throughout the whole computation. Due to this correspondence, we do not differentiate between a BDD and the Boolean function that it represents. Furthermore, note that any set or relation X consisting of Boolean vectors (e.g. $X \subseteq \mathbb{B}^k$) can be represented by a Boolean function that returns 1 exactly for the members of X and 0 otherwise. Therefore, any such X can be also represented using a BDD. In particular, this includes subsets of network states ($X \subseteq \mathbb{B}^n$), subsets of PSBN interpretations ($X \subseteq \mathbb{B}^m$), and relations of the two ($X \subseteq \mathbb{B}^n \times \mathbb{B}^m$). Logical operations on BDDs then correspond to set operations (e.g. $\cap \equiv \wedge$, $\cup \equiv \vee$, etc.).

Another important aspect is that a perturbation Q is not a Boolean vector, but rather a vector over $\mathbb{B}_*$. For our purposes, however, the algorithms do not need to manipulate whole perturbations. Instead, we only encode which variables in the network are perturbed ($q_i = 0$ or $q_i = 1$) and which are free ($q_i = *$). For this task, subsets of $\mathbb{B}^n$ are sufficient. To summarize, we use the following notation:

- $\mathbb{V} \equiv \mathbb{B}^n$ encodes network states (vertices of STG($\mathcal{M}$));
- $\mathbb{I} \equiv \mathbb{B}^m$ encodes network interpretations (i.e. the set of all possible function symbols valuations);
- $\mathbb{L} \equiv \mathbb{B}^n$ encodes sets of perturbed variables (without the actual perturbation value).

Similar to the previous method, target state t may not be an attractor state in all interpretations of the given PSBN. Consequently, instead of $\mathbb{I}$, we again consider a smaller subset of interpretations $\mathcal{I}_A \subseteq \mathbb{I}$ where t is guaranteed to be a member of some attractor. As this requires no functional changes to the presented algorithms, we generally use the set $\mathcal{I}_A$, with the implicit assumption that a smaller $\mathcal{I}'$ can be also used when desired.

The reason why we do not have to encode full perturbations is because in our algorithms, we use a relation $X \subseteq \mathbb{V} \times \mathbb{I} \times \mathbb{L}$ to encode the full state of the system. In such an X, we assume that if the network is perturbed, only the states that are compatible with the perturbation are permitted (this invariant is maintained throughout the algorithms). Consequently, in a triple $(v, I, l) \in X$ where l_i encodes whether a variable is perturbed, v_i encodes either the state of the network (when $l_i = 0$, i.e. unperturbed) or the constant value of the perturbation (when $l_i = 1$, i.e. perturbed).

We then use $\mathbb{X} = \mathbb{V} \times \mathbb{I} \times \mathbb{L}$ as a shorthand for this extended set of perturbable states and network interpretations. We also write that a pair $(v, l) \in \mathbb{V} \times \mathbb{L}$ is *equivalent* to a perturbation $Q \in \mathbb{B}_*^n$, $(v, l) \equiv Q$, if $Q_i = *$ when $l_i = 0$, and $Q_i = v_i$ when $l_i = 1$.

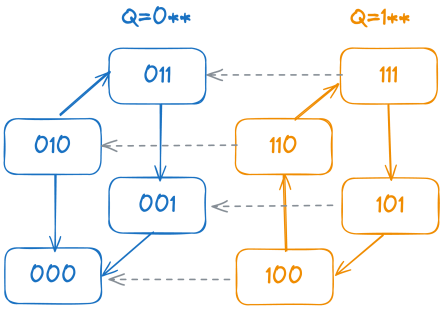


Figure 14: **Encoding of perturbations.** Illustration of how perturbations are encoded using pairs of network states and perturbation masks.

Observe that this approach is more efficient than simply incorporating the set of all possible perturbations $\mathbb{B}_*^n$ directly into the encoding. Each $\mathbb{B}_*$ element would require more than one bit per network variable to encode, whereas our approach only adds one bit on top of the existing state space. The details of the decoding for the actual perturbations will be described in Section 4.3.3.

4.3.2 *Symbolic operations*

Aside from standard set operations ($\cap, \cup, \setminus, \times$, etc.), we also use $\exists_{\mathbb{L}}(X \subseteq \mathbb{X})$ to denote existential projection over the perturbed variables, i.e. $\exists_{\mathbb{L}}(X) = \{(s, c) \mid \exists l \in \mathbb{L}\ (s, c, l) \in X\}$. This can be implemented as a single operation on the directed graph of a BDD as well. Finally, we need a mechanism to symbolically explore the graph $\text{STG}(\mathcal{M})$ that is compliant with the applied variable perturbations. For this purpose, we define the following two operations:

$$\begin{aligned}\textsc{Pre}(X \subseteq \mathbb{X}) = \{(s, I, l) \mid \exists (t, I, l) \in X \wedge s \neq t \\ \wedge \exists i \in [1, n].t = s[i \mapsto f_i(s) \text{ of } \mathcal{M}(I)] \wedge l_i = 0\} \\ \textsc{Post}(X \subseteq \mathbb{X}) = \{(t, I, l) \mid \exists (s, I, l) \in X \wedge s \neq t \\ \wedge \exists i \in [1, n].t = s[i \mapsto f_i(s) \text{ of } \mathcal{M}(I)] \wedge l_i = 0\}\end{aligned}$$

Recall that the conditions above are almost exactly the condition under which $(v, I, u) \in E$ of STG$(\mathcal{M})$ (as per Definition 14). That is, Pre and Post compute the sets of predecessors and successors of the states in X under their respective $\mathcal{M}$ interpretations. However, for each computed transition, we also require that $l_i = 0$, i.e. the i-th variable is *not* perturbed. Hence values of the perturbed variables remain constant. For example, when $l_i = 1$ for all $i \in [1, n]$, no transition is possible. Meanwhile, when $l_i = 0$ for all $i \in [1, n]$, we get exactly the behaviour of STG$(\mathcal{M})$ without any perturbations. These two operations can also be implemented using BDDs, since P_i is an expression over Boolean variables (which trivially corresponds to a Boolean function and hence a BDD). The rest are either pure logical operations or quantifications that can be realised as BDD operations.

4.3.3 *Supporting algorithms*

Our main algorithms rely on a collection of utility procedures defined in Algorithm 3. Before we proceed with the main result, we quickly explain the rationale behind these procedures.

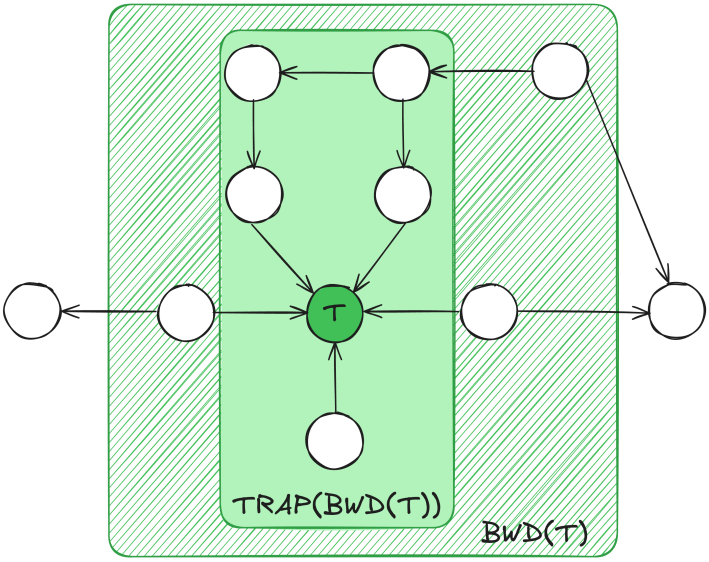


Figure 15: **Bwd and Trap procedures.** An intuitive example demonstrating the Bwd and Trap procedures from Algorithm 3. The highlighted dark green state is the target attractor state T. The shaded area represents the set Bwd$(\{T\})$ (weak basin of T). The solid green area represents the set Trap$(\text{Bwd}(\{T\}))$ (strong basin of T).

Bwd This is a standard backward reachability procedure. That is, for the given set of initial states, it computes all transitive predecessors of these initial states. It also uses our modified notion of state space $\mathbb{X}$ which incorporates information about the perturbed vari-

ables. That is, if a variable is perturbed, it cannot be updates (recall our definition of PRE).

TRAP This procedure computes a generalized variant of *trap sets* as defined in Section 2.2. The procedure takes a set of initial labelled states $X \subseteq \mathbb{X}$ and iteratively eliminates any components that have outgoing transitions leading out of X. For unperturbed components, this corresponds exactly to the notion of the trap set (grouped by possible interpretations encoded in the component C of $\mathbb{X}$). For cases where the network is perturbed, the result is still a maximal subset of X with no outgoing transitions, however, only with respect to the unperturbed variables.

Figure 15 shows an example demonstrating the use of BWD and TRAP (for simplicity, we do not consider multiple network interpretations or perturbations here). Given a target t attractor state, the figure shows the set of all backwards reachable states $\text{BWD}(\{t\})$ (weak basin of t). Set $\text{TRAP}(\text{BWD}(\{t\}))$ then depicts the maximal trap set contained within $\text{BWD}(\{t\})$ (strong basin of t).

CANPERTURB Observe that while all procedures introduced so far *respect* existing perturbations (by the means of the $\mathbb{L}$ component of $\mathbb{X}$), no procedure actually *performs* the perturbation step from source s to $\delta(s, Q)$ as assumed by all three BN control approaches. However, since our control method works in a bottom up manner, we actually seek to answer a slightly modified question: given a set $T \subseteq \mathbb{V}$, what subset $T' \subseteq T$ is reachable from source s by an admissible perturbation? Formally, the goal is to compute triples $(s, I, l) \in \mathit{targets}$ satisfying $\delta(\mathit{source}, Q) = s$ for $Q \equiv (s, l)$ (i.e. Q is the perturbation encoded within the (s, I, l) triple).

To compute $\text{CANPERTURB}(s, X)$, need we respect the following: Let $(s', I, l) \in X$ be a triple such that $s'_i \neq s_i$ and $l_i = 0$ for some $i \in [1, n]$. That is, source s and s' differ in the value of the i-th variable, but the variable is marked as unperturbed in l. Then, and only then we have that $\delta(s, Q) \neq s'$ for $Q \equiv (s, l)$. In other words, every (s, c, l) for which such variable exists must be removed from X when computing CANPERTURB. Algorithm 3 thus iterates through all network variables, constructing a symbolic representation of the set of triplets which violate the aforementioned condition, and subtracting this set from the original X. The result is a subset of the X' relation where every member is reachable from source s via its respective perturbation.

4.3.4 *Control algorithms*

In Algorithm 4, we describe the bottom-up procedures which compute the relations $\mathbb{C}_O$, $\mathbb{C}_P$, and $\mathbb{C}_T$. Each algorithm relies on the pro-

Algorithm 3: Basic graph algorithms.

```
1  Fn Bwd(X ⊆ 𝕏)
2      repeat X ← X ∪ Pre(X) until X reaches fixpoint;
3      return X;
4  Fn Trap(X ⊆ 𝕏)
       /* Eliminate elements that escape from X in one
          step.                                          */
5      repeat X ← X \ Pre(Post(X) \ X) until
        X reaches fixpoint;
6      return X;
7  Fn CanPerturb(s ∈ 𝕍, X ⊆ 𝕏)
8      for i ∈ {1...n} do
9          diff_i ← { s' ∈ V | s'_i ≠ s_i };
10         unperturbed_i ← { l ∈ 𝕃 | l_i = 0 };
11         X' ← X \ (diff_i × 𝕃 × unperturbed_i);
12     return X';
13 X
```

cedures from Algorithm 3, but incorporates specific adjustments to accommodate the temporal aspects of a specific control approach.

PERMANENT The procedure for computing permanent control is the most straightforward. First, we use $\textsc{Bwd}(\{t\} \times \mathbb{I} \times \mathbb{L})$ to determine all states that are able to reach t across all possible PSBN interpretations ($\mathbb{I}$) and combinations of perturbed variables ($\mathbb{L}$). Procedure Trap then retains only the states from which all runs are guaranteed to eventually reach t. This follows from the fact that t is a member of the only attractor within the initial set, hence fair runs that do not leave $\textsc{Bwd}(\{t\} \times \mathbb{I} \times \mathbb{L})$ must eventually reach it. Finally, CanPerturb is used to determine which of these cases are actually admissible under the given source s . The result is a $\mathtt{map} \subseteq \mathbb{X}$ which describes the control relation $\mathbb{C}_P$. As $\mathbb{C}_P$ is not a Boolean relation, it does not have a direct BDD representation—instead, the return statement describes how members of $\mathbb{C}_P$ correspond to the members of map. Intuitively, map can be seen as a simple Boolean encoding of a non-Boolean relation $\mathbb{C}_P$.

ONE-STEP Since one-step control does not assume that perturbation is held over time, component $\mathbb{L}$ in the initial states is substituted for $\{0\}^n$. This is a singleton set that only admits the case where no perturbation is applied. By computing $\textsc{Trap}(\textsc{Bwd}(\{t\} \times \mathbb{I} \times \{0\}^n))$, we then get the combinations of states and interpretations which guarantee that t is eventually reached (and visited infinitely often) in the unperturbed network. This corresponds to the strong basin of t.

Algorithm 4: Control procedures.

Fn PERMANENT($\mathtt{s} \in \mathbb{V}, \mathtt{t} \in \mathbb{V}$)
  $\mathtt{basin} \leftarrow \textsc{Trap}(\textsc{Bwd}(\{\mathtt{t}\} \times \mathbb{I} \times \mathbb{L}))$;
  $\mathtt{map} \leftarrow \textsc{CanPerturb}(\mathtt{s}, \mathtt{basin})$;
  return $(\mathrm{Q}, \mathrm{I}) \in \mathbb{C}_{\mathrm{P}} \Leftrightarrow (\mathtt{s}, \mathrm{I}, \mathtt{l}) \in \mathtt{map} \wedge (\mathtt{s}, \mathtt{l}) \equiv \mathrm{Q}$;
Fn ONESTEP($\mathtt{s} \in \mathbb{V}, \mathtt{t} \in \mathbb{V}$)
  $\mathtt{basin} \leftarrow \textsc{Trap}(\textsc{Bwd}(\{\mathtt{t}\} \times \mathbb{I} \times \{0\}^n))$;
  $\mathtt{basin} \leftarrow \exists_{\mathbb{L}}(\mathtt{basin}) \times \mathbb{L}$;
  $\mathtt{map} \leftarrow \textsc{CanPerturb}(\mathtt{s}, \mathtt{basin})$;
  return $(\mathrm{Q}, \mathrm{I}) \in \mathbb{C}_{\mathrm{O}} \Leftrightarrow (\mathtt{s}, \mathrm{I}, \mathtt{l}) \in \mathtt{map} \wedge (\mathtt{s}, \mathtt{l}) \equiv \mathrm{Q}$;
Fn TEMPORARY($\mathtt{s} \in \mathbb{V}, \mathtt{t} \in \mathbb{V}$)
  $\mathtt{basin} \leftarrow \textsc{Trap}(\textsc{Bwd}(\{\mathtt{t}\} \times \mathbb{I} \times \{0\}^n))$;
  $\mathtt{basin} \leftarrow \exists_{\mathbb{L}}(\mathtt{basin}) \times \mathbb{L}$;
  $\mathtt{basin} \leftarrow \textsc{Trap}(\textsc{Bwd}(\mathtt{basin}))$;
  $\mathtt{map} \leftarrow \textsc{CanPerturb}(\mathtt{s}, \mathtt{basin})$;
  return $(\mathrm{Q}, \mathrm{I}) \in \mathbb{C}_{\mathrm{T}} \Leftrightarrow (\mathtt{s}, \mathrm{I}, \mathtt{l}) \in \mathtt{map} \wedge (\mathtt{s}, \mathtt{l}) \equiv \mathrm{Q}$;

Once $\mathtt{basin}$ is computed, we cannot simply use CANPERTURB to identify possible perturbation candidates, as $basin$ only contains cases where the network is unperturbed. Instead, we use $\exists_{\mathbb{L}}(basin) \times \mathbb{L}$ to eliminate the $\{0\}^n$ component from the relation and substitute it for the full set $\mathbb{L}$. Intuitively, this enables all of the possible perturbations within the $basin$, allowing CANPERTURB to correctly select cases that are relevant for the given initial s state.

TEMPORARY Finally, TEMPORARY is conceptually a combination of both approaches: first, a $basin$ is computed with all perturbations disabled (i.e. $\{0\}^n$). At this point $basin$ represents the collection of possible states where lifting the perturbation guarantees that target t is eventually visited infinitely often. Afterwards, $\{0\}^n$ is exchanged for $\mathbb{L}$, which enables all possible perturbations in these $basin$ states. The combination of BWD and TRAP is then repeated, yielding a larger set of states that eventually always reach target t even if the corresponding perturbation is initially applied, but eventually lifted (i.e. applied only temporarily). Finally, CANPERTURB again limits this set to perturbations that are compatible with source s.

Note that under the assumption that t is a member of an attractor for all $I \in \mathbb{I}$, we have that the result of permanent control is always a subset of results for temporary control. Also note that for permanent control, if a combination of perturbed variables disturbs the attractor in which t resides, the TRAP procedure ensures that such case is eliminated from the $basin$ relation entirely. This is because in this situation, all states (including t) can reach states that are not backwards reachable from t and are thus eliminated.

Correctness of the procedures follows from the discussion above. All introduced procedures also surely terminate. The fixed-point procedures (Bwd and Trap) necessarily reach the fixed-point, because the superset $\mathbb{X}$ on which they operate is finite, and the resulting set can only grow (or shrink) in each iteration. The procedure CanPerturb is a straightforward for-loop over all BN variables. All control procedures are thus a composition of the described terminating operations or symbolic operations on BDDs.

4.3.5 *Evaluation*

To evaluate our methods, we have implemented algorithms from the previous section by using internal libraries of the tool AEON [Ben+20]. Our experiments were run using a computer with AMD Ryzen Threadripper 2990WX 32-Core Processor and 64GB of memory. We describe the software implementation and how to access in Chapter 6.

4.3.5.1 *Benchmark model set*

We use real-world biological BN models to evaluate our methods. The first model is a *myeloid* differentiation network [Kru+11] which models a bone marrow tissue cell differentiation from common myeloid cells to specialised blood cells (megakaryocytes, erythrocytes, granulocytes and monocytes). The second model depicts heart development done by *cardiac* kernel transcription factors [Waa+14]. The third model explains regulatory interactions, which may cause *ERBB* kinase over-expression – a marker of breast cancer [Sah+09]. The fourth model focuses on specific conditions which lead to a metastatic *tumour* [Coh+15]. The last model, the Mitogen-Activated Protein Kinase (*MAPK*) network, consists of signalling pathways involved in diverse cellular processes, such as cell cycle, survival, apoptosis and differentiation [Gri+13].

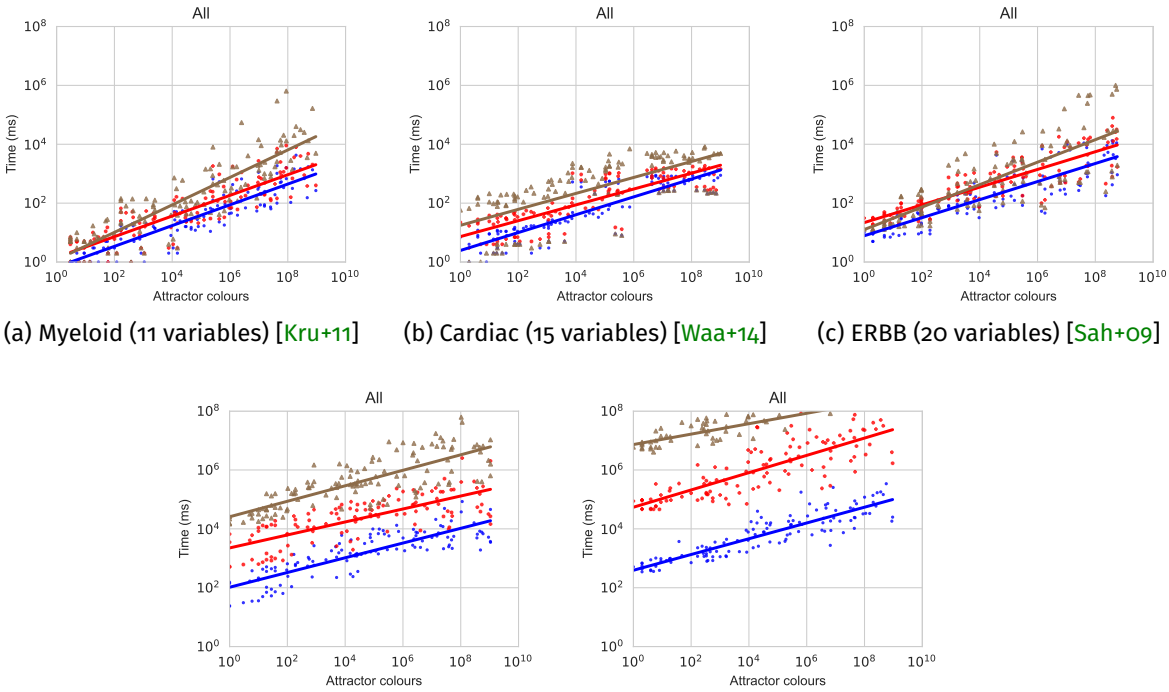
As our methods focus on the ability to handle systems with nontrivial unknown behaviour, we synthetically derive partially specified versions of the models listed above. We distinguish six model groups that differ in the number of interpretations (size of $|\mathbb{I}|$).

The numbers of interpretations corresponding to the individual groups are $[2^0, 2^5], [2^6, 2^{10}], \ldots, [2^{26}, 2^{30}]$. For every class, we then generate 20 samples of partially specified networks by randomly removing some fully-specified functions. The arity of removed functions is limited to 5 to prevent intractable parameter explosion. This methodology approximates a uniform distribution of samples across an exponential scale of benchmark model sizes while still covering models with varying numbers and arity of uninterpreted functions.

We then also randomly select the target and source attractor for every sampled model. However, we always verify that the actual number of model interpretations admitting the target attractor belongs

to the given range. That is, for a model in class $2^{26} - 2^{30}$, we guarantee that the target attractor is actually admissible for at least 2^{26} interpretations, even though the total number of all possible interpretations for the model can be higher. That is because we optimize control methods by restricting them to the interpretations where the attractor is present. As discussed in Section 4.1.2, one-step and temporary controls are unable to stabilize a system in a state which is not part of any attractor. Since we want the results to be consistent for all methods and phenotypic variation is not the focus of this work, we pose the same constraint to permanent control.

4.3.5.2 *Scalability*



(a) Myeloid (11 variables) [Kru+11] (b) Cardiac (15 variables) [Waa+14] (c) ERBB (20 variables) [Sah+09]

(d) Tumour (32 variables) [Coh+15] (e) MAPK (53 variables) [Gri+13]

Figure 16: **Performance of the control methods.** Plots in every row show results for a different real-world model. The horizontal axis of all plots is showing the number of attractor interpretations (given by model instances from our benchmark set) and the vertical axis is for the duration (in ms) of the given method computation. Plots in the first three columns show all measurements for one-step, permanent and temporary controls using the colored markers which are surrounded by polygons to better visualize their disposition. The computations that did not finish computation within a threshold of 24 hours are not displayed in the figure. The last column displays scalability trends of all method results by interpolating logistic regression slope with $\log y = \log x$ function.

Our first experiment evaluates the performance and scalability of our method with respect to the model size (count of variables) and unknown knowledge (number of interpretations). We use the bench-

mark model set as described above. We list all results in Figure 16. All plots are showing the relation of method computation time to the number of considered attractor interpretations. The rows depict results for particular real-world models. In every column, there are results for different control methods (one-step, permanent, and temporary). The last column displays results from all methods in a single plot. To capture the scalability trends we use $\log y = \log x$ type of logistic regression (due to exponential scaling of interpretations as well as time). All computations had 24-hour time-out. Nonetheless, this case never occurred for one-step control and only in a MAPK model for permanent control (5 times). The temporary control yields more such cases, but only in the most demanding scenarios in ERBB (2 times), tumour (1 time) and MAPK (70 times) models. Note that as we disregard these results for compact logistic regression visualisation, the logistic regression might be slightly steeper in the reality.

As described before, the symbolic representation is used to encode three types of entities: a set of the STG vertices, a set of interpretations (PSBN interpretations) which admit an edge in the STG, and a set of perturbations that permit the given edge. Therefore, when we increase the number of interpretations, only one dimension of the resulting symbolic representation grows. In opposition, when we increase the number of the system's variables, both the number of states and the number of available perturbations grow. Therefore, it is expected that increasing the model size has a bigger impact on the performance than increasing the number of model interpretations. This hypothesis is to some extent supported by our experiment captured in Figure 16. While the maximal increase of variables (myeloid vs. MAPK models) is just five-fold, we need to increase interpretations 10^7 times to obtain the similar one-step control runtime of the myeloid model as the best-performing MAPK version. This difference is even amplified for other types of control. For example, in the case of temporary control, no myeloid model version performs as bad as the best-performing MAPK.

Nonetheless, it must be highlighted that the impact of scaling up interpretations as well as variables is to some extent unpredictable. This is given by the intrinsic mechanics of the introduced STGs. As the nature of our method relies on fixed-point computation where each iteration eliminates only the direct predecessors of a particular set, in cases where e.g. a long path in the graph arises, our method will not be able to scale well: the algorithm will need many iterations to converge, each eliminating only a small fraction of the STG vertices. Another unforeseeable effect on the method's performance have the BDDs themselves. The ordering of the BDD diagram has a significant impact on the effectiveness of all operations and no single ordering strategy can perform best in all cases [HIB01]. Our observation is also supported by the experiment.

Notice, the high variance of run times for the similar size of attractor interpretations shown in the plots. For example, in the case of temporary control in the ERBB model, compared to versions with fewest interpretations, some highest attractor interpretations (over 10^8) cause just approximately 10-times longer execution time while other cases with a similar number of interpretations result in time-outing computations, therefore the slow-down of over 10^7. Or in the case of the myeloid model, even despite its somewhat smaller variable count than in the cardiac model, some computations take longer than in the bigger cardiac model, especially in the temporary control case.

Another insight that can be obtained from Figure 16 is how one-step, temporary and permanent controls differ in their complexity in terms of computational time. We can notice that one-step control is always over-performing other methods. This is because we do not need to consider perturbed state space when computing the trap set of the target attractor. In contrast, temporary control is usually by far the most computationally demanding compared to the other types of control. This should be given by the need to compute two trap sets. The first computed trap set is the same as in the case of one-step control and should be thus easy to compute.

However, the seed for the second computed trap space is a much bigger vertex set, and therefore the resulting second trap space (which incorporates perturbations) of the first trap space (without perturbations) can generate much more complex symbolic structure and also gives more space for slowly converging fixed-points which were discussed previously. Nonetheless, in some rare cases with models containing few interpretations (myeloid and ERBB), the temporary control might over-perform the permanent one. This can be caused by the first (unperturbed) strong basin computed by temporary control being very similar to the second (perturbed one) while the first computation might be somewhat faster as it does not consider perturbations. We do not observe such scenarios in bigger models (tumor and MAPK) where the trends of one-step over-performing permanent control, and permanent over-performing temporary one look quite consistent.

4.3.5.3 *Robustness*

Next, we conduct experiments evaluating the presented robustness metrics across different types of controls. In particular, we use maximal robustness ρ_{max} and union robustness $\rho_{\cup}$ metrics which we measure on sets of fixed perturbation size. The results are shown in Figure 17 and Figure 18.

First, let us look at what particular types of global robustness metrics can tell us about the inspected systems. Under the assumption that the real biological system corresponds to a single “correct” in-

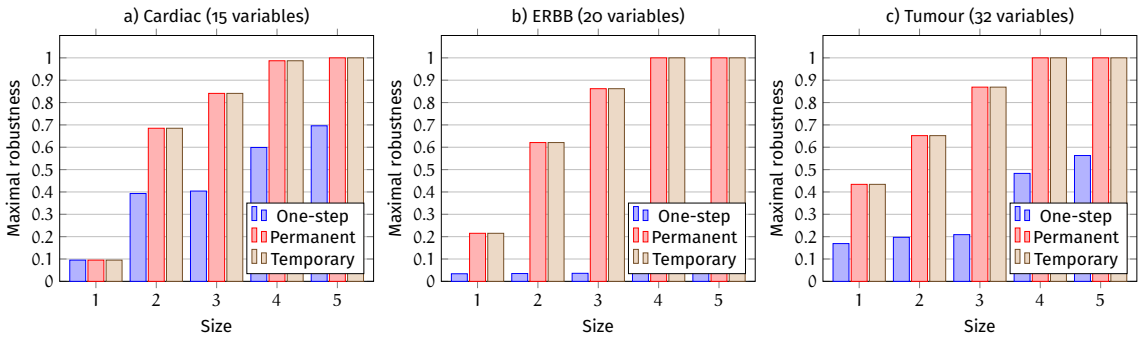


Figure 17: Maximal robustness ρ_{max} per set of perturbations with a given size – comparison between different types of controls. The horizontal axis shows the size of perturbations in the set and the vertical axis displays the maximal robustness. Each chart is showing robustness for a different model.

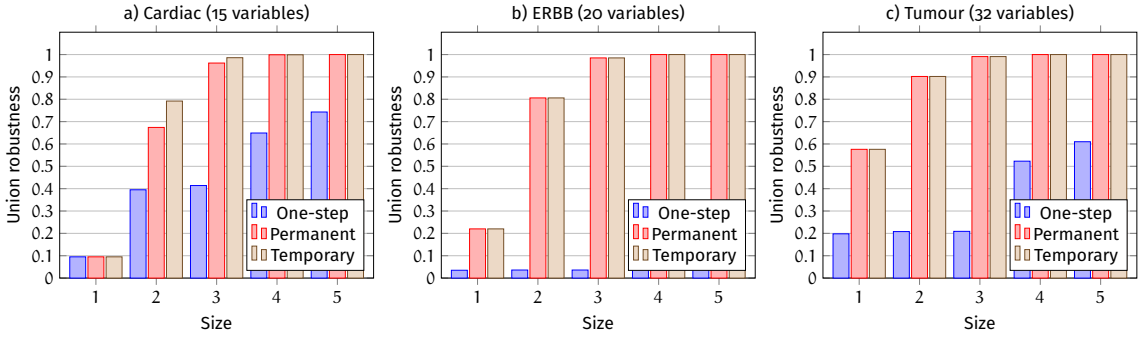


Figure 18: Union robustness $\rho_{\cup}$ per set of perturbations with a given size – a comparison between different types of controls. The horizontal axis shows the size of perturbations in the set and the vertical axis displays the union robustness. Each chart is showing robustness for a different model.

terpretations of our partially specified BN model, what the maximal robustness tells us is how high the chance that *a particular perturbation* will work in the real model. Alternatively, we use union robustness to evaluate a chance that there is *some perturbation* of the given size that will work. For example, in the ERBB model (Figure 17b and Figure 18b), we can notice that there is a 60% chance that the most robust perturbation (temporary or permanent) of size two will work for the real, fully specified model. However, even if this particular perturbation fails, we know that 80% of the interpretations can be influenced by a perturbation of this size. This means that out of the 40% of interpretations that cannot be influenced by the perturbation with the maximal robustness, half is still controllable by perturbing only two variables. For all three models, it is thus highly likely that a temporary perturbation of size no bigger than 3 exists which controls the network. That is a very small portion of the total model variables which could be possibly perturbed.

As the one-step control is subsumed by temporary control, it is expected that the temporary perturbations achieve the same or higher

robustness than the one-step perturbations. This is also shown in the experiments, where we find no instance in which the one-step control could achieve higher maximal or union robustness. We can also observe that temporary and permanent controls mostly perform very similarly.

Specifically, they achieve 100% maximal robustness for perturbations of size three (in the case of the cardiac model) or four (in the case of other models). Therefore, we successfully computed perturbations which surely work even despite the missing knowledge we introduced into the models for the purpose of our experiments. This is also in line with previous research demonstrated by, e.g., [SPP19b; SP20b], where the necessary size of temporary or permanent control is often very small.

The temporary control outperforms the permanent in the union robustness only in the cardiac model (Figure 18a). This means there are some perturbations which work when applied temporarily as opposed to when applied permanently. The possible reasons for this were already discussed in the previous sections. There are two plausible explanations for this observation.

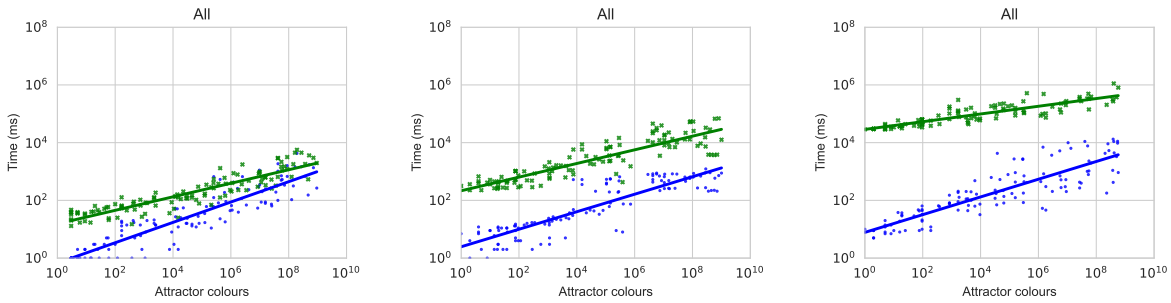
The first may be a simple fact that the trap set computed for temporary control is expected to be bigger since the seed of the trap set is not only the given target state but also its unperturbed trap set. The second explanation for this is a scenario when a permanent perturbation disrupts the BN dynamics in such a way that the target attractor becomes unreachable or completely ceases to exist. Our observation is also in line with [SPP19a], where the minimal temporary control also outperformed the permanent one in a few cases.

4.3.5.4 *Comparison of source-target methods*

We also compare the symbolic and the semi-symbolic approach described in the previous Section 4.2 (from [Bri+21]) one-step controls. The results are captured in Figure 19. We use the same benchmark model set as in the previous experiment. We also use a similar layout for plots in the figure.

However, there are fewer models considered as the semi-symbolic method was not able to compute any instance of tumor or MAPK models due to out-of-memory (OOM) errors. Also, some instances of ERBB model were not processed due to OOM rather than time-outs. The first column shows the results of the fully symbolic one-step control method while the second column shows the results of the previous semi-symbolic method. The plots in the last column compactly compare these two methods using the same logistic regression as in the previous experiment.

Another insight of our experiments is a comparison of the novel symbolic approach to the semi-symbolic approach from [Bri+21]. The experiment captured in Figure 19 shows that the symbolic approach



(a) Myeloid (11 variables) [Kru+11] (b) Cardiac (15 variables) [Waa+14] (c) ERBB (20 variables) [Sah+09]

Figure 19: **Comparison with existing semi-symbolic method from [Bri+21].** Plots in every row show results for a different real-world model. The horizontal axis of all plots shows the number of attractor interpretations (given by model instances from our benchmark set) and the vertical axis is for the duration (in ms) of the given method computation. Both axes have logarithmic scaling. Plots in the first column show all measurements for the fully symbolic one-step method from this work. The second plot is showing results for the one-step control method from [Bri+21] using the colored markers. Plots display scalability trends of all methods by interpolating the results with the logistic regression slope of $\log y = \log x$ function.

is performing better in all measured cases. As can be seen from the regression trends, the bigger the model, the bigger is also the margin by which the novel method over-performs the semi-symbolic one. However, in the cases with high number of interpretations, the parallelization of the semi-symbolic method might be faster than the fully symbolic one. This is also suggested by the linear regression trends where we can notice, that the semi-symbolic method has a less steep slope. Nonetheless, the greatest benefit of our novel method is the ability to process models with significantly bigger state space such as the tumor model or MAPK, as well as the highly unknown variants of the ERBB model, where the explicit state-space considered by the semi-symbolic approach completely fails to fit into the available memory.

4.3.6 *Summary*

In this section we presented fully symbolic methods for source-target control problem for PSBNs. We have described the algorithms for computing one-step, temporary and permanent control relations. Finally, we have evaluated our methods on a benchmark set of real-world biological models with varying sizes and degrees of uncertainty. We have compared the performance of different types of control and also compared our one-step control method to the semi-symbolic approach from Section 4.2. We have shown, that new symbolic methods are superior in both performance and scalability, as well as in the size of perturbations enabled by temporary and permanent control.

5

PHENOTYPE CONTROL OF PARTIALLY SPECIFIED BOOLEAN NETWORKS

The previous chapter was focused on the problem of source-target control for PSBNs. However, in many applications, the goal is not to reach a specific state but rather to achieve a certain *phenotype* – a set of states with shared properties. For example, in the context of gene regulatory networks, a phenotype may correspond to a particular cell type or cellular behavior.

Moreover, in the context of PSBNs, the target of the control is prone to change across different interpretations of the network. In the previous chapters, we tackled this problem by narrowing down the interpretation space to those where the target state was a part of some attractor. Nonetheless, this approach might discard interpretations where the network stabilizes in a *similar* attractor although not exactly the same as the target state. That is why phenotype control is a more suitable framework for PSBNs: it allows us to consider a broader set of interpretations where the network stabilizes and describe the desired behavior in a more flexible way rather than focusing on a single state.

In this chapter, we first formally define the problem of phenotype control for PSBNs and its variants. Then we present an algorithm with symbolic state space exploration to solve the posed problem. Finally, we demonstrate the applicability of our approach to middle-sized PSBNs.

5.1 PROBLEM DEFINITION

Before defining the problem of the phenotype control, we need to think about the relationship between attractors and phenotypes. In the context of Boolean networks, an attractor represents a stable state or a set of states that the system can converge to. These attractors are often associated with specific biological phenotypes, which are observable characteristics or behaviors of the system.

It has been shown that attractors are closely tied to the notion of biological phenotypes [Kla+18; CFTS22]. Each attractor represents a possible stable *outcome* achievable within a particular $\mathcal{N}$. The combination of traits observable as part of this outcome then forms the actual phenotype. However, in many cases, there can be multiple attractors that exhibit the same set of traits, and thus the same biological phenotype. We formalize this concept as follows:

Definition 19. *A* character *is a set of BN variables* $\mathcal{U} \subseteq [1, n]$ *which cover the observable real-world properties of the system. A* trait T *is then a valuation of these character variables:* $T : \mathcal{U} \to \mathbb{B}$.

Such character variables $\mathcal{U}$ typically correspond to the network outputs, but this is not required. Each trait defines a subspace $\mathcal{T} \in \mathbb{B}_*^n$ s.t. $\mathcal{T}_i \neq *$ for $i \in \mathcal{U}$, and $\mathcal{T}_i = *$ otherwise.

Definition 20. *A* phenotype *is a set of states* $\phi \subseteq \mathbb{B}^n$ *described by an arbitrary combination of the BN traits. An attractor* A *may have three types of relationships toward a given phenotype* ϕ*:*

- A stabilizes *in* ϕ *iff* $A \subseteq \phi$ *(or equivalently* $A \cap \phi = A$*).*
- A avoids ϕ *iff* $A \setminus \phi = A$ *(or equivalently* $A \cap \phi = \emptyset$*).*
- A oscillates *through* ϕ *otherwise (equivalently* $\emptyset \neq (A \cap \phi) \neq A$*)*

If all *attractors of a BN have the same relationship with a phenotype, we can generalize the relationship to a whole BN.*

Note that while attractor is an *intrinsic* property of a BN, a phenotype is a modeler's choice. The same BN can be studied with respect to different phenotypes, depending on the research question.

While a *trait* is always a subspace, a phenotype is an arbitrary combination of traits. For example, assuming $n = 4$ and $\mathcal{U} = \{1, 2\}$, $11{*}{*}$ and $00{*}{*}$ are two of the four admissible traits (or rather trait subspaces). Each of these traits can represent a single phenotype, but they can also represent a combined phenotype $00{*}{*} \cup 11{*}{*}$. We typically assume that if a network admits more than one phenotype, these are mutually disjoint. Finally, note that not all phenotypes can be manifested by a BN (e.g. if a trait is not biologically viable).

Table 5: **Phenotype related terms.** The table summarizes the notions of character, trait, and phenotype in the context of real-world concepts compared to Boolean network models. The considered example is a small excerpt of MAPK pathway.

Concept	Real-world	Example	Network	Example
Character	Property of an interest	A phosporolation on FRS2 tyrosine residue	A set of variables $\mathcal{U}$	{FRS2}
Trait	Observation	FRS2 tyrosine residue is not phosphorylated	Valuation $T : \mathcal{U} \to \mathbb{B}$, a subspace generator	FRS2=0, *0*
Phenotype	Observation combination	FRS2 or ERK is phosphorylated	A combination of traits ϕ	*1* ∪ **1

Now that we have defined the relationship between attractors and phenotypes, we can define the problem of permanent phenotype control for Boolean networks and PSBNs.

We say that a permanent variable perturbation $Q \in \mathbb{B}^n_*$ *controls* Boolean network $\mathcal{N}$ towards stabilization of phenotype ϕ if and only if $\forall A \in \mathbb{A}(\mathcal{N}_Q).A \subseteq \phi$ (all attractors of a BN stabilize in ϕ). Similarly, we can also define control towards a different relationship of the network with the phenotype (avoidance, oscillation). What is also interesting, is that $\mathbb{A}(\mathcal{N}_Q)$. is not necessarily a subset of $\mathbb{A}(\mathcal{N}_Q)$. A permanent perturbation can change existing attractors or even introduce new ones (e.g., by elimination of outbound transitions from some states).

This concept naturally applies to interpretations of PSBNs as well: given a $\mathcal{M}$, an interpretation I and a perturbation Q, the interpretation has a perturbed $\text{STG}_Q(\mathcal{M}(I))$. As such, a perturbation Q controls $\mathcal{M}$ under the interpretation I if it ensures $\mathbb{A}(\mathcal{M}(I)_Q) \subseteq \phi$. Formally:

Definition 21. *Assume a partially specified network $\mathcal{M}$, a phenotype $\phi \subseteq \mathbb{B}^n$, a desired relationship between the network and phenotype $R \in \{$*stabilization, oscillation, avoidance*$\}$, and a set of admissible perturbations $\mathbb{Q} \subseteq \mathbb{B}^n_*$. The goal of the* complete *phenotype control is to compute a relation $\mathbb{C}_\phi$ of all pairs $(I, Q) \in \mathbb{I}(\mathcal{M}) \times \mathbb{Q}$ such that Q controls $\mathcal{M}(I)$ towards the desired phenotype relationship R with ϕ.*

Same as for the source-target control, the concept of size (number of perturbed variables) and robustness (ratio of successfully controlled interpretations) applies to perturbations as well. A slight difference is, that since we are considering only permanent perturbations and the objective is more general, we do not restrict the interpretation space to those where the target state is an attractor. Instead, we consider all interpretations and check whether the attractors of the perturbed system have the desired relationship with the phenotype:

$$\rho(Q) = \frac{|\{I \in \mathbb{I}(\mathcal{M}) \mid Q \text{ controls } \mathcal{M}(I)\}|}{|\mathbb{I}(\mathcal{M})|}$$

The concepts of the maximal and union robustness would be adjusted in the same way.

5.2 METHODS

5.2.1 *Symbolic computation model*

For the phenotype control problem, we rely on the same symbolic computation model as for the source-target control from Section 4.3. Again, the state-space, interpretation-space and perturbation-space are encoded as BDDs, composed of three components:

$$\mathbb{X} = \mathbb{V} \times \mathbb{I} \times \mathbb{L}$$

Where $\mathbb{V}$ encodes the state-space $\mathbb{B}^n$, $\mathbb{I}$ encodes the interpretation-space $\mathbb{B}^m$ of a normalized PSBN and $\mathbb{L}$ of size $(\mathbb{B}^n)$ encodes the perturbation-space without an actual value of a perturbation (i.e., which variables are perturbed, but not how they are perturbed).

As opposed to source-target control however, we do not know the source state beforehand, and we want to control the system globally regardless of the source state. As we will see later, that renders the problem more complex and requires explicit manipulation with control map for each query of working perturbations. This makes our method not fully symbolic and requires some enumeration of perturbations. However, we still rely on symbolic state space exploration to compute the control map for a large set of perturbations at once, which significantly speeds up the process compared to naive enumeration. Moreover, the typical size of admissible perturbations is small (e.g. up to 3 perturbed variables), which further mitigates the need for enumeration. We also use this fact to optimize the size of the BDD encoding of the perturbation space $\mathbb{L}$ — since we are usually interested in perturbations of size up to k, we can restrict the set of admissible perturbations.

SYMBOLIC OPERATIONS Recall from the Section 4.3.2, that set operations ($\cap, \cup, \setminus$, etc.) on symbolic sets (or relations) are implemented through Boolean logical operators ($\wedge, \vee, \neg$, etc.) as is customary for BDDs. By the same token, we also use standard methods of reachability, namely functions PRE, POST, BWD and TRAP as defined in Section 4.3.3. Furthermore, the following standard BDD operations are used:

$$\begin{aligned}
\textsc{Project}_{\mathcal{V}}(X \subseteq \mathbb{B}^k) &= \{x \in \mathbb{B}^k \mid \exists b \in \mathbb{B}.\, x[\mathcal{V} \mapsto b] \in X\} \\
\textsc{Select}_{\mathcal{V}=b}(X \subseteq \mathbb{B}^k) &= \{x \in \mathbb{B}^k \mid x \in X \wedge x[\mathcal{V}] = b\} \\
\textsc{Restrict}_{\mathcal{V}=b}(X \subseteq \mathbb{B}^k) &= \textsc{Project}_{\mathcal{V}}(\textsc{Select}_{\mathcal{V}=b}(X))
\end{aligned}$$

Here, $\mathcal{V}$ denotes a variable of the BDD encoding (e.g. $\mathcal{V}_3$). We can also use a list of conditions in the subscript of the method as a shorthand for multiple nested calls to the same method. Intuitively, PROJECT is equivalent to existential quantification, SELECT is implemented using conjunction, and RESTRICT is a combination of both.

Finally, we use $\mathbb{VAL}^n_k$ to denote a BDD which encodes a function of n inputs that is *true* iff exactly k inputs are *true*. This construct is useful for restricting the set of admissible perturbations. To construct such BDD efficiently, we observe the following:

$$\begin{aligned}
\mathbb{VAL}^n_0 &= \bigwedge_{i \in [1,n]} \neg x_i \\
\mathbb{VAL}^n_k &= \bigvee_{i \in [1,n]} (\mathbb{VAL}^n_{k-1} \vee x_i)
\end{aligned}$$

5.2.2 *Control algorithm*

Our control approach is based on Algorithm 5 where we describe:

- CompletePhenotypeControl: Core algorithm that iteratively computes the *complete* control map for an admissible set $\mathfrak{Q}$.
- RobustPhenotypeControl: A wrapper for the core algorithm that facilitates minimal control under the desired *robustness*.
- Enumerate: An auxiliary method to enumerate all *working* perturbations and find the maximum robustness.

Algorithm 5: Permanent phenotype control

Fn CompletePhenotypeControl($\phi \subseteq \mathbb{V}, Q \subseteq \mathbb{V} \times \mathbb{L}$ (encodes $\mathbb{B}^n_*$))
  universe $\leftarrow \{ (v, i, l) \in \mathbb{X} \mid (v, l) \in Q \}$;
  phenotype $\leftarrow$ universe $\cap (\phi \times \mathbb{I} \times \mathbb{L})$;
  phenotype_trap $\leftarrow$ Trap(phenotype);
  non_phenotype $\leftarrow$ universe $\setminus$ phenotype_trap;
  cannot_ctrl $\leftarrow$ Trap(non_phenotype);
  for $i \in [1, n]$ **do**
    not_pert $\leftarrow$ Project$_{\mathbb{V}_i}$(Select$_{\mathbb{L}_i=0}$(cannot_ctrl));
    cannot_ctrl $\leftarrow$ not_pert $\cup$ Select$_{\mathbb{L}_i=1}$(cannot_ctrl);
  control_map $\leftarrow$ universe $\setminus$ cannot_ctrl;
  return control_map;
Fn RobustPhenotypeControl($r, \phi \subseteq \mathbb{V}, Q \subseteq \mathbb{V} \times \mathbb{L}$)
  for $k \in [0, n]$ **do**
    $Q_k \leftarrow Q \cap (\mathbb{V} \times \mathbb{VAL}^n_k)$;
    control_map $\leftarrow$ CompletePhenotypeControl(ϕ, Q_k);
    $\rho_{best} \leftarrow$ Enumerate(1, control_map);
    if $\rho_{best} \geqslant r$ **then return**;
Fn Enumerate($i \in \mathbb{N}$, control_map $\subseteq \mathbb{X}$ (encodes $\mathbb{B}^m \times \mathbb{B}^n_*$))
  if control_map $= \emptyset$ **then return** 0;
  if $i > n$ **then return** $\frac{|\{I \in \mathbb{I} \mid \exists (v,I,l) \in \mathbb{X}.\, (v,I,l) \in \text{control\_map}\}|}{|\mathbb{I}|}$;
  not_controlled $\leftarrow$ Restrict$_{\mathbb{L}_i=0}$(control_map);
  controlled_true $\leftarrow$ Restrict$_{\mathbb{L}_i=1,\mathbb{V}_i=1}$(control_map);
  controlled_false $\leftarrow$ Restrict$_{\mathbb{L}_i=1,\mathbb{V}_i=0}$(control_map);
  best $\leftarrow$ Enumerate($i+1$, not_controlled);
  best $\leftarrow$ *max*(best, Enumerate($i+1$, controlled_true));
  best $\leftarrow$ *max*(best, Enumerate($i+1$, controlled_false));
  return best;

COMPLETE CONTROL The main idea behind Algorithm 5 is the following: A perturbation achieves phenotype control if and only if all attractors in the perturbed STG are within the phenotype subset ϕ. Consequently, we want to *discard* any perturbation that provably retains some attractor not contained in ϕ.

Therefore, we compute a non-phenotype trap set that is guaranteed to contain all non-phenotype attractor states in it. First, we compute a similar trap subset of the phenotype ϕ which is guaranteed to contain all phenotype attractors. Then, we invert this set and repeat the operation to obtain a trap which is a superset of all non-phenotype attractors. Note that simply computing the largest trap set within $\mathbb{V} \setminus \phi$ would only cover attractors that are completely within $\mathbb{V} \setminus \phi$. The above-described process is necessary to also cover attractors that intersect ϕ but are not subsets of ϕ.

The algorithm then iterates over all network variables and performs projection in cases where the variable is *not perturbed*. Initially, the `cannot_ctrl` set contains at least one state of the perturbed STG of each $Q \in \mathbb{Q}$ that admits a non-phenotype attractor. After this operation, `cannot_ctrl` contains *all* states of such perturbed STG. In other words: the resulting set can depend on variable $\mathcal{V}_i$ only for $\{l \in \mathbb{L} \mid l_i = 1\}$ (i.e. the variable is perturbed). In such cases, the role of $\mathcal{V}_i$ is to encode the actual perturbed value of the i-th network variable. Finally, we invert the `cannot_ctrl` set to only retain perturbations where no non-phenotype attractor state exists.

ROBUST CONTROL To extend this algorithm to robust control, we test the perturbations of increasing size (using the $\mathbb{VAL}^n_k$ BDD) and use a recursive ENUMERATE method to find perturbations with maximal robustness. Notice that the necessity of this step makes our algorithm *semi-symbolic*.

Instead of iterating through all possible control strategies of size k, procedure ENUMERATE recursively branches into three cases for each variable i: i is not perturbed, i is perturbed to *true*, and i is perturbed to *false*. Note that each call completely eliminates both $\mathbb{L}_i$ and $\mathbb{V}_i$ from `control_map` (for the case of $l_i = 0$, $\mathbb{V}_i$ was already eliminated by the core algorithm). As such, once $i > n$, the resulting `control_map` only depends on BDD components of $\mathbb{I}$ and we can use it to compute the robustness.

Note that for simplicity, we do not store the actual perturbations with maximal robustness explicitly in the algorithm. However, these can be easily reconstructed from the recursion path in the ENUMERATE algorithm. Furthermore, note that the recursion in the ENUMERATE algorithm can be replaced using *projected iteration*, where we first project the `control_map` to the admissible $l \in \mathbb{L}$, and then project only to the state variables perturbed within each such l. This approach can be faster as it uses fewer symbolic steps. However, it is

also highly specific to each BDD library. As such, we chose to present the more widely applicable algorithm.

5.2.3 *Phenotype control with oscillations*

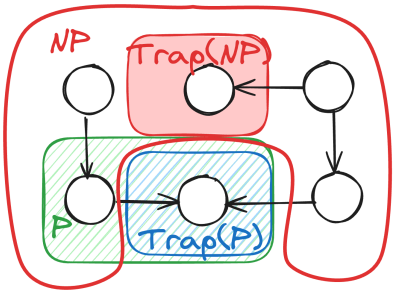
Method for phenotype control as shown above works only for the case of stabilization (or avoidance) of a phenotype. In the case of oscillation, we need to ensure that the system can both enter and exit the phenotype. This means that we need to check both whether the system can be controlled to stabilize in the phenotype and whether it can be controlled to stabilize in its complement.

Since we already showed the niches of the state space exploration and control map enumeration, let us zoom in to the support of oscillatory phenotypes. In Algorithm 6 we show a simplified version of the control algorithm for oscillatory phenotypes. The function returns true or false depending on whether the perturbed Boolean network is guaranteed to comply with the desired relationship with the phenotype. This algorithm is then a part of the complete symbolic control algorithm, where state space, interpretations and perturbations are explored simultaneously. Also, the final result would be the same type of a control map as in the previous algorithm, and further enumeration of working perturbations would be required to find the optimal one.

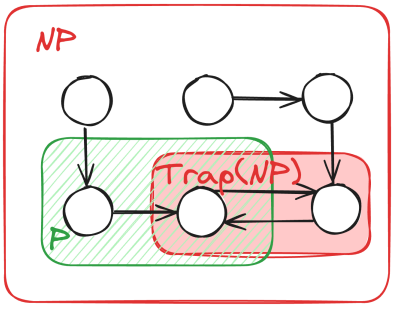
Algorithm 6: Phenotype Control Algorithms

```
1  Fn IsControlled(ϕ ⊆ 𝕍, osc ∈ 𝔹)
2      phenotype_space ← if osc then Bwd(ϕ) else ϕ;
3      phenotype_trap ← Trap(phenotype_space);
4      non_phenotype_trap ← Trap(𝔹^n \ phenotype_trap);
5      return non_phenotype_trap = ∅;
6  Fn IsPhenotypeControl(ϕ ⊆ 𝕍, type ∈ {S, A, 0})
7      if type = S then
8          return IsControlled(ϕ, false);
9      else if type = A then
10         return IsControlled(𝕍 \ ϕ′, false);
11     else if type = 0 then
12         in_phen ← IsControlled(ϕ, true);
13         out_phen ← IsControlled(𝕍 \ ϕ, true);
14         return in_phen ∧ out_phen;
```

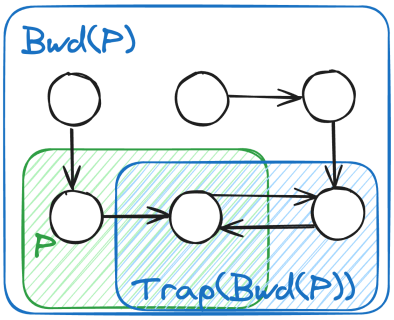
Even though there are three types of relationships between a network and a phenotype, the core of our algorithm IsPhenotypeControlAux relies on two different relationships – we see the oscillation as either allowed or forbidden.



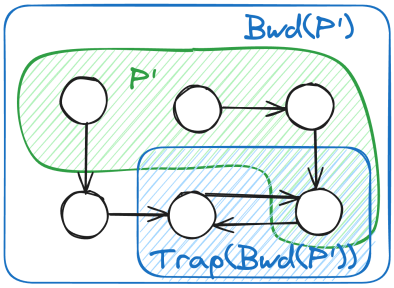
(a) BN not stabilizing in phenotype ϕ due to the non-phenotype attractor.



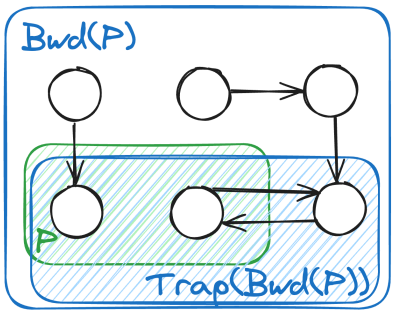
(b) BN not stabilizing in phenotype ϕ due to the oscillating phenotype attractor.



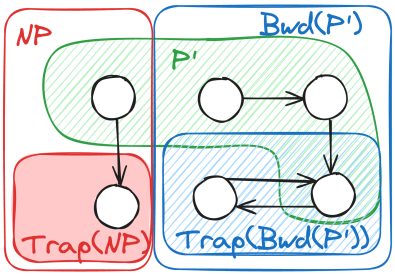
(c) BN exhibiting oscillating phenotype ϕ – computation of ϕ.



(d) BN exhibiting oscillating phenotype ϕ – computation of $\phi' = \mathbb{V} \setminus \phi$.



(e) BN not exhibiting oscillating phenotype ϕ – computation of ϕ.



(f) BN not exhibiting oscillating phenotype ϕ – computation of $\phi' = \mathbb{V} \setminus \phi$.

Figure 20: **Examples of Algorithm 6 progressions.** Figure (a) and Figure (b) show situations where BN does not stabilize in the given phenotype considering not allowed oscillation. Figure (c) and Figure (d) show the computation of the oscillating phenotype ϕ and its complement ϕ' for the BN exhibiting the oscillating phenotype. Figure (e) and Figure (f) show the same computation for the BN not exhibiting the phenotype.

In the case of a non-oscillatory phenotype, we first find the maximal trap set in the phenotype space itself. Thus, we obtain a set of states that contain all attractors stabilizing in the phenotype. We do not need to compute the specific attractors, since it is not necessary

for the phenotype control and such a specific attractor refinement might be costly for big networks. After that, we verify, whether remaining reachable space contains any trap set (which would surely contain at least one attractor). If the obtained trap set is empty, that means there are no other attractors than the ones which stabilize in the phenotype, and thus we can say, that the whole network stabilizes the phenotype. The example of an opposite situation can be seen in Figure 20a. Also, as it can be seen from Figure 20b, the attractors only oscillating trough ϕ will be also detected as trap sets negating the phenotype, since they will not be included in $\textsc{Trap}(\phi)$.

In order to allow also the oscillatory attractors, we extend the phenotype space with the states that are backward-reachable from the phenotype space. This way, also attractors that only oscillate through the phenotype are captured by the trap set as well. Notice, that set $\textsc{Trap}(\textsc{Bwd}(\phi))$ cannot contain any attractors completely outside of ϕ (in $\textsc{Bwd}(\phi) \setminus \phi$). This is because attractor states, by definition, cannot include any states that are unreachable from themselves. This would contradict the method by which we obtained the set (the backward reachability from the phenotype itself). Therefore, by expanding the considered states for trap set computation, we obtain both - attractors that stabilize in the phenotype and attractors that oscillate through it.

The function IsPhenotypeControlAux can be used as a stand-alone function when either just phenotype-stabilizing or both phenotype-stabilizing and oscillating networks are desired. Also, the phenotype avoidance algorithm is rather trivial, since it is sufficient to reverse the phenotype space as $\phi' = \mathbb{V} \setminus \phi$ and compute the control without allowing oscillation. However, the method IsPhenotypeControlAux on its own cannot guarantee that all attractors in the network are indeed oscillatory towards ϕ.

That is why we also introduce a wrapper function IsPhenotypeControl which exactly supports the defined network-phenotype relationships. The algorithm for phenotype stabilization and avoidance is trivial. For oscillatory phenotype also quite a straightforward approach is used – we verify whether the network doesn't avoid both ϕ and $\phi' = \mathbb{V} \setminus \phi$ when the oscillation is allowed. These two steps of computation on a network that oscillates through a phenotype are illustrated in Figure 20c and Figure 20d. On the other hand, Figure 20e and Figure 20f show a scenario, where the network is not oscillating because of the presence of an attractor stabilizing in ϕ. Notice, that such cases are not possible to detect with a single run of IsPhenotypeControlAux.

Table 6: Comprehensive overview of the tested real-world Boolean networks. The first four columns contain the model name and the counts of network inputs, all variables, and perturbable variables, respectively. Column $\mathcal{Q}_{\leqslant 3}$ represents the number of all admissible perturbations of size up to three. Sixth column gives a reference to the literature that describes each model. Lastly, the seventh column contains variables that we do not allow to be perturbed.

Model	**Ins**	**Vars**	**Per.**	$\mathcal{Q}_{\leqslant 3}$	**Ref.**	**Uncontrollable vars**
Cardiac	2	13	11	6,252	[Her+12]	`Tbx1, Tbx5`
Red. MAPK	4	14	11	25,008	[Gri+13]	`Apoptosis, Growth_Arrest, Proliferation`
ERBB	1	19	18	14,354	[Ito+12]	`pRB1`
Tumour	2	30	24	69,380	[Coh+15]	`Apoptosis, Metastasis, Invasion, Migration, EMT, CellCycleArrest`
Cell Fate	2	31	26	88,612	[Cal+10]	`Apoptosis, Survival, Death, Division, NonACD`
Full MAPK	2	49	46	2,010,768	[Gri+13]	`Apoptosis, Growth_Arrest, Proliferation`

5.3 EVALUATION

In this section, we evaluate our proposed phenotype method. As previously, all experiments were performed using a computer with AMD Ryzen Threadripper 2990WX 32-Core Processor and 64GB of memory.

5.3.1 *Performance*

BENCHMARK MODEL SET We use real-world Boolean networks to evaluate our method. All tested networks contain input nodes which can be viewed as functionally equivalent to zero-arity uninterpreted functions in PSBNs. We thus treat these inputs as unknown external signal beyond our control.

Table 6 lists all models and their relevant characteristics, including number of inputs, variables, perturbable variables, and reference to the original publication. We also state the number of admissible perturbations of size up to three. Previous work shows that such relatively small size is both realistic to implement in practice [BD21; CFTS22; SPP19a] and robust enough in the presence of partially unknown dynamics [Bri+23]. This number therefore represents how many models would need to be explored in a brute-force based methods for the models of the given size. The table also lists variables which we explicitly do not allow to be perturbed. These variables are mostly outputs and they represent traits which induce individual phenotypes. Therefore, perturbing these variables would lead to trivial control which is neither viable nor interesting.

Table 7: Performance of PSBN control. The first two columns contain model and phenotype names. Then, for perturbations of the size up to three, we list the computation time and maximal robustness found in perturbations of this size. Last column gives the number of minimal perturbations with $\rho = 1.0$.

Model	Phenotype	Size 1		Size 2		Size 3		# Min. per.
		Time	ρ	Time	ρ	Time	ρ	($\rho = 1.0$)
Cardiac	FHF	<1s	0.5	<1s	1.0	<1s	1.0	4
	SHF	<1s	0.5	<1s	1.0	<1s	1.0	2
	No mesoderm	<1s	0.5	<1s	1.0	<1s	1.0	3
Red. MAPK	Apoptosis	<1s	1.0	<1s	1.0	<1s	1.0	1
	Growth arrest	<1s	0.75	<1s	1.0	<1s	1.0	1
	No decision	<1s	1.0	<1s	1.0	<1s	1.0	1
	Proliferation	<1s	0.25	<1s	1.0	<1s	1.0	3
ERBB	Phosphor.	<1s	1.0	<1s	1.0	<1s	1.0	7
	Non-phospor.	<1s	1.0	<1s	1.0	<1s	1.0	8
Tumour	Apoptosis	2s	1.0	8s	1.0	23s	1.0	2
	EMT	<1s	0.5	3s	1.0	11s	1.0	15
	Hybrid	<1s	0.25	5s	1.0	24s	1.0	3
	Metastasis	<1s	0.5	<1s	1.0	1s	1.0	6
Cell Fate	Apoptosis	<1s	0	4s	1.0	58s	1.0	24
	Naive	<1s	0	2s	1.0	29s	1.0	8
	Necrosis	<1s	1.0	<1s	1.0	11s	1.0	1
	Survival	<1s	1.0	2s	1.0	21s	1.0	1
Full MAPK	Apoptosis	<1s	1.0	7s	1.0	14min	1.0	6
	Growth arrest	4s	0.75	4s	1.0	22min	1.0	47
	No decision	3s	0.81	107s	1.0	22min	1.0	45
	Proliferation	3s	0.25	3s	1.0	20min	1.0	8

The first model illustrates cardiac progenitor cells differentiation into the first heart field (FHF) or second heart field (SHF) [Her+12]. The second and sixth models represent Mitogen-Activated Protein Kinase (MAPK) network representing signalling pathways involved in diverse cellular processes including cancer deregulation. We use both the full and reduced versions of this model as stated in [Gri+13]. The third model depicts the key event preceding breast cancer cells proliferation which is the hyper-phosphorylation and subsequent lack of pRB. This process is regulated by ERBB kinase, the lack of which is considered a breast cancer marker [Sah+09]. The fourth model focuses on specific conditions which lead to a metastatic tumour [Coh+15]. Finally, the cell fate model provides a high-level understanding of the interplays between pro-survival, necrosis, and apoptosis pathways in response to death receptor-mediated signals [Cal+10].

PERFORMANCE EVALUATION ON REAL-WORLD MODELS In Table 7, we show results of computing phenotype control on all our benchmark models from Table 6. We use real-world phenotypes as described in the source literature. These are typically subspaces obtained by fixing network outputs to the desired values. The details of exact phenotypes can be found in Appendix B

For each model and its phenotype, we computed all working perturbations of size up to three. We then show the times needed for individual computations (including enumeration and robustness calculation), the highest robustness achieved for each case, and the number of minimal perturbations with 100% robustness.

Table 8: Minimal perturbations for reduced MAPK model. The first column is target phenotype while other columns contain minimal perturbations for unperturbed model, model with over-expressed EGFR, and model with FGFR3 gain-of-function. Variables divided by / stand for perturbing either of them. ∅ is used when no perturbation is necessary.

Phenot.	**Unperturbed**	**Perturbed** EGFR=1	**Perturbed** FGFR3=1
Apoptosis	DNA_dmg=1, TGFBR_st=1, FRS2=1	DNA_dmg=1, TGFBR_st=1, ERK=0, p53=1	DNA_dmg=1, TGFBR_st=1, ERK=0, p53=1
¬Apoptosis	∅	AKT=1, ERK=1, MSK=0, PTEN=0, p14=0, p53=0	AKT=1, ERK=1, MSK=0, PTEN=0, p14=0, p53=0
Prolif.	ERK=1	p14=0, p53=0	{p14/p53=0, FRS2=1}, {p14/p53=0, PI3K=1}, {p14/p53=0, EGFR=1}
¬Prolif.	∅	DNA_dmg=1, TGFBR_st=1, AKT=0, ERK=0, MSK=0, PI3K=0, PTEN=1, p53=1	DNA_dmg=1, TGFBR_st=1, AKT=0, ERK=0, MSK=0, PI3K=0, PTEN=1, p53=1
No decis.	∅	MSK=0	MSK=0
¬No decis.	DNA_dmg=1, TGFBR_st=1 EGFR=1, ERK=1, FRS2=1, p53=1	∅	DNA_dmg=1, TGFBR_st=1 ERK=1, FRS2=1, p53=1, PI3K=1

5.3.2 *Method validation*

Now we demonstrate how phenotype control can be used to replicate observations conducted in [Gri+13] and even strengthen these results with a more robust assessment of the model. In [Gri+13], various perturbations of a reduced MAPK model are simulated and their effect on phenotypes is observed. Specifically, networks with all inputs set to 0 and with EGFR or FGFR3 over-expressed (gain-of-function) are exposed to a set of further perturbations. The model has three outputs (Apoptosis, Growth_Arrest, and Proliferation) and three attractors are observed: apoptosis (Apoptosis=Growth_Arrest= 1), proliferation (Proliferation= 1) and no decision (all outputs set to false).

We first compute phenotype control on the fully specified but reduced model. We list discovered minimal perturbations for network

variants of interest in Table 8. Here, the perturbations which were also discovered in [Gri+13] are shown as green. In the enumeration, where appropriate, we only considered perturbations that contain over-expressed EGFR or FGFR3, as in the original paper. We also list the minimal perturbations working for the unperturbed network. This way we can compare such perturbations with the EGFR and FGFR3 over-expressed variants. We can see that with our method we were able to not only replicate all solutions from [Gri+13], but also conveniently obtain more perturbation options, including the truly minimal controls (if we consider over-expressions of EGFR or FGFR3 as perturbations, the further perturbations to these networks lead to a non-minimal perturbations in most of the cases).

The interest of the original MAPK study [Gri+13] is also to observe the impact on phenotypes caused by various gain- or loss-of-function mutations. Here, authors replace such functions with constants to simulate these effects. Nonetheless, such an approach could be too restrictive: a mutation could alter the function in unpredictable ways instead of knocking-out (resp. over-expressing) the variable permanently. To model such mutations, we employ the uninterpreted functions of the PSBN framework. This application is demonstrated in Table 4. Here, we selected apoptosis phenotype as the phenotype of interest (the "healthy" phenotype preserving non-cancerous cell behaviour). We then replace the dynamics of variables studied in [Gri+13] with uninterpreted functions in the full MAPK model.

Our method performs well in spite of the significant amount of interpretations introduced by the model uncertainty. Moreover, we see that perturbations with relatively small size are still capable of successful control. The obtained observations can be for example used to refute hypotheses about model's update functions. If a candidate perturbation is shown as non-viable, this indicates that the interpretations where such perturbation works do not represent the true dynamics of the system. This can guide further refinements of the partially specified model.

5.4 SUMMARY

In this chapter, we introduced a novel method for phenotype control in partially specified Boolean networks. We first defined the problem of phenotype control and then presented a semi-symbolic algorithm to solve it. Finally, we evaluated our method on real-world models and demonstrated its practical applicability and performance. Our results show that our method can efficiently compute phenotype control strategies even in the presence of a model uncertainty, making it a valuable tool for analyzing complex biological systems.

6 SOFTWARE

In this chapter, we describe the technical details of the software tools we developed for the control of Boolean networks. We first give an overview of the software. Then we describe the implementation underlying backend details of the main components of the software. Finally, we describe the Python package and the web application that we developed to make the software accessible to a wider audience.

6.1 SOFTWARE OVERVIEW

The software for the control of PSBN networks was developed as part of the BioDivine tools[1] [Bar+09] developed by the Sybila [2] (Systems Biology Laboratory) research group at Masaryk University.

The tools were initially oriented on a *colored model checking* of parametrized continuous models of biological systems [Bri+15]. The approach was also later extended to hybrid models [Šmi+20]. The main idea behind colored model checking was encoding the continuous models as discrete color-labelled transition systems, and then applying SMT model checking techniques to analyze their behavior. This approach also enabled solving the parameter synthesis and attractor bifurcation problems for these models [Ben+16; Ben+17].

Building on top of the previous work, the group shifted focus from discretized continuous models to Boolean networks, which are a more direct "natively" discrete modeling formalism. The former parameters of continuous models and their paramters synthesis was reoriented from continuous values of parameters to unknown functions of the network structure.

At first, the tools were mostly focused on the bifurcation analysis of PSBN [Ben+19]. To nurture the adoption of the tool, a web application was developed to make the tool accessible to a wider audience [Ben+20]. The web application allowed users to upload their own partially specified Boolean network models, adjust them and run the bifurcation analysis.

Since globally there was not a lot of tools supporting analysis of PSBN models, the AEON tools were extended to support also other types of analysis. In particular, we implemented the control of PSBN models, which is the main topic of this thesis. The control of PSBN models was implemented as a new component of the software, and it was integrated into the web application as well.

1 https://sybila.fi.muni.cz/tools.html
2 https://sybila.fi.muni.cz/index.html

Later, to make the control of PSBN models more accessible to users who prefer working with code, we also implemented a Python package that provides an interface to the control algorithms implemented in the software [Ben+22a]. The tool also provides all underlying algorithms and data structures such as the symbolic representation of the state space, and the algorithms for computing the attractors and their basins of attraction. The tool is also being extended by higher-level algorithms for the analysis of PSBN models, such model checking of temporal properties, and the analysis of the structure of the state space [Ben+23a; Ben+24].

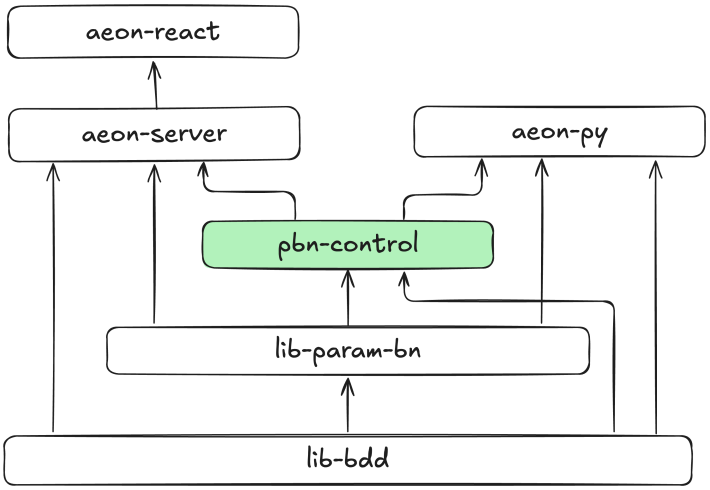


Figure 21: **Biodivine AEON software architecture overview.** The figure depicts components of the biodivine software related to the control of Boolean networks. The green box represents the main Rust library, which contains the core algorithms and data structures for Boolean network control. The arrows indicate the dependencies between the components.

In Figure 21, we give an overview of the software architecture of the AEON tools. The core components are implemented in Rust language because of its performance, memory safety and mature developer experience. Component lib-bdd is a low-level library for the symbolic representation of the state space using binary decision diagrams (BDDs). It is independent of Boolean networks and can be used for any application that requires symbolic representation of state spaces.

Component lib-param-bn is the main library for the performant analysis of PSBN models, which contains the algorithms for computing attractors, reachability algorithms and bifurcation. On top of the two libraries, we implemented the control of PSBN models as a new component of the software. Since the control of PSBN requires custom encoding of the perturbations it also relies on the lib-bdd library for the symbolic representation of the state space. The control component also relies on the lib-pbn library for computing the attrac-

tors and their basins of attraction, which are used in the control algorithms.

Finally, the Rust components serve as the backend of a web application that makes the developed methods accessible through an interactive interface. Aeon-server provides a server functionality such as validation of the uploaded models and progress reporting to interact with the software through a web interface. The web application (aeon-react) then uses just the server component of the software, which provides an API for the users to interact with the software through the web interface. Later, also a Python package aeon-py was implemented to provide a convenient Python interface for all low-level algorithms and data structures.

6.2 CORE PACKAGES

LIB-BDD The lib-bdd library provides a symbolic representation of the state space using binary decision diagrams. It provides basic operations for manipulating BDDs, such as conjunction, disjunction, negation, and quantification.

Compared to many other implementations, each BDD instance manages its own memory. This design simplifies serialization and enables safe sharing between threads, which is particularly beneficial for applications that process large numbers of BDDs concurrently. It also aligns well with the ownership and safety principles emphasized in the Rust programming language.

The implementation provides support for a range of common operations on BDDs. These include standard logical operations, evaluation of Boolean expressions, and transformations between BDD representations and conjunctive or disjunctive normal forms. Additional functionality allows inspection of BDD structures, conversion back to Boolean expressions, and visualization through graph exports. The system also supports several relational-style operations commonly used in symbolic computation, such as projection and restriction. Note that these operations were used in Section 4.3.2 and Section 5.2 for the control of PSBN models.

LIB-PARAM-BN The lib-param-bn library is the main library for the performant analysis of PSBN models, which contains the algorithms for computing attractors, reachability algorithms and bifurcation. In particular, it implements fully *symbolic* representation of the state space of PSBN models using BDDs, that can combine interpretations and states in a single BDD.

Previously the library used semi-symbolic representation of the state space, where only interpretations were represented symbolically, while states were represented explicitly. This approach was used

```
.
├── Cargo.toml
├── Cargo.lock
├── README.md
├── auxiliary_scripts/    # Scripts to do & process experiments
├── models/               # Base experimental models
├── results/              # Raw measured results from experiments
└── src/                  # Library source code
    ├── aeon/             # Simplified algorithms taken over from aeon-server
    ├── bin/              # Alternative console entry-point scripts
    ├── control/          # Source-target control algorithms
    ├── perturbation/     # Perturbed state-transition graph implementation
    ├── phenotype_control/  # Phenotype control algorithms
    ├── lib.rs            # Library declaration
    └── main.rs           # Main console entrypoint
```

Figure 22: **Biodivine pbn-control library file structure overview.** The figure depicts the main files and directories of the `biodivine-pbn-control` library, which implements the control algorithms for partially specified Boolean networks. The file structure follows the standard layout of a Rust library project managed using Cargo.

in the initial version of the control algorithms implemented in this thesis, which are described in Section 4.2.1.

You can also note, that the terminology of library is slightly different from the one used in this thesis. Formerly, we called partially specified Boolean networks as "parametrized Boolean networks", and the library was named accordingly. However, since the parameters of the model are not just parameters in the traditional sense, but rather unknown functions of the network structure, we decided to change the terminology to "partially specified Boolean networks" in later versions of the presentation of our work. The library name was not changed to avoid confusion with the previous versions of the software and to maintain consistency with the existing codebase.

Also, interpretations were formerly called "colors". This is due to the fact that the original approach of colored model checking was based on encoding the continuous models as discrete color-labelled transition systems. The term "color" was used to refer to the different interpretations of the model, which were represented as different colors in the transition system. However, since the term "color" can be confusing in the context of Boolean networks, we decided to change the terminology to "interpretations" in the later group's line of work.

PBN-CONTROL The pbn-control library provides all of the necessary functionality for the control of PSBN models, including the implementation of various control strategies and algorithms.

The core structure of the pbn-control library follows the standard layout of a Rust library project managed using Cargo. The main components of the library are explained below.

The file `Cargo.toml` defines package metadata and dependencies, while `Cargo.lock` ensures reproducible builds by fixing exact dependency versions. The main library implementation resides in the `src/` directory, with `lib.rs` serving as the crate entry point and additional modules organized according to functionality.

The package can be compiled locally by running the cargo command `cargo build` in the project root directory. During this step, Cargo automatically resolves all dependencies, downloads any missing external crates, and compiles both the dependencies and the library source files located in the `src/` directory. The resulting build artifacts are stored in the `target/` directory, typically under `target/debug/` for development builds or `target/release/` when compiled using the optimized command `cargo build -release`.

The folder `auxiliary_scripts/` contains various Python scripts for preparation and analysis of the control results, such as generation of PSBNs from BNs or data analysis. Folders `models/` and `results/` contain the models and results of the experiments described in this thesis.

Finally, the folder `src/` contains the main implementation of the control algorithms. The file `lib.rs` serves as the entry point of the library, where the main public API of the library is defined. The file `main.rs` is the main executable entry point for running the control algorithms demonstration from a command line. Alternative entry points for running experiments from a command line are defined in the `src/bin/` directory. The entry points can be run using command `cargo run` (or `cargo run -release` for optimized version) optionally followed by the name of the entry point (e.g. `cargo run -bin <bin_name>`).

The folder `src/aeon/` contains the implementation of the high level algorithms taken over from aeon-server package. Since aeon-server implements the algorithms with a focus on the web application, the codebase of the server was not designed to be easily reusable for other purposes. Therefore, we decided to implement a lighweight version of the algorithms for reachability, attractor computation and trap set computation here in the pbn-control library. These algorithms are described in Section 4.3.3 and Section 5.2.2.

In `src/perturbations/` we implemented the encoding of perturbations as function symbols, that are added to the original PSBN model. This allows us to use the same symbolic representation of the state space for both the original model and the perturbations, which is crucial for the performance of the control algorithms. We use the same encoding for both source-target control and phenotype control. For encoding perturbations, we use only a single function symbol per variable, which is used when computing control with temporary and permanent perturbations. The concept of this encoding is described in Section 4.3.1.

Finally, `src/control/` and `src/phenotype_control/` folders contain the implementation of the algorithms described in Section 4.3.4 and Section 5.2.2, respectively. Since we have shown, that symbolic one-step control is superior to semi-symbolic one-step control, the current version of the control algorithms implemented in the software relies on the fully symbolic representation of the state space provided by the lib-param-bn library. However, the initial version of the source-target one-step control algorithm implemented in this thesis, which are described in Section 4.2.1, relied on the former semi-symbolic representation of the state space.

6.3 AEON WEB APPLICATION

AEON web application is a web-based interface for the analysis of PSBN models. It allows users to upload their own models, adjust them and run various analyses, including the control of PSBN models. The initial version of the web application was developed to make the bifurcation analysis of PSBN models accessible to a wider audience. The initial version was first released in 2020 [Ben+20].

In version 0.5.0 of the web application, we integrated the control of PSBN models as a new component of the software [Ves+25]. During implementation of the new control module, it was discovered, that technical debt in the codebase of the web application was making it difficult to implement the new control module. Therefore, it was decided to refactor the web application to make it more modular and maintainable. This has led to a major refactor of the web application, which was released in version 0.6.0 and which is also briefly presented here.

AEON tool is available at `https://biodivine.fi.muni.cz/aeon/` as an online tool. Due to the potential high computational complexity of the analyses, Sybila group does not provide a public instance of the web application server, but instead provides a downloadable version of the server that users can run on their own machines.

The main flow of how a model can be uploaded and analyzed in the web application is shown in Figure 23. The user first needs to download and run the server component of the software on their own machine. Then, they can access the web application functionality through a web browser. The user can upload their own PSBN model in the form of a text file, which is then validated by the server or select one of the example models provided by the web application. Once the model is loaded, the user can adjust the model by changing the model functions, or by changing the regulations of the model.

After the model is loaded, the user can run various analyses on the model, including the permanent phenotype control of PSBN models. Source-target control is not support via the AEON UI. It could be implemented later if there is a user interest. The control interface of the

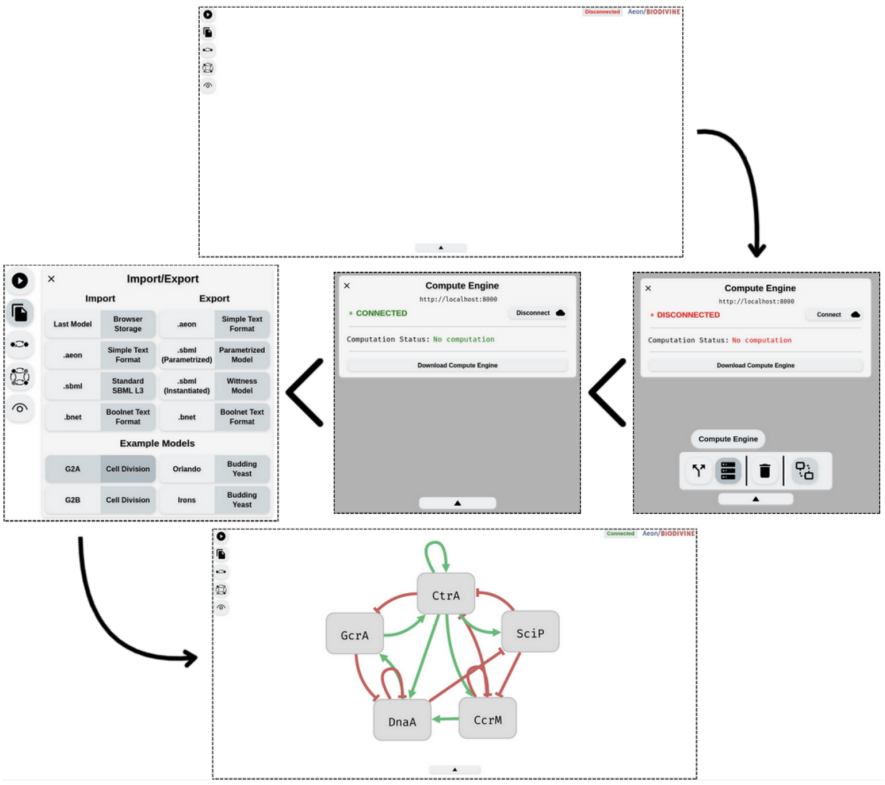


Figure 23: **Biodivine AEON web application interface.** From [Ves]. The figure shows the main interface of the AEON web application, where users can upload and load their own PSBN.

web application is shown in Figure 24. The user can select variables that can be perturbed, the type of the phenotype control (allowed, required or forbidden oscillation) and the target phenotype that they want to reach. The user can also adjust the parameters of the control algorithm, such as the minimal required robustness, maximum number of perturbations allowed or maximum number of results.

After the control algorithm is run, the results are displayed in the web application, where the user can inspect the results and download in a CSV format for further analysis. The results include the comprehensive statistics about the control problem, such as elapsed time, number of interpretations that can be controlled, the number of perturbations found, highest robustness achieved, and other relevant information. Moreover, a full list of all possible perturbations that can be applied to reach the target phenotype is also available. The list can be filtered and sorted by various criteria, such as the number of perturbations, the robustness of the perturbation, or the variables that are perturbed.

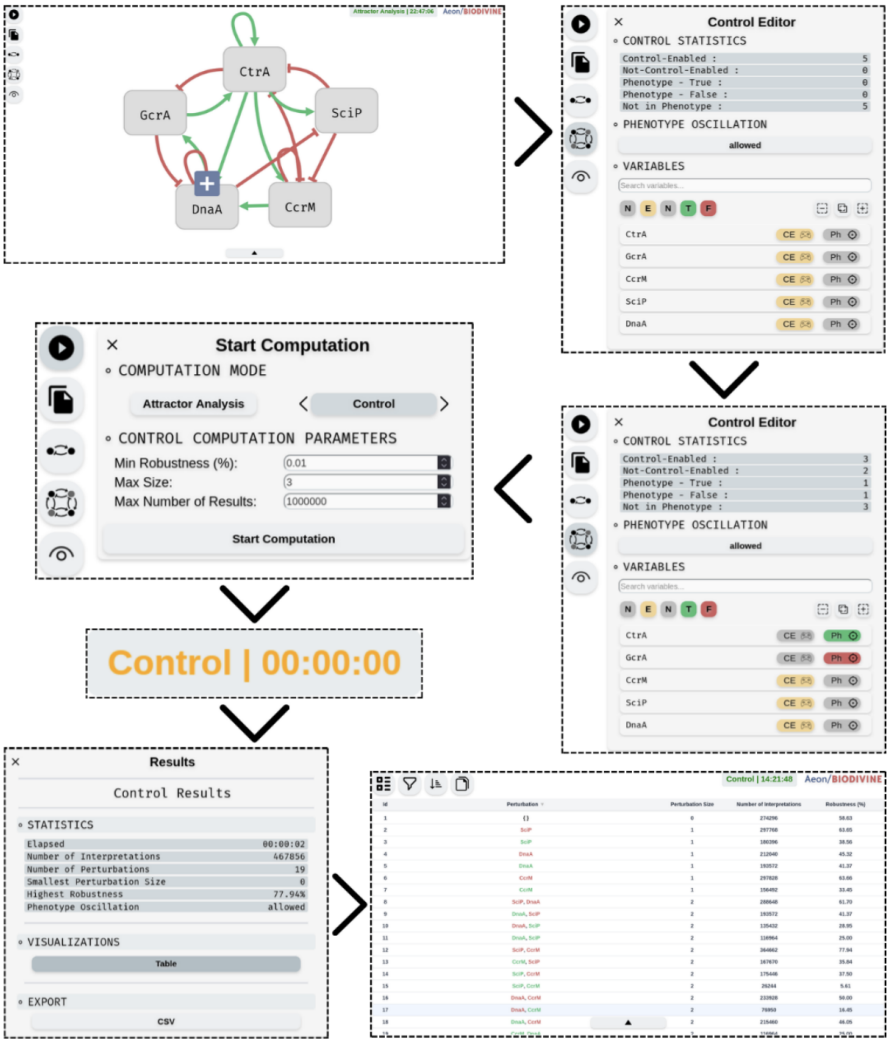


Figure 24: **Biodivine AEON web application control interface.** From [Ves]. The figure shows the control interface of the AEON web application, where users can control their own PSBN.

6.4 AEON.PY

To provide access to the full functionality of the control algorithms beyond what can be conveniently exposed in a web application, a Python package is also available [Ben+22a]. The Python package is available on PyPi[3] and is also a part of the CoLoMoTo docker image [Nal+15].

The CoLoMoTo toolset provides a robust and reproducible computational framework for the analysis of Boolean networks, enabling both qualitative and formal investigation of complex biological regulatory systems. Its Jupyter environment based on Docker ensures interoperability and reproducibility while facilitating scalable exploration of state-space dynamics under different updating schemes.

3 https://pypi.org/project/biodivine-aeon/

Since it is widely used in the community of Boolean network modeling, we decided to make the AEON control algorithms available as part of the CoLoMoTo docker image. This allows users who are already using the CoLoMoTo toolset to easily integrate the control algorithms into their existing workflows.

The Python package provides a convenient interface for users who prefer working with code rather than a web application. It allows users to load their own PSBN models, adjust them and run the control algorithms directly from a Python environment and run a large scale experiments programmatically.

To implement the Python package, we used the maturin[4] library, which provides bindings between Rust and Python. The maturin library allowed us to expose the functionality of the Rust libraries to Python in a seamless way, while also providing good performance and memory safety.

The most convenient way to use the Python package is through a Jupyter notebook, which provides an interactive environment for users to explore the control of PSBN models. The Jupyter notebook allows users to load their own models, adjust them and run the control algorithms, while also providing a convenient way to visualize the results and share them with others.

6.5 SUMMARY

In this chapter, we described the software tools we developed for the control of Boolean networks. We gave an overview of the software architecture and the main components of the software, including the core libraries for symbolic representation of state spaces and the control algorithms.

We also described the web application that we developed to make the software accessible to a wider audience, and the Python package that provides a convenient interface for users who prefer working with code. The software is available online and can be used by researchers interested in analyzing and controlling Boolean network models. In the next chapter, we will demonstrate the use of the software on a case studies with real-world applications.

4 https://www.maturin.rs/

7

APPLICATIONS

This chapter demonstrates the applicability of our method. First, we list practical examples of PSBN control in the field of regenerative medicine (cell reprogramming). We show the way the cell programming can be achieved on a specific example of a myeloid cell. Then we explain a holistic workflow of how the control of partially specified Boolean networks can be used to design a therapy or to refine the model. Finally, we show the full workflow on the second case study that is focused on the MAPK signalling pathway and demonstrates the control-guided model refinement.

7.1 MOTIVATION: CELL REPROGRAMMING

In nature, the most common process by which a cell acquires a different phenotype is *cell differentiation*. A typical example is the differentiation of stem cells into specialised daughter cells in response to the organism's needs. The outcome of this process is often described as a *cell fate* (or a cell-fate decision). Another natural mechanism is *cell dedifferentiation* (or *retrodifferentiation*), in which differentiated cells revert to an earlier developmental state. Dedifferentiation is observed in certain organisms, such as worms and amphibians [JBB11]. In addition, under specific conditions, differentiated cells can convert directly into a different differentiated cell type; this process is known as *transdifferentiation*. These processes are illustrated in Figure 25. The existence of such natural mechanisms has provided important inspiration for the emerging field of *regenerative medicine*.

The organism's need for cell differentiation is communicated via *signalling pathways* [BD10]. These signals are transmitted via cascades of biochemical reactions, predominantly through protein phosphorylation catalyzed by protein kinases. Such signalling dynamics can be modelled using Boolean networks. If we can identify the signals that promote dedifferentiation or transdifferentiation, we may be able to emulate them and artificially induce the corresponding cellular transition. This procedure is known as *cell reprogramming*. In this context, finding efficient perturbations for controlling BNs provides an abstract formulation of the signal-identification problem.

T-cells are a central component of the human immune system. They must differentiate rapidly to mount an effective immune response. However, diseases affecting the immune or haematopoietic system (e.g., leukaemia) can disrupt the signalling pathways that regulate T-cell differentiation, making treatment difficult. Cell reprogramming

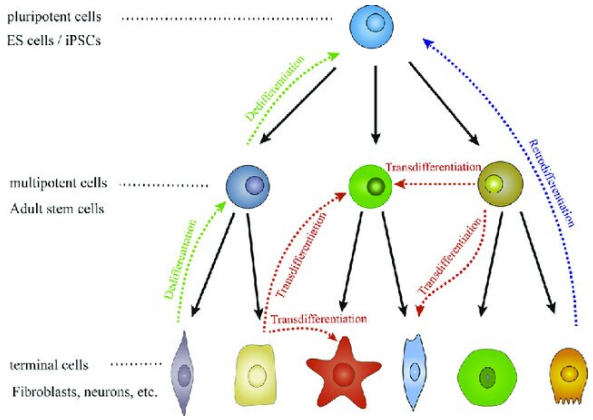


Figure 25: **Cell differentiation.** From [WSA12]. The figure illustrates examples of, and differences among, cell differentiation, dedifferentiation, and transdifferentiation.

approaches may help identify dysregulated signals and support the development of therapeutic interventions [MZ+13; Saa+11; ZA15].

Another important target for cell reprogramming is *embryonic stem cells*. Because these cells can differentiate into many cell types, they hold considerable clinical promise for applications such as the treatment of diabetes and heart disease, as well as modulation of tumour-related processes and undesired immune responses [HKH11; SD09; TY06; You11; Wer+07].

There are many additional potential applications of cell reprogramming. Examples include the conversion of white adipocytes to brown-like adipocytes as a potential therapy for obesity [Zhu+15], restoration of malfunctioning cells [Mot+08], and, potentially, regeneration or repair of organs [Gol19]. For these reasons, we consider the PSBN control problem to be a timely and broadly applicable research direction.

7.2 MYELOID CASE STUDY

To demonstrate an example of how the control of PSBN can be applied to cell reprogramming, we consider a case study of myeloid cell differentiation. Myeloid cells are a type of blood cell that can differentiate into various cell types, including erythrocytes, megakaryocytes, monocytes, and granulocytes. Understanding the regulatory mechanisms underlying myeloid differentiation is crucial for developing therapies for diseases such as leukaemia and other blood disorders. We will be using model from [Kru+11]. The model has 11 variables and its regulatory network along with the model semantics can be seen in the Figure 26.

We use AEON.py [Ben+22a] to do the analysis of the model and to compute the control. First, we do a standard attractor search to find the attractors of the model. The binary single-state attractors are

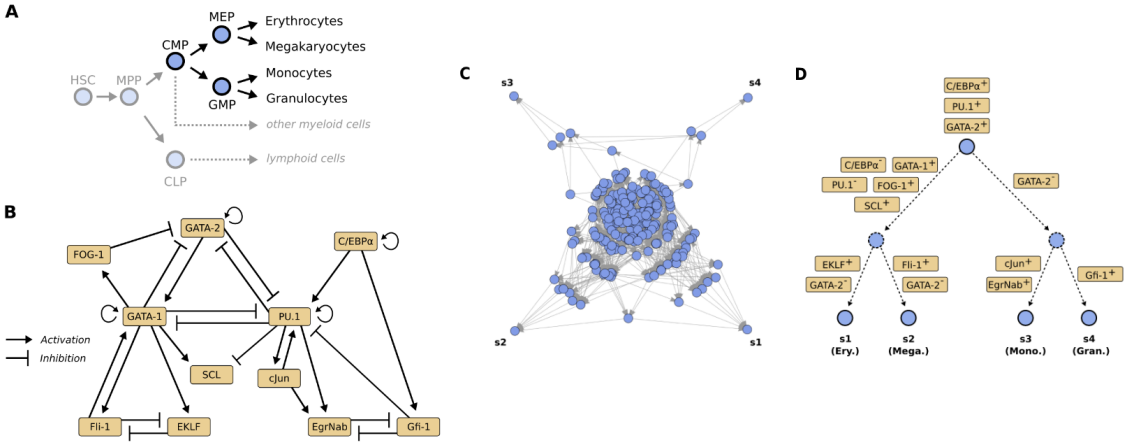


Figure 26: **Myeloid cell differentiation.** From [Kru+11]. The figure illustrates what differentiation process is exhibited by the model (A), the regulatory network of the model (B), a visualization of the state space of the model (C), and the deciding update factors driving a cell from common myeloid progenitor (CMP) to the specific phenotypes (D).

Table 9: **Attractor states of myeloid model**. All attractor sizes are 1 and represent four types of specialized blood cells. Values are binary and represent value assignments of model variables. The highlighted values represent the differentiating factor for each cell type that is considered in the phenotype specification later.

	Erythrocyte	Megakaryocyte	Monocyte	Granulocyte
CEBPa	0	0	0	1
EKLF	1	0	0	0
EgrNab	0	0	1	0
FOG1	1	1	0	0
Fli1	0	1	0	0
GATA1	1	1	0	0
GATA2	0	0	0	0
Gfi1	0	0	0	1
PU1	0	0	1	1
SCL	1	1	0	0
cJun	0	0	1	0

shown in Table 9. Even though the tool does not tell us the semantics of the attractors, we can identify them by comparing the attractor states with the known phenotypes of the myeloid differentiation. The single differentiating factor for each cell type is highlighted in the table.

Since we have results from the attractor search, we can perform the source-target control for all the attractor pairs. The presumption is that eventually the cell converges to one of the possible attrac-

tors and that using source-target control we can find perturbations which can drive the cell from one attractor to another. The results of the control are shown in Table 10. We can see that in some cases one-step perturbations are bigger than temporary or permanent perturbations. The worst case is reprogramming from megakaryocytes to granulocytes, where one-size perturbation needs to be of size 4 while temporary or permanent perturbation suffice with size 2. Also, even though temporary and permanent perturbation are of the same size, for some cases there are more options for temporary perturbations than for permanent ones.

Table 10: **Source-target control between attractors.** Perturbation values are written as x=1 (True) and x=0 (False). When a perturbation contains multiple assignments, they are grouped in braces $\{\cdot\}$.

Source	Target	One-step	Temporary	Permanent
Ery.	Mega.	{EKLF=0,Fli1=1}	Fli1=1;EKLF=0	Fli1=1;EKLF=0
Ery.	Mono.	PU1=1	PU1=1	PU1=1
Ery.	Gran.	{CEBPa=1,GATA1=0,Gfi1=1}	{CEBPa=1,cJun=0};	{CEBPa=1,cJun=0,Gfi1=1}
			{CEBPa=1,EgrNab=0};	{CEBPa=1,EgrNab=0};
			{CEBPa=1,Gfi1=1}	{CEBPa=1,Gfi1=1}
Mega.	Ery.	{EKLF=1,Fli1=0}	Fli1=0; EKLF=1	Fli1=0; EKLF=1
Mega.	Mono.	PU1=1	PU1=1	PU1=1
Mega.	Gran.	{CEBPa=1,Gfi1=1,PU1=1,SCL=0}	{CEBPa=1,cJun=0};	{CEBPa=1, cJun=0};
		{CEBPa=1,Gfi1=1,PU1=1,GATA1=0}	{CEBPa=1,EgrNab=0};	{CEBPa=1,EgrNab=0};
		{CEBPa=1,Gfi1=1,Fli1=0,GATA1=0}	{CEBPa=1,Gfi1=1}	{CEBPa=1,Gfi1=1}
Mono.	Ery.	{EKLF=1,GATA1=1,PU1=0}	{Fli1=0, GATA2=1, PU1=0}; ...	{Fli1=0, GATA2=1, PU1=0}; ...
			... *5 more options*	{EKLF=1, GATA2=1, PU1=0};
Mono.	Mega.	{Fli1=1,GATA1=1,PU1=0}	{PU1=0, Fli1=1}	{Fli1=1, PU1=0}
Mono.	Gran.	{CEBPa=1,EgrNab=0,Gfi1=1}	{CEBPa=1, cJun=0};	{CEBPa=1, cJun=0};
			... *3 more options*	... *2 more options*
Gran.	Ery.	{CEBPa=0,EKLF=1,GATA1=1,PU1=0}	{Fli1=0,GATA2=1,PU1=0}; ...	{Fli1=0,GATA2=1,PU1=0};
			... *5 more options*	{EKLF=1,GATA2=1,PU1=0}
Gran.	Mega.	{CEBPa=0,GATA1=1,Fli1=1,PU1=0}	{PU1=0, Fli1=1}	{Fli1=1, PU1=0}
Gran.	Mono.	CEBPa=0	CEBPa=0	CEBPa=0

Next, we can also run phenotype control to find the perturbations that stabilize the network regardless of the source attractor. The results of the phenotype control are shown in Figure 27. We can see, that in some cases, like in case of megakaryocytes, the minimal perturbation size is the same for source-target and phenotype control, while with control to other cell types we can fine-tune the perturbation to the known source and use a subset of the complete phenotype perturbation. Nonetheless, the phenotype control is always a superset of the source-target control, which is consistent with the fact that the phenotype control is a more general problem.

Finally, we can briefly compare our results with the source paper of the model that is depicted in Figure 26. We can see, that our frame-

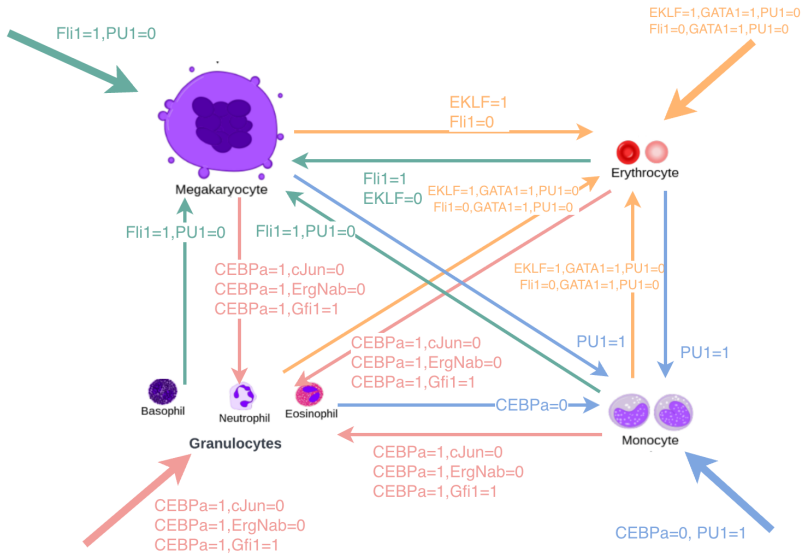


Figure 27: **Control results for myeloid model.** The figure shows the results of control for each cell type in the myeloid model. The arrows going from one cell type to another represent results of permanent source-target control while the arrows going into a cell type (without any source) show results of phenotype control. Each row of the arrow label represents one perturbation. The color of each arrow depends on the control's target – green for megakaryocytes, blue for monocytes, orange for erythrocytes and red for granulocytes.

work yielded results consistent with the original study, since no perturbation negates the perturbation evolution depicted in branches (the root is only initial state and it can be negated). Sometimes, the negation of factors that are driving to one cell type work as a driver to another cell type, which is consistent with the known biology of the system. For example, the perturbations to erythrocytes can be enabled either by over-expression of EKLF (as depicted in the original study) or by the knock-out of Fli1 which over-expression is driving to megakaryocytes.

In order to fully demonstrate the capability of the developed methods, we decided to simulate partial specification of the model by arbitrarily introducing some unknown functions in the model. The model adjustments can be seen in Table 11.

However, attractor analysis of the partially specified model revealed that some attractors satisfy multiple phenotype specifications when phenotypes are defined by the expression of a single marker. For example, there was an attractor that had $\texttt{EKLF=1}$ and $\texttt{Gfi1=1}$, therefore it was shared between erythrocytes and granulocytes. To resolve this issue, we adjust the phenotype specification to require only one marker to be set to true, while other markers were required to be false. For example, the phenotype specification for erythrocytes was changed from $\texttt{EKLF=1}$ to $\texttt{EKLF=1} \wedge \texttt{Fli1=0} \wedge \texttt{Gfi1=0} \wedge \texttt{cJun=0}$. This ad-

Table 11: **Functions of partially specified myeloid model.** The functions of the original model are shown in the second column, while the functions of the partially specified model are shown in the third column. The symbol '—' means, that the function is not changed. The new functions are defined as functions of the original variables and thus they are consistent with the original model.

Variable	Function	New function
GATA2	$\texttt{GATA2} \wedge \neg(\texttt{GATA1} \wedge \texttt{FOG1}) \wedge \neg\texttt{PU1}$	$\mathrm{f}(\texttt{GATA2}, \texttt{GATA1}, \texttt{FOG1}, \texttt{PU1})$
GATA1	$(\texttt{GATA1} \vee \texttt{GATA2} \vee \texttt{Fli1}) \wedge \neg\texttt{PU1}$	—
FOG1	GATA1	—
EKLF	$\texttt{GATA1} \wedge \neg\texttt{Fli1}$	—
Fli1	$\texttt{GATA1} \wedge \neg\texttt{EKLF}$	—
SCL	$\texttt{GATA1} \wedge \neg\texttt{PU1}$	—
CEBPa	$\texttt{CEBPa} \wedge \neg(\texttt{GATA1} \wedge \texttt{FOG1} \wedge \texttt{SCL})$	$\mathrm{g}(\texttt{CEBPa}, \texttt{GATA1}, \texttt{FOG1}, \texttt{SCL})$
PU1	$(\texttt{CEBPa} \vee \texttt{PU1}) \wedge \neg(\texttt{GATA1} \vee \texttt{GATA2})$	$\mathrm{h}(\texttt{CEBPa}, \texttt{PU1}, \texttt{GATA1}, \texttt{GATA2})$
cJun	$\texttt{PU1} \wedge \neg\texttt{Gfi1}$	—
EgrNab	$(\texttt{PU1} \wedge \neg\texttt{cJun}) \wedge \neg\texttt{Gfi1}$	—
Gfi1	$\texttt{CEBPa} \wedge \neg\texttt{EgrNab}$	—

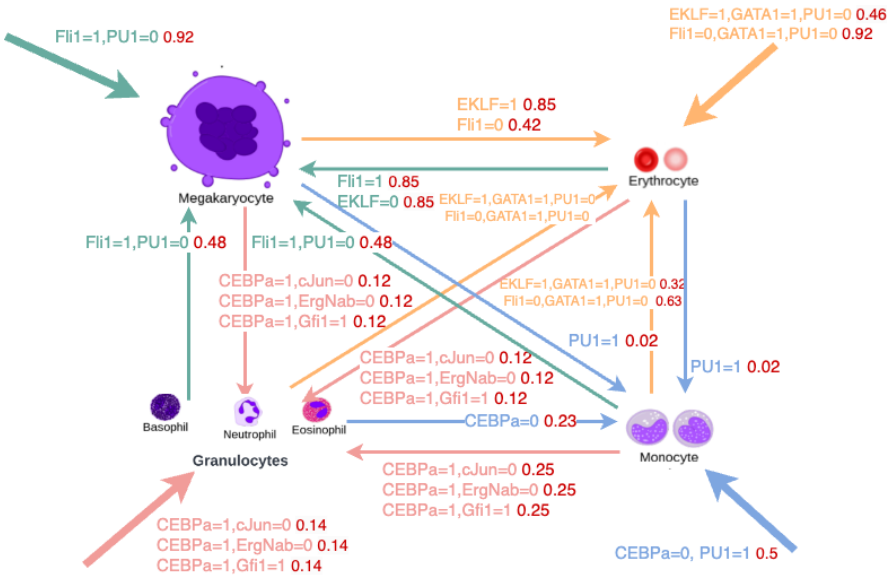


Figure 28: **Control results for partially specified myeloid model.** The figure shows the results from Figure 27 in the partially specified model. The considered perturbations are the equal to the ones in the fully specified model, but they are also labelled with the robustness in the partially specified model.

justment ensured that the phenotypes are disjoint, and thus we can perform the control analysis without any issues. The results of the control in the partially specified model are shown in Figure 28.

What is interesting, is that phenotype control in some cases, despite being global and therefore solving a harder problem in a sense has more robust perturbation than source-target control (for example, for megakaryocytes or monocytes). This is because the source-target control is more restrictive and requires stabilization in one specific state, while the phenotype control requires stabilization allows for stabilization in any state of the phenotype, which is more flexible and robust toward model uncertainty. This well illustrates the advantage of the phenotype control over the source-target control in the context of model uncertainty.

In this section we have seen, that introducing uncertainty into the model can lead to unexpected behaviors of the model, such as shared attractors between phenotypes. This can be seen as an example of how the model uncertainty can lead to a loss of model precision. In the next section, we will show how the control can be used to guide the model refinement and thus to lower the model uncertainty and increase the model precision.

7.3 CONTROL WORKFLOW FOR THERAPY AND MODEL REFINEMENT

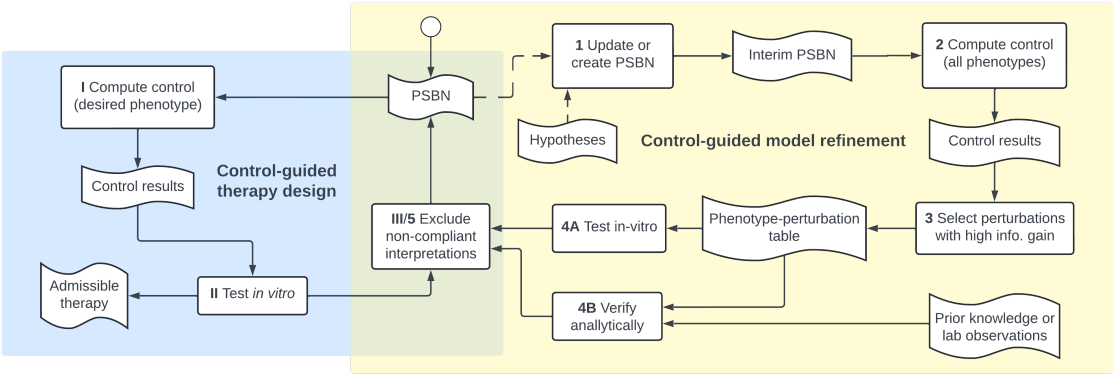


Figure 29: **Workflows of control framework.** The diagram in the picture depicts two workflows of how the phenotype control can be used to achieve two different goals – to find a therapy or to refine a PSBN model. The workflows can be combined or achieve each other's goals as a side effect (e.g., we might refine model after refuting that the obtained therapy works). The rectangle boxes represent the steps of the workflow, while the tape shaped boxes represent the artifacts obtained from the conducted steps.

In this section, we will zoom in on the practical workflow of how the control of partially specified Boolean networks can be used to design a therapy or to refine the model. The workflow is depicted in Figure 29. We will explain the two main branches of the workflow – control-driven therapy design and control-guided model refinement.

We will also show how these two branches can be combined and how they can benefit each other.

7.3.1 *Control-driven therapy design*

In the previous sections, we have demonstrated the application of PSBN control in context of cell reprogramming. In general, we can refer to this process as *therapy design* in case the goal is to find a suitable treatment which achieves a desired (healthy) system behavior [Blo+18; Kim+17; ML+17; Saa+11; Zhu+16].

Also, as we have seen that such an objective can be directly translated to the phenotype control problem where the system behavior is described via PSBN and the target behavior via a phenotype. The output of the control problem is a set of perturbations which can be implemented back to the real-world, for example as gene knockouts and over-expressions.

In an ideal case, the artifact obtained from this workflow is a therapy design that can be proven to work in in-vitro (or in-vivo) experiments. Usually, the preferred perturbations are the ones which are easiest to implement in practice (i.e., with the smallest perturbations or the ones which are the most biologically feasible) [SPP19b]. The second criterion to consider is an expected chance of the therapy to work. This aspect is estimated by the model via a robustness metric of the perturbation [Bri+23].

Nonetheless, since the model is imprecise to some extent, the designed therapy might not work in practice. In such case the observations from the experiment can be used to refine PSBN model. Usually, we can simply refute the interpretations which do not comply with obtained observation. For example, if we expected a perturbation FGFR3=1 to stabilize the network in a phenotype ERK=1 for 60% of interpretations, but the real-world experiment do not stabilize this phenotype under FGFR3 over-expression, we can conclude, that these 60% of interpretations are not representing the studied system correctly, and thus we can discard them from the considered set of interpretations. This would complete the workflow cycle depicted on the left side of the diagram in Figure 29 and we can attempt to recompute the therapy design with the remaining interpretations.

7.3.2 *Control-guided model refinement*

In more complex cases, it is also possible, that the treatment is not replicated in vitro even for a perturbation with 100% robustness. As a result, we might need to rework the model significantly, e.g., by introducing new model parts (regulations and variables) or by changing the existing components. Usually, these model adjustments significantly expand unknown parts of the model, since they are based on

hypotheses and uncertainty. Then, a typical next step is verification of the hypotheses we posed as well as efforts to lower the introduced uncertainty.

As shown on the right side of the diagram in Figure 29, we use phenotype control to guide an experimental design of knockout or over-expression experiments for the model refinement [UD+16]. Let us zoom in to the details of this process.

INITIAL PSBN MODEL We start with an initial PSBN model (Step 1; Fig. Figure 29) and its phenotypes. Such PSBN can be constructed directly from prior knowledge and known assumptions about biological behavior, or it can be an extension of some existing model relying on new information (e.g., newly discovered regulations or pathways).

SCREENING ADMISSIBLE PERTURBATIONS We analyze the feasible perturbations w.r.t. known phenotypes of the PSBN (Step 2; Figure 29). In therapy design, we would seek perturbations of high robustness achieving our phenotype of interest. For refinement, perfect robustness (i.e., $\rho(Q) = 1.0$) is *not* desirable as it means the perturbed interpretations are indistinguishable through phenotype observations. Similarly, if two perturbations achieve equivalent phenotypes across all interpretations, only one needs to be considered further.

SELECTION OF PERTURBATIONS USING INFORMATION-GAIN While the initial screening can significantly reduce the space of admissible perturbations, it likely still remains too large to explore *in vitro*. We therefore prioritize perturbations using a *normalized mutual information score* (NMIS; [KEN83]), quantifying how well is the outcome of a perturbation explained by a specific unknown *feature* of the PSBN (Step 3; Figure 29). We use the following formulas to compute the NMIS:

$$\mathrm{IS}(X;Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

$$H(X) = -\sum_{x} P(x) \log P(x)$$

$$\mathrm{NMIS}(X;Y) = \frac{2\mathrm{IS}(X;Y)}{H(X)H(Y)}$$

To compute NMIS, we observe that interpretations $\mathbb{I}$ can be partitioned into equivalence classes $\mathcal{I}_1, \ldots, \mathcal{I}_k$ based on their outcomes, with each class consisting of interpretations that are indistinguishable by any admissible perturbation. Each class is assigned a set of *behavior labels* and *feature values*.

Here, behavior labels correspond to the phenotypes observed when subject to individual perturbations (collectively, we call these the *be-*

havior pattern of $\mathcal{I}_i$). Similarly, *feature values* are based on selected properties of the underlying class of networks $\mathcal{I}_i$. These features are derived from the unknown elements of the PSBN, such as the uninterpreted update functions or properties of regulations (essentiality, monotonicity, canalization, etc.).

We use NMIS to express the mutual information between the behavior labels achieved by a perturbation Q and the values given by a particular network feature, quantifying how well the network feature predicts the outcome of perturbation Q.

High NMIS for a feature-perturbation pair indicates that observing the perturbed phenotypes should (with high likelihood) either confirm or refute that network feature. The perturbation with the highest NMIS (for some network feature) is selected for *in vitro* testing (Step 4A; Fig. 29). Alternatively, prior knowledge can be introduced to assert the expected phenotypes of the model under perturbation (Step 4B; Fig. 29).

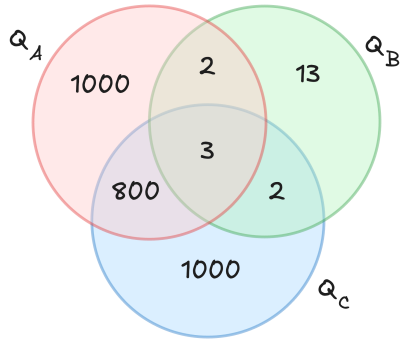
Finally, model instances that do not conform to the expected behavior are excluded (Step 5; Fig. 29), allowing us to restart the workflow with an updated PSBN.

EXAMPLE OF PERTURBATION SELECTION For example, let us suppose, that we consider some PSBN $\mathcal{M}$ *similar* to the one from example in Figure 10 (i.e., a PSBN with multiple variables and functional symbols). The PSBN has 2820 interpretations. and that we compute stabilization in prolifertion (P) phenotype. The result of the control could be perturbations Q_A, Q_B, Q_C, while the union of the interpretations for which they work is exactly $\mathbb{I}(\mathcal{M})$. The cardinality and intersection of the interpretations they cover are shown in Figure 30a.

We can notice, that despite the perturbations Q_A and Q_C are working for exactly the same amount of interpretations, they might control very different sets of particular interpretations. If there would be multiple perturbations which work for exactly same interpretations as the perturbation Q_A, it is not expected to be informative to use them all in the real-world experiments. Finally, we can see, that the perturbation Q_B is also covering distinct sets of perturbations, although their size is much smaller.

If we were in the situation, when we would need to select only two perturbations for the real-world experiments (e.g, due to the high experiment price) we can consider the reduced interpretation sets in *perturbation-phenotype tables* depicted in Figure 30b and Figure 30c. The rows of these tables represent non-overlapping interpretation *classes* with distinct *behavior patterns*, while column values show differing phenotype outcomes of these interpretations under the selected perturbations.

When deciding which two perturbations out of three should be kept, we would be probably inclined to select Q_A and Q_C since in



(a) Interpretation sets controlled by perturb. $\mathrm{Q_A}, \mathrm{Q_B}, \mathrm{Q_C}$.

$\mathrm{Q_A}$	$\mathrm{Q_B}$	$\lvert \mathcal{I} \rvert$
P	P	5
P	A	1800
A	P	15
GA	A	1000

(b) Phenotype-perturb. table for selection of $\mathrm{Q_A}$ and $\mathrm{Q_B}$.

$\mathrm{Q_A}$	$\mathrm{Q_C}$	$\lvert \mathcal{I} \rvert$
P	P	803
P	A	1002
A	P	1002
A	A	13

(c) Phenotype-perturb. table for selection of $\mathrm{Q_A}$ and $\mathrm{Q_C}$.

Figure 30: **An example of interpretations partitioning of working perturbations.** Figure (a) shows sets of interpretations and their intersections for some three distinct perturbations $\mathrm{Q_A}, \mathrm{Q_B}, \mathrm{Q_C}$. Then, the Figure (b) and Figure (c) show phenotype-perturbation matrices if perturbations $\mathrm{Q_A}$ and $\mathrm{Q_B}$ are selected or $\mathrm{Q_A}$ and $\mathrm{Q_C}$ respectively. Each row represents a set of interpretations. The values in the table cells contain a phenotype in which model stabilizes under the given perturbation one of proliferation (P), apoptosis (A) or growth arrest (GA). The last column contains cardinality of the relevant interpretation set.

the worst (and most probable) case we would reduce the model uncertainty to a third of the original interpretation cardinality. On the other hand, if we would select $\mathrm{Q_A}$ and $\mathrm{Q_B}$, there would be a chance, that we would reduce the model uncertainty to a very small fraction of the original one, but the expected chances of this are very low. Nonetheless, the in ideal scenario we would be able to try all three perturbations in the real-world experiments, or we would prefer the perturbations which are easier to implement.

Notice, that in phenotype-perturbation matrix we may not just represent whether the perturbation works or not, but also which phenotype in particular it exhibits. This is particularly useful for the model refinement, since we can then distinguish behavior classes depending on the exhibited phenotype.

Therefore, this table can be then directly used for design of knock-out and over-expression experiments in the real world. Since rows of a table always contain a distinct combination of phenotypes, we expect to be able to match the results from the phenotypes observed in the real-world to one particular row of the phenotype-perturbation table and thus significantly reduce the model uncertainty.

Performing the designed experiments in vitro is an ideal approach to refine the model. However, the experiments are usually costly and might not always be possible to implement. In such cases, we can still observe which model mechanics allow or prevent a perturbation working for some interpretations. For this, purpose, we can infer *features* of an interpretation sets split obtained in the previous step.

	Perturbations		Features	
$\lvert \mathcal{J} \rvert$	Q_A	Q_C	$\exists I \in \mathcal{J}.\ \mathtt{ERK} \rightarrow \mathtt{FRS2}$	$\forall I \in \mathcal{J}.\mathtt{FGFR3} = 0$
803	P	P	1	0
1002	P	A	0	1
1002	A	P	0	0
13	A	A	0	1

Table 12: **Example of phenotype-perturbation tables with extracted features.** The table shows the binarized phenotype-perturbation table for the perturbations Q_A and Q_C from Figure 30. The table is extended with features extracted from the interpretation sets.

For example, consider Table 12, where we have selected two perturbations from the previous example in Figure 30. In the table, we assume that we have selected perturbations Q_A and Q_C. We might be interested in finding out *why* a particular perturbation works for some interpretations and not for the others. For this purpose, we infer the features of the interpretation sets. The vector of phenotypes for perturbations can thus be considered as a *label*, and we can observe their NMIS to measure impact of a feature. Note, that since question we are interested in is whether a perturbation works or not, thus it is more advantageous to consider binary labels (yes/no) for the labels. Given a set of interpretations $\mathcal{J}$, the features we consider are following:

- A regulation is essential in all interpretations $\forall I \in \mathcal{J}.\ \mathtt{GRB2} \rightarrow \mathtt{FGFR3}$ is observable in $\mathcal{M}(I)$.
- A regulation is essential in any interpretation $\exists I \in \mathcal{J}.\ \mathtt{GRB2} \rightarrow \mathtt{FGFR3}$ is observable in $\mathcal{M}(I)$.
- A presence of a particular function instance in all of the given interpretations ($\forall I \in \mathcal{J}.\mathtt{FGFR3} = \mathtt{FGFR3\_stimulus} \wedge \neg \mathtt{GRB2}$ in $\mathcal{M}(I)$). This way we learn, that a particular function is necessary for the phenotype control to work.
- A presence of a particular function instance in any of the given interpretations ($\exists I \in \mathcal{J}.\ \mathtt{FGFR3} = \mathtt{FGFR3\_stimulus} \wedge \neg \mathtt{GRB2}$ in $\mathcal{M}(I)$). This way we might find, that a particular function is preventing a perturbation from working.

Note, that the features can be combined in various ways as well as there might be other features to consider (e.g. monotonicity of regulations, or particular values of the function symbols, etc.). A rule of the thumb is to refine as many potentially relevant features as possible. In our example, we can notice, that perturbation Q_C avoids proliferation exactly if an interpretation has update function FGFR3=0. This could be a surprising observation for the modeler since maybe

they did not expect FGFR3 function to be so trivial, nor Q_C to have such effect on proliferation. If such is the case, approximately a third of the interpretations can be refuted using the analytical approach only.

In more complex cases, it becomes complicated to assess which single feature can completely predict the observed phenotypes. To quantify an extent to which the phenotype is dependent on a given feature we use the mutual information gain metric. Therefore, we consider a single vector of features at a time, and a single vector of phenotype labels yielded by a perturbation, in order to select features best describing each perturbation, to analytically evaluate mechanics of a given perturbation. Since there might be different cardinality for possible phenotype outcomes (i.e. models and perturbations might achieve different number of varying phenotypes), we use normalization for the mutual information gain [VEB09].

To summarize, the workflow in Figure 29 can be used to design a therapy, or guide the model refinement, or both. The cycle can be repeated any amount of times while having both of these goals in mind. The main difference between the two workflows is the selection of perturbations for the real-world experiments. In the therapy design, we are interested in the perturbations which are the most likely to work, while in the model refinement, we are interested in the perturbations which are the most informative.

7.4 MAPK CASE STUDY

The Mitogen-Activated Protein Kinase (MAPK) signaling pathway is a critical cellular signaling pathway that transmits signals from the cell surface to the nucleus in response to various extracellular or internal stimuli, such as growth factors, cytokines, or DNA damage. This pathway regulates a wide range of cellular processes, including proliferation and apoptosis. The disruption of these regulations can lead to diseases, in particular various types of cancer.

We employ our methodology to analyze three models. First, a small-scale abstract model projecting the current knowledge of FGFR3-ERK signalling is considered. Several reported unknown facts are represented by means of a PSBN and control-guided refinement is applied to precise the model with respect to hypothesis suggested in literature.

Second, we consider a larger-scale model [Gri+13] of general MAPK signalling affecting cell proliferation, apoptosis, and growth arrest. Third, we embed FGFR3-ERK mechanism into the model [Gri+13], thus introducing the unknown details of the signalling logics. We explore the extended model further with the control-guided framework.

7.4.1 *Isolated FGFR3 pathway model*

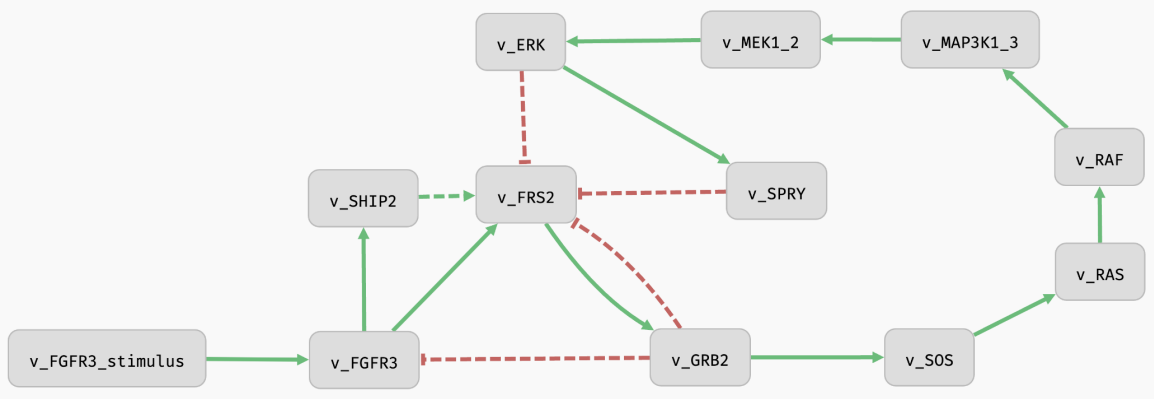


Figure 31: **An isolated FGFR3 pathway model.** The model consists of 12 variables and 16 regulatory interactions. The variables FRS2 and FGFR3, are partially specified. The positive regulations are depicted in green color while negative regulations are red. The lines which are dashed represent regulations which are not required to be essential in the model interpretations. The model serves as an example for demonstrating the capabilities of our method.

Based on our previous research[Haj+16], we consider a minimalistic model representing an abstracted form of FGFR3-MAPK signaling pathway, focusing on the interactions among FGFR3, FRS2, SHIP2 and ERK proteins [Bar+17]. The model is encoded as a PSBN consisting of 12 variables and 16 regulatory interactions. The model contains one input – `FGFR3_stimulus`, which is considered implicitly set to true to simulate constitutive activation of the pathway via suitable FGF ligands. This stimulus can be however disabled (perturbed to the value 0) with the phenotype control.

The update logics of FGFR3 and FRS2 are partially specified reflecting the fact that the combined effect of variables regulating their activity as well as the presence of particular regulations is not well understood [FT+12; Kum+23]. The exact regulatory graph is shown in Figure 31, while the update functions are shown in Table 13.

The dynamics of the final protein component ERK closing the isolated pathway forms the phenotype. In particular, we consider three protein-level phenotypes targeting ERK activity: ERK stabilization, ERK avoidance, and ERK oscillation. Due to the introduced uncertainty, the FGFR3 model has 888 possible interpretations.

Let us we observe the non-perturbed behavior – the "natural" phenotypes of the model. After running a simple algorithm for searching the attractors supported in AEON.py, we learn that the unperturbed model stabilizes in 2 types of attractors – ERK stabilization (556 interpretations) and ERK oscillation (332 interpretations). The ERK avoidance phenotype was not observed in the non-perturbed version of the model. This is due to the presence of the `FGFR3_stimulus` which

Table 13: **FGFR3 pathway model update functions.** The table shows the update functions of the FGFR3 pathway model. The variables FGFR3 and FRS2 are partially specified, while the rest of the variables are fully specified. The function symbols f, g and h are unary function symbols, while $\wedge$ and $\vee$ are binary logical operators.

Variable	Function
FGFR3	$f(\texttt{FGFR3\_stimulus}, \texttt{GRB2})$
GRB2	FRS2
FRS2	$\texttt{FGFR3} \wedge g(\texttt{ERK}, \texttt{GRB2}, \texttt{SPRY}) \vee (\texttt{FGFR3} \wedge h(\texttt{SHIP2}))$
ERK	MEK1_2
SHIP2	FGFR3
SPRY	ERK
SOS	GRB2
RAS	SOS
RAF	RAS
MAP3K1_3	RAF
MEK1_2	MAP3K1_3

we have set to true for our experiments. The stimulus transitively activates ERK via FGFR3, FRS2 and other transitive variables.

INITIAL PERTURBATION SCREENING We computed the phenotype control for all three considered phenotypes. The results are summarized in Table 14. Due to the small size of the model, we list only perturbations up to size one, since perturbations of bigger size did not achieve better results; not even in case of oscillation, where no perturbation has 100% robustness. Even when perturbations of all sizes were computed, no perturbation achieved better robustness towards oscillation phenotype than SHIP2=0. This might be due to the incomplete model design, which we will discuss later.

We can also notice, that perturbation of the same variable to both true and false may lead to the same phenotype (e.g. SPRY in case of oscillation). Similarly, the same perturbations can lead to multiple phenotypes (e.g. SHIP2=1). This is because in these cases the partially specified functions have a greater influence on the network dynamics, than the perturbation itself – thus, another plausible explanation is, that the network behaves the same way as without any perturbations. In case, this is not expected behavior, this can be seen as a flaw introduced in some interpretations of the model which should be therefore refuted.

To probe further all the unexpected model behavior we use the framework of perturbation experiments. First, we need to observe which perturbations yield the most differing results in terms of con-

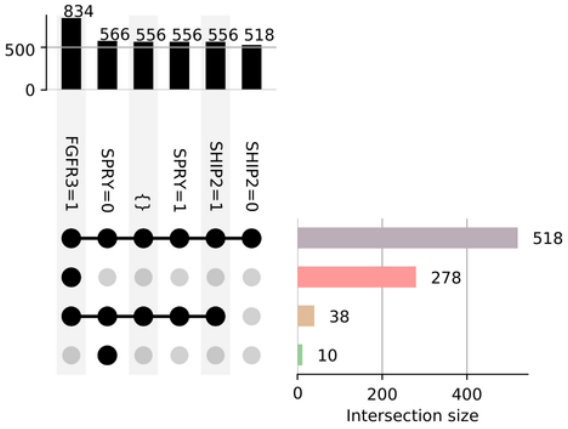
Table 14: **Phenotype control of the isolated FGFR3 pathway model.** The table shows the perturbations that control the model towards the ERK stabilization, avoidance and oscillation phenotypes. The symbol ∅ denotes an absence of perturbations. The robustness of each perturbation is of all interpretation of the original model is shown in the third column.

Phenotype	**Perturbation**	ρ
ERK stabilization	FRS2=1, GRB2=1, MAPK3K1_3=1, MEK1_2=1, RAF=1, RAS=1, SOS=1	1
	FGFR3=1	0.94
	SPRY=0	0.637
	∅, SHIP2=1, SPRY=1	0.626
	SHIP2=0	0.583
ERK avoidance	FGFR3=0, FRS2=0, GRB2=0, MAPK3K1_3=0, MEK1_2=1, RAF=0, RAS=0, SOS=0	1
	FGFR3_st.=0	0.667
	SPRY=1	0.017
ERK oscillation	SHIP2=0	0.41
	∅, SHIP2=1	0.374
	SPRY=0	0.363
	SPRY=1	0.357
	FGFR3=1	0.061
	FGFR3_stimulus=0	0.333

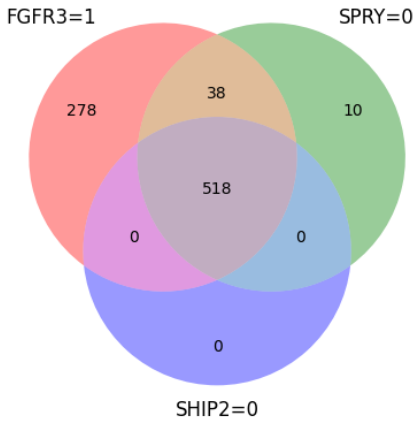
trolled interpretations. To do this, we can depict intersections of the interpretation sets controlled by various perturbations with a robustness lesser than 100%. The results of such an analysis are displayed in Fig. 32. We use Upset plots for the visualization [Lex+14].

We can see, that ERK avoidance case (Subfig. 32c) gives us very straightforward answer of which perturbations should be selected, since there are only two perturbations (FGFR3_stimulus=0 and SPRY=1) covering unique sets of interpretations (the same can be seen from Table 14).

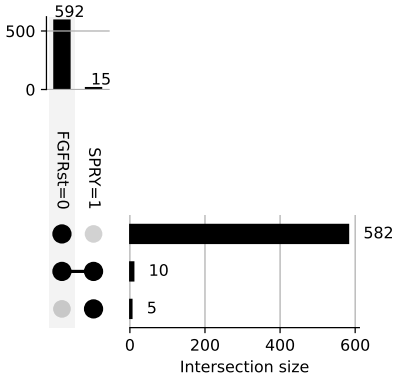
The case of ERK stabilization (Subfig. 32a) is a bit more complicated, as there are multiple perturbations that could be selected. Nonetheless, we can notice, that SPRY=0 and FGFR3=1 should be selected, since they cover some unique set of interpretations. Similarly, SHIP2=0 should be selected, because it is missing some interpreta-



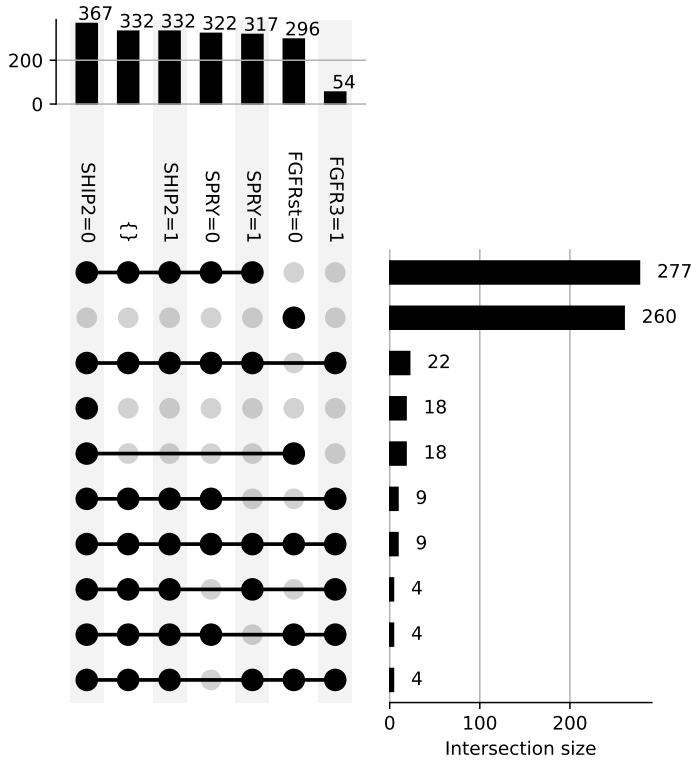
(a) ERK stabilization.



(b) ERK stabilization - Venn diagram.



(c) ERK avoidance.



(d) ERK oscillation.

Figure 32: **Data informing selection of knockout and over-expression experiments to refine model uncertainty.** Figure 32a,Figure 32c, and Figure 32d use UpSet plots [Lex+14] to represent intersections between sets of interpretations controlled by particular perturbations into ERK stabilization, avoidance and oscillation. Figure 32b then represent intersections between controlled interpretations by perturbation candidates selected from Figure 32a.

tions covered by all other perturbations (i.e, it is unique by interpretations which it is *not* covering). Finally, as we can see in the Sub-

fig. 32b, these three perturbations are sufficient to cover all subset cases of the ERK stabilization.

Last but not least, we consider phenotype control to oscillating ERK (Subfig. 32d). We can notice, that we have already selected all other perturbations to be in the perturbation experiments apart from an empty perturbation and SHIP2=1. However, these two perturbations have exactly the same behavior, and thus we can arbitrarily choose any of them – we choose an empty perturbation (since it would be expected to be easier to implement). The results of the perturbation experiments are summarized in Table 17.

Table 15: **Information gain of interpretation features.** The table shows information gain of the three best performing features (four in case of exact match) per each labelling (assessment whether a perturbation is working). The features highlighted in blue were selected to be considered in the next experiment.

Perturbation	Feature	NMIS
∅	∃SPRY→FRS2 (F_1)	0.364
	∀SPRY→FRS2 (F_2)	0.279
	∃SPRY→SHIP2	0.279
SPRY=0	∃FGFR3 = FGFR3_stimulus ∧ ¬GRB2 (F_3)	0.529
	∀GRB2→FGFR3 (F_4)	0.293
	∀FRS2 = FGFR_stimulus	0.293
SHIP2=0	∀SPRY→FRS2 (F_2)	0.855
	∀FRS2 = FGFR3 ∧ SHIP2 (F_5)	0.735
	∃.FRS2 = FGFR3 ∧ SHIP2	0.735
FGFR3_stimulus=0	∀FGFR3 = ¬GRB2 ∨ FGFR3_stimulus (F_6)	1.000
	∃FGFR3 = ¬GRB2 ∨ FGFR3_stimulus	1.000
SPRY=1	∃SPRY→FRS2 (F_1)	0.485
	∃FRS2 = FGFR3 ∧ ¬SPRY (F_7)	0.437
FGFR3=1	∀SHIP2→FRS2 (F_8)	0.562
	∃SPRY→FRS2 (F_1)	0.562
	∀SPRY→FRS2 (F_2)	0.442
	∃SHIP2→FRS2	0.442

Before diving into the results of the interpretation split and model behavior under differing perturbations we might first pose the question: “Why a perturbation works for some particular set of interpretations and not for others?” in order to better explain the observed model behavior. To inquire that, we can look into a set of common *features* of the labelled interpretations, where *label* is an assessment

Table 16: **Mutual information score of interpretation features.** NMIS of the three best performing features (four in case of equal scores) per each labeling (assessment of phenotypes under perturbation). We then consider only non-redundant features (e.g., an existential feature is redundant if its universal variant has an equal or higher NMIS); these are assigned names $F_1, \ldots, F_8$.

	Pert.		**Feature**	**NMIS**
	$\emptyset$	F_1	$\forall$SPRY$\rightarrow$FRS2	0.364
		F_2	$\exists$SPRY$\rightarrow$FRS2	0.279
			$\forall$SPRY$\rightarrow$SHIP2	0.279
Q_1	SPRY=0	F_3	$\exists$FGFR3 = FGFR3_st. $\wedge \neg$GRB2	0.529
		F_4	$\exists$GRB2$\rightarrow$FGFR3	0.293
			$\exists$FRS2 = FGFR3_st.	0.293
Q_2	SHIP2=0	F_2	$\exists$SPRY$\rightarrow$FRS2	0.855
		F_5	$\forall$FRS2 = FGFR3 $\wedge$ SHIP2	0.735
			$\exists$FRS2 = FGFR3 $\wedge$ SHIP2	0.735
Q_3	FGFR3_st.=0	F_6	$\forall$FGFR3 = $\neg$GRB2 $\vee$ FGFR3_st.	1.000
			$\exists$FGFR3 = $\neg$GRB2 $\vee$ FGFR3_st.	1.000
Q_4	SPRY=1	F_1	$\forall$SPRY$\rightarrow$FRS2	0.485
		F_7	$\exists$FRS2 = FGFR3 $\wedge \neg$SPRY	0.437
Q_5	FGFR3=1	F_8	$\exists$SHIP2$\rightarrow$FRS2	0.562
		F_1	$\forall$SPRY$\rightarrow$FRS2	0.562
		F_2	$\exists$SPRY$\rightarrow$FRS2	0.442
			$\forall$SHIP2$\rightarrow$FRS2	0.442

which phenotype (ERK avoidance, activation, or oscillation) is manifested by a given interpretation.

FEATURE-BASED PERTURBATION SELECTION To select the most relevant perturbations out of the 5 remaining, we consider which features of the PSBN best explain the perturbation outcomes. First, the candidate perturbations classify the 888 PSBN interpretations into 16 equivalence classes based on the phenotypes of the perturbed model (i.e., ERK stabilisation, avoidance, or oscillation). We then assign each class a collection of feature values, corresponding to concrete properties of the candidate BNs. Each such feature can be either held universally by *all* members of the class (denoted $\forall$), or existentially (denoted $\exists$) by *some* members of the class (naturally, if a feature is universally held in a class, it is by definition also held existentially). Here, the specific set of features consists of (a) essentiality of the five

Table 17: **Behavior of the isolated FGFR3 pathway under perturbations.** Each row corresponds to a class of interpretations characterized by their perturbation response. Here, # is the row index and $|\mathcal{J}|$ is size of the interpretations class. Then, we describe ERK phenotypes achieved under specific perturbations, as listed in Table 16 (phenotypes represent ERK stabilisation (1), avoidance (0), or oscillation (↻)). The last two columns represent selected model features (also listed in Table 16) representing guaranteed presence of regulations, and guaranteed update function specification. Rows highlighted in gray allow non-trivial ERK activity even with `FGFR3_st.=0`, violating our first assumption. The three green rows are the rows which satisfy both of our control-guided assumptions.

		ERK under perturbation						$\exists A \to B$			$\forall A = ex.$	
#	$\lvert\mathcal{J}\rvert$	$\emptyset$	Q_1	Q_2	Q_3	Q_4	Q_5	F_2	F_4	F_8	F_5	F_6
1	277	↻	↻	↻	0	↻	1	1	1	1	0	0
2	259	1	1	1	0	1	1	0	0	0	0	0
3	259	1	1	1	↻	1	1	0	1	0	0	1
4	22	↻	↻	↻	0	↻	↻	1	1	0	0	0
5	18	1	1	↻	0	1	1	1	0	1	0	0
6	18	1	1	↻	↻	1	1	1	1	1	0	1
7	9	↻	↻	↻	0	0	↻	1	1	0	0	0
8	9	↻	↻	↻	↻	↻	↻	1	1	0	0	1
9	4	↻	1	↻	0	↻	↻	1	0	0	0	0
10	4	↻	1	↻	↻	↻	↻	1	1	0	0	1
11	4	↻	↻	↻	↻	0	↻	1	1	0	0	1
12	1	↻	↻	0	0	↻	1	0	1	1	1	0
13	1	↻	1	↻	↻	0	↻	1	1	0	0	1
14	1	1	1	0	0	1	1	0	0	1	1	0
15	1	1	1	0	↻	1	1	0	1	1	1	1
16	1	↻	1	↻	0	0	↻	1	0	0	0	0

uncertain regulations; (b) individual interpretations of the network's partially unknown update functions.

For every perturbation-feature pair, we compute the NMIS between the perturbation outcome and the feature. This score indicates which features correctly predict phenotypes of the perturbed model. The results of this analysis are presented in Table 16. Here, the most prominent perturbation is `FGFR3_stimulus=0`, where the phenotypes are perfectly predicted by the choice of update function in FGFR3 (NMIS is 1.0). The second most prominent feature is the regulation SPRY $\to$ FRS2 when subject to perturbation `SHIP2=0` (NMIS is 0.855). Both SHIP2 and SPRY are considered uncertain regulators of FRS2 in our PSBN, which is the second variable with a partially specified update function in our model. Consequently, this indicates that the interplay of these two regulators is the most important for determining the update function of FRS2.

MODEL REFINEMENT Based on our results in Table 16, we consider FGFR3_stimulus=0 (Q_3) and SHIP2=0 (Q_2) to be the most relevant for the identification of a concrete model refinement. We performed a literature search to identify plausible experimental results for these two most relevant perturbations, leading to the following assumptions: First, based on [FT+12], we expect FGFR3_stimulus to be *necessary* to achieve ERK stabilisation or oscillation in this isolated model. Second, based on [Faf+18], we observe that SHIP2=0 knock-out causes down-regulation of ERK activity.

To put these assumptions into context, we prepared Table 17, which partitions the PSBN interpretations into equivalence classes based on the effects of our five chosen perturbations on its phenotypes. Within this table, we can easily identify that only 9 equivalence classes (namely 1,2,4,5,7,9,12,14,16) adhere to our first assumption, completely eliminating ERK activity in response to FGFR3_stimulus=0 (Q_3) perturbation. Out of these nine options, the second assumption allows to select rows 5, 12, and 14, as these are the only cases where the activity of ERK is plausibly down-regulated compared to the unperturbed network in response to SHIP2=0 knock-out (Q_2), resulting in 20/888 candidate networks.

To disambiguate between rows 5, 12, and 14, we would have to quantify the effects observed by [Faf+18]. However, based on the fact that [Faf+18] do not report ERK to be completely inactive, we can safely eliminate rows 12 and 14 (where ERK phenotype drops to 0). Furthermore, considering the fact that ERK is known to oscillate [Rai+22], we prefer row 5 as the most likely option.

Interestingly, the 18 networks left in the class at row 5 are completely indistinguishable both in terms of their unperturbed phenotypes and their phenotype response to single-variable perturbations. More importantly, in these 18 networks, it universally holds that GRB2 does *not* regulate FGFR3, but SHIP2 always regulates FRS2 (other regulations remain undetermined).

7.4.2 *Grieco MAPK*

Another model we consider is the Grieco et al. version of MAPK model as presented in [Gri+13]. This model represents a bladder cancer behavior using a combination of three pathways (FGFR3, EFGR, and TGFBR). The model has four inputs (a stimulus for each pathway and a DNA damage). We consider these inputs set to a constant false with an exception to FGFR3 which set to true (as in the previous model).

The regulatory network of the model can be found in Figure 33. The exact function definitions can be found in Appendix B.

Moreover, the MAPK model contains three outputs variables (apoptosis, growth arrest, proliferation). These outputs are then used to

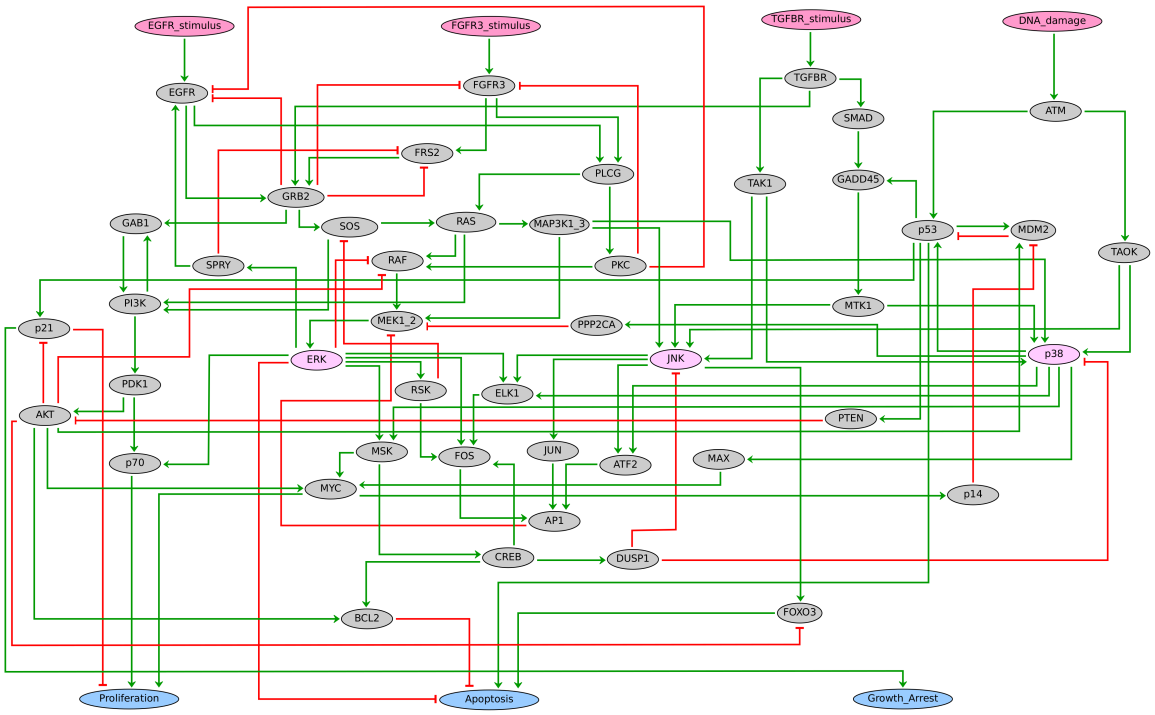


Figure 33: **Regulatory network of the MAPK signalling pathway model.** The model consists of 53 variables and 104 regulatory interactions. The model is directly taken from [Gri+13]

represent the four cardinal phenotypes of the model. These phenotypes and their exact configuration is depicted in Table 18.

Table 18: **MAPK phenotypes.** The cardinal phenotypes of the Grieco MAPK model which will be used throughout the experiments. The first two columns represent the full name and abbreviation of the phenotype. The last three columns represent the model output values which are representing a given phenotype.

Phenotype	Abbreviation	Apoptosis	Growth arrest	Proliferation
Apoptosis	A	1	*	0
Growth arrest	GA	0	1	0
Proliferation	P	0	0	1
No decision	ND	0	0	0

In the study [Gri+13], the full model version with 53 variables was presented. However, the simulations of knockout and over-expression experiments were performed on a reduced variant of the model (having only 17 variables) to observe how they impact the model phenotypes. Our novel approach allows us to consider the full model and all possible perturbations of the model at once, which helps us to obtain more comprehensive results. Table 19 shows the exhaustive enumeration of one-size perturbations which lead to any phenotype (as in Table 18), including all observed combinations of phenotype oscillations.

Table 19: **MAPK perturbations.** A list of perturbations of size up to 1 which drive the MAPK model to a phenotype given in the first column. An ∅ symbol stands for no perturbation while perturbations contained in the square brackets are needed to be combined in order to achieve the perturbation goal. The last column contains a count of unique smallest perturbations achieving the given phenotype.

Phenotype	Perturbations	#
Apoptosis	ATM=1; CREB=0; DNA_damage=1; DUSP1=0; FRS2=1; GRB2=1; TAK1=1; TAOK=1; TGFBR=1; TGFBR_st.=1	10
Growth arrest	[AKT=p21=1]; [ATM=BCL2=1]; [ATM=ERK=1]; [ATM=1,FOXO3=0]; ...	85
No decision	FGFR3=0; FGFR3_st.=0; MAP3K1_3=0; MSK=0; MYC=0; PKC=1; RAS=0	7
Proliferation	ERK=1; MEK1_2=1; RAF=1	3
⟳A-GA	p21=1, p38=1, p53=1	3
⟳A-ND	[AP1=1,p21=0]; [ERK=p21=0]; [GRB2=p21=0]; [JNK=1,p21=1]; ...	10
⟳A-P	N/A	0
⟳GA-ND	[AP1=BCL2=1]; [AP1=1,FOXO3=0]; [AP1=1=JNK=0]; [BCL2=1,ERK=0]; ...	26
⟳GA-P	N/A	0
⟳ND-P	AKT=1; CREB=1; DUSP1=1; MDM2=1; MSK=1; PTEN=0; p14=0; p21=0; p38=0; p53=0	10
⟳A-GA-ND	AP1=1; ERK=0; GRB2=0; JNK=1; MEK1_2=0; PDK1=0; PI3K=0; PPP2CA=1; p70=0	9
⟳A-GA-P	N/A	0
⟳A-ND-P	p21=0	1
⟳GA-ND-P	BCL2=1; FOXO3=0; JNK=0	3
⟳∀	∅	1

Stabilization in all phenotypes apart from growth arrest is possible with perturbations of size one. The unperturbed model behavior is an oscillation through all phenotypes. Moreover, disabling the only input set to true, FGFR3_stimulus, causes the model to stabilize in the no-decision phenotype.

The results also highlight the influence of ERK, which represented the phenotype in the previous model. Since perturbing ERK leads to proliferation, ERK appears to be a strong driver of the proliferation phenotype. However, several previously observed drivers stabilizing ERK (from Section 7.4.1) are not present among the perturbations of the full model. This may be because the full model contains more variables, making their influence on ERK more complex.

7.4.3 *Extended FGFR3 pathway MAPK*

To demonstrate our method on a larger-scale model, we extend the fully specified model [Gri+13] that puts FGFR3-MAPK signalling into a broader context of protein interactions having the crucial impact on cell growth. The model has originally targeted bladder tissue cells. Our extension employs the PSBN framework allowing us to establish a partially specified model that incorporates mechanisms which are not yet understood well. In particular, based on Reactome [Mil+23] and state-of-the-art knowledge [Faf+18; Faf+22] we compile an extended model bringing the updated FGFR3 activation mechanism to the Grieco model.

Table 20: **Changes done to model MAPK model.** The table shows the changes done to the original model [Gri+13] in order to obtain the extended model. The first column shows the variable and a type of the change. The second column shows the original list of regulations or functions. The third column shows the new list of regulations or functions with the changes highlighted in blue font. The regulations marked with + are positives, - are negatives and ? denotes that regulation might not be present in an interpretation. Double ?? denotes that the regulation might be present in an interpretation as both positive and negative. Functions are denoted with f and g are the functions which are not specified in the model, but their instance must adhere to the listed regulations.

	Original	**New**
FGFR3 reg.	FGFR3_stim+, GRB2-, PKC-	FGFR3_stim+, GRB2-?, PKC-?
FGFR3 def.	FGFR3_stim $\wedge \neg$(GRB2 $\vee$ PKC)	FGFR3_stim $\wedge$ f(GRB2,PKC)
FRS2 reg.	FGFR3+, GRB2-, SPRY-	ERK-?, FGFR3+, GRB2-?, SHIP2??, SPRY-?
FRS2 def.	FGFR3 $\wedge \neg$(GRB2 $\vee$ SPRY)	g(FGFR3,ERK,GRB2,SHIP2,SPRY)

This is done by embedding the isolated FGFR3-MAPK pathway studied in previous section. At the level of regulations, the embedding brings in:

1. Addition of SHIP2 affecting the FRS2 adapter phoshorylation capabilities as discussed within the isolated model
2. Addition of the positive feedback from ERK to FRS2 (experimentally studied mostly in the context of chondrocytes)
3. Addition of the positive feedback from ERK to GRB2 (experimentally studied mostly in the context of chondrocytes)

At the level of logical rules, the update functions of FGFR3 and FRS2 are made unspecified with the only constraint that FGFR3_stimulus is left as a necessary precursor of FGFR3 activation. The chosen level of abstraction reflects the fact that biochemical mechanisms specifying how the respective regulations affecting FGFR3 and FRS2 are combined are not currently known. Same as before, we consider the

Table 21: **Extended MAPK model performance.** The table shows the performance of the extended MAPK model. The first column shows the phenotype, the second column shows the relation to the perturbation (standard, avoid, oscillation), and the third column shows the time of the computation. The computations were performed on a computer with AMD Ryzen Threadripper 2990WX 32-Core Processor and 64GB of memory.

ϕ	**relation**	**time**
GA	standard	00:02:45
P	standard	00:24:59
A	standard	00:35:59
A	avoid	00:39:26
ND	avoid	00:40:00
GA	avoid	00:41:56
ND	standard	01:13:09
GA	oscillation	01:17:13
A	oscillation	02:42:08
P	avoid	03:50:26
ND	oscillation	04:05:15
P	oscillation	13:09:52

inputs to be fixed to 0, with an exception to FGFR3 stimulus. The resulting model contains 54 variables and four cardinal phenotypes as described in [Gri+13] – apoptosis (A), growth arrest (GA), no decision (ND), and proliferation (P). The exact overview of the model changes is available in the Table 20.

The goal of the analysis is to apply control-guided refinement to identify perturbations that can reduce the set of possible interpretations of the extended PSBN. We consider the hypotheses on the FGFR3 signalling mechanisms stated in previous section and the pior knowledge based on [Gri+13; Vaq+14; Sta+12].

First, we compute control for all phenotypes including oscillations among them. We restrict the results to perturbations of size up to 1. The performance results are shown in Table 21. We can see, that our method was able to compute perturbations for all cardinal phenotypes in a reasonable time.

The results are shown in Table 22. We can see, that perturbations can achieve all the four cardinal phenotypes as well as achievable oscillations among them. We can also observe the correlation between the ERK activity and the proliferation phenotype – in particular, the ERK stabilization is required for the proliferation phenotype, while apoptosis phenotype drives ERK avoidance.

The results highlighted in blue correspond with the perturbations computed on the original model [Gri+13] (see Table 19). Moreover, the results include several perturbations that have not been observed

Table 22: **Phenotype control of extended MAPK model.** The first column shows the (stable) phenotype or multiple oscillating phenotypes manifested by the model under perturbations listed in the second column. Perturbations in blue font are working in the original [Gri+13] model. Perturbations highlighted in green are the reference perturbations we used for the model refinement. The third column displays the long-term ERK activity, and the last column shows the robustness of respective perturbations. The symbol '*' denotes that we observe amiguous ERK behavior under the given perturbations.

ϕ	**Perturbations**	ERK	ρ for $\mathbb{I}$	ρ for $\mathbb{J}$
A	ATM=1,CREB=0,DNA_dmg=1,DUSP1=0, TAK1=1,TAOK=1,TGFBR=1,TGFBR_st.=1	0	1	1
	PLCG=0	0	0.80	1
	FRS2=1,GRB2=1	0	0.64	1
	AP1=1,ERK=0,GADD45=1,JNK=1,JUN=1, MEK1_2=0,MTK1=1,PPP2CA=1,SMAD=1, p38=1,p53=1,SHIP2=1,SHIP2=0,PKC=1, $\emptyset$,SPRY,RSK=0,PKC=0,SPRY=1	0	$<$0.6	$<$0.5
GA	BCL2=1,FOXO3=0,JNK=0	0	0.61	0
	ERK=1,MEK1_2=0,p21=1	1	$<$0.3	0
ND	AKT=1,PTEN=0;p38=0	0	0.61	0
	MAP3K1_3=0,MSK=0,MYC=0,RAS=0	*	0.50	1
	AKT=0,FGFR3=0,GAB1=0,RSK=1,SOS=0, FGFR3_stimulus=0,p53=0,p70=0, PDK1=0,PI3K=0,PKC=1,PLCG=0,PTEN=1	*	$<$0.3	$<$0.2
P	ERK=1, MEK1_2=1	1	0.3	1
	RAF=1	1	0.19	1
	GAB1=1,GADD45=0,MAX=0,MDM2=1, MTK1=0,p14=0,p53=0,PDK1=1,PI3K=1	1	$<$0.1	0
A-GA	p38=1,p53=1	0	0.25	1
	p21=1	*	0.19	1
ND-P	p53=0	$\circlearrowleft$	0.55	1
	MDM2=1,p14=0,AKT=1,CREB=1,DUSP1=1, MSK=1,PTEN=0,p38=0	$\circlearrowleft$	$<$0.3	1
A-GA-ND	AP1=1,ERK=0,JNK=1,MEK1_2=0,PPP2CA=1	0	0.25	1
	GRB2=0	$\circlearrowleft$	0.2	0.6
	PDK1=0, PI3K=0, p70=0	$\circlearrowleft$	$<$0.2	1
A-ND-P	p21=0	$\circlearrowleft$	0.19	1
GA-ND-P	BCL2=1; FOXO3=0; JNK=0	$\circlearrowleft$	0.19	1
$\circlearrowleft\forall$	$\emptyset$	$\circlearrowleft$	0.28	1

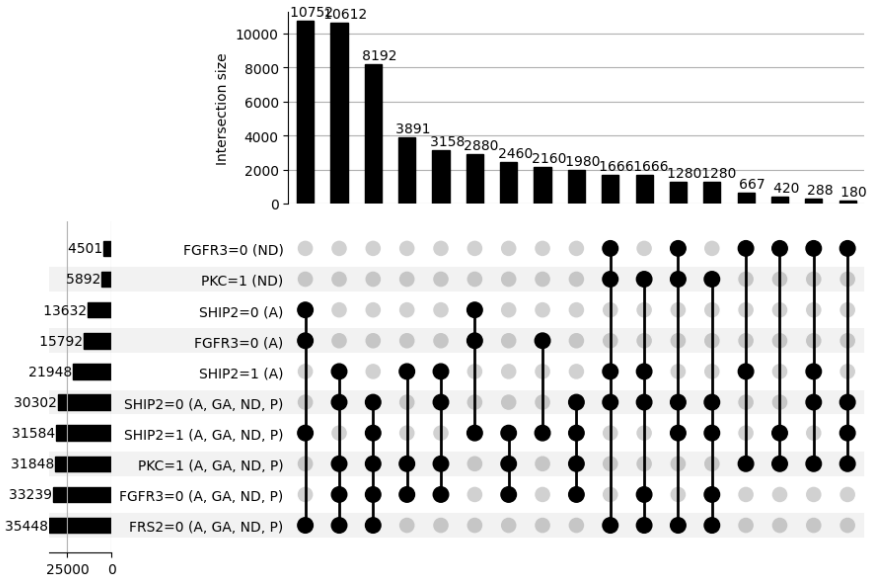


Figure 34: **Upset plot for the FGFR3 extended model.** The plot clearly shows, that each perturbation achieves a unique interpretation partitioning. Moreover, it is not possible to achieve two different phenotypes at once using the same perturbation. The perturbations are not unique, since we consider all phenotypes and perturbations at once.

before. Notably, there is the perturbation PLCG=0 which leads to apoptosis stabilization. This perturbation is not present in the original model, however, it has been observed in [Vaq+14; Sta+12] that PLCG is a driver for cell proliferation while its inhibition can lead to apoptosis. This is a strong evidence that this perturbation is indeed valid, and we can use it for our model refinement.

Based on the discussion above, the perturbations ERK=1 (achieving proliferation) and PLCG=0 (leading to apoptosis) are strongly affecting the long-term behavior of the model. To that end, we use this knowledge to complete the first iteration of the control-guided refinement procedure.

Analogously to the isolated model case study, we compute the new control results shown in the last column of Table 22.

Next, we select perturbations achieving partitioning to distinct behavior patterns. Due to a big number of viable perturbations with non-100% robustness, we first cross-compared all combinations of perturbations and phenotypes they achieve, to obtain a set of perturbation equivalence classes. Such equivalence classes were 10 having 1-3 members. Then, we selected one perturbation from each equivalence class. To verify, that they are indeed unique, we show their intersections in Figure 34. Since we consider all phenotypes and perturbations at once, the perturbations are not unique. Using the UpSet diagram, we confirmed that all selected perturbations contribute to

Table 23: **Behavior of interpretations under perturbations of extended MAPK model.** The first column denotes the size of the interpretation set, while the next five columns denote the behavior of the interpretations under the selected perturbations. The | separates possible bi-stability outcome while ↻ denotes oscillation between all model phenotypes.

$\lvert\mathcal{I}\rvert$	FGFR3=0	FRS2=0	PKC=1	SHIP2=0	SHIP2=1
10752	A	↻	A	A	↻
10612	↻	↻	↻	↻	A
8192	↻	↻	↻	↻	↻
3848	↻	A|ND|P	↻	ND|P	A
3158	↻	A|ND|P	↻	↻	A
2880	A	A|ND|P	A	A	↻
2430	↻	A|ND|P	↻	ND|P	↻
2160	A	A|ND|P	A	A|ND|P	↻
1980	↻	A|ND|P	↻	↻	↻
1666	ND	↻	ND	↻	A
1666	↻	↻	ND	↻	A
1280	ND	↻	ND	↻	↻
1280	↻	↻	ND	↻	↻
624	ND	A|ND|P	↻	ND|P	A
390	ND	A|ND|P	↻	ND|P	↻
288	ND	A|ND|P	↻	↻	A
180	ND	A|ND|P	↻	↻	↻
43	ND	A|ND|P	↻	A|ND|P	A
43	↻	A|ND|P	↻	A|ND|P	A
30	↻	A|ND|P	↻	A|ND|P	↻
30	ND	A|ND|P	↻	A|ND|P	↻

a unique partitioning of the interpretation classes and that no perturbation achieves two different phenotypes at once, as expected for the correctness of our method.

As a result we obtain only five distinct perturbations promising for the model refinement. The perturbations are shown in Table 23. The table displays the admissible perturbations that can be considered for *in vitro* perturbation experiments design leading to potential exclusion of non-complying interpretations of the PSBN. The wet lab testing is necessary to complete the second iteration of the model refinement process, as illustrated in Figure 29.

7.5 SUMMARY

In this chapter, we have demonstrated the applicability of our method on two case studies. The primary application of control was to make cell reprogramming perturbations. This type of application is commonly used for therapy design, where the goal is to find perturbations that can drive the system from a diseased state to a healthy one. We have shown the application on a myeloid differentiation model, where we have found perturbations that can drive the system between different types of cell differentiations.

Next, we have shown a framework for the model refinement using control. We have applied this framework on a MAPK model variants where we demonstrated how to use the control results to refine the model and reduce the set of possible interpretations either analytically or searching for perturbations that can be used for the wet lab experiments design.

CONCLUSIONS

In this thesis, we investigated control of partially specified Boolean networks, a problem that had not been systematically explored before this work. Existing control methods for Boolean networks mostly assume fully specified update functions. This thesis closes part of this gap by bringing control to the partially specified setting and by formulating control objectives that explicitly account for uncertainty in the Boolean update functions.

The first contribution of this work is formalization of source-target control for partially specified Boolean networks. In contrast to classical Boolean network control, where the result is typically a minimal perturbation set, the partially specified setting requires a different view of optimality. Since a perturbation may succeed only for some interpretations of the unknown functions, the methods developed in this thesis compute not only candidate perturbations, but also their robustness with respect to model uncertainty. This provides a more informative output: instead of a single optimal intervention, the user obtains a set of possible interventions together with the subset of interpretations for which they are guaranteed to work.

The second contribution is a suite of algorithms solving this problem. The initial semi-symbolic method addressed one-step source-target control, while the latter fully symbolic method generalized the approach to one-step, temporary, and permanent perturbations. These methods improve the state of the art by avoiding the explicit enumeration of all Boolean network instances, whose number can be very large in partially specified models. Instead, they exploit symbolic state-space representations to reason about states, interpretations, and perturbations together. The experiments presented in this thesis show that temporary and permanent perturbations can provide smaller and more robust control strategies than one-step interventions, demonstrating that the richer perturbation models are not only theoretically more general but also practically useful.

While developing source-target control, we identified an important limitation of attractor-based objectives in the partially specified setting. Different interpretations of the same partially specified model may exhibit shifted or slightly different attractors. As a consequence, requiring convergence to one predefined target attractor can be too restrictive for biological applications, where the exact attractor may be less important than the biological traits it represents. To address this limitation, we introduced phenotype control, in which the target is specified by a set of marker conditions rather than by an explicit attractor. This formulation improves the usability of control in bio-

logical case studies, because it allows the user to specify the desired phenotype through characteristic traits, such as the activation or inhibition of selected genes, without having to know the precise target attractor in advance.

The third contribution is the implementation of the developed algorithms in the AEON ecosystem. This part involves the symbolic manipulation of both the state space and the space of interpretations, as well as a compact encoding of perturbations. The implementation builds on BDD-based symbolic representations and extends the existing AEON infrastructure with control-specific components. In particular, perturbations are encoded in a way that allows them to be handled within the same symbolic framework as the original partially specified model. The methods were implemented in performant Rust libraries, exposed through a Python interface for programmatic use, and integrated into a web-based interface for interactive phenotype control. As a result, the thesis does not only provide theoretical algorithms, but also makes them available to users through practical software.

The fourth contribution is the application of the methods to biological systems. We demonstrated the applicability of the developed control algorithms on real-world models, including case studies related to cell differentiation and MAPK/RTK signalling. These examples show how control can be used not only as a tool for proposing reprogramming interventions, but also as a tool for model refinement. In particular, control results can identify perturbations whose experimental validation would be informative for distinguishing between alternative interpretations of a partially specified model. This connects the control problem back to the broader modelling workflow: rather than treating model construction, analysis, control, and experimental refinement as separate steps, the proposed framework uses control to guide the refinement of the model itself.

At the same time, the work has several limitations. First, the methods focus on variable perturbations, corresponding to fixing some variables to constant values. Other forms of interventions, such as edge, function, or sequential perturbations, remain outside the scope of this thesis. Second, phenotype control is currently developed only for permanent perturbations, while temporary phenotype control remains an open direction. Third, although symbolic methods avoid explicit enumeration of all interpretations, scalability is still limited by the size and structure of the symbolic representation. Large models or models with many unknown functions may still become computationally challenging. Fourth, the biological interpretation of computed perturbations requires care: a perturbation that is valid in the formal model may not always correspond to a feasible, safe, or experimentally realizable biological intervention. Finally, the proposed refinement workflow depends on the availability of suitable experi-

mental readouts that can discriminate between relevant model interpretations.

Several directions for future research follow from these limitations. A direct extension is phenotype control with temporary perturbations, which would combine the biological flexibility of phenotype-based targets with less intrusive intervention strategies. Another promising direction is the combination of control-guided refinement with model checking, so that control objectives and temporal specifications can be used together to select the most informative experiments. Future work could also investigate richer classes of perturbations, including edge or function level interventions, and their relationship to biological mechanisms such as mutations, pathway inhibition, or context-dependent regulation.

Overall, this thesis lays the foundation for control of partially specified Boolean networks as a formal and computational framework for reasoning about interventions under model uncertainty. By combining new problem formulations, symbolic algorithms, practical software support, and biological case studies, it provides a basis for using control not only to guide system behavior, but also to support the iterative refinement of uncertain biological models.

A

NOTATION

Boolean algebra related symbols	
$\mathfrak{b}$	Boolean value (true or false)
$\mathbb{B}$	Set of Boolean values (true or false)
$*$	Undefined Boolean value (might be true or false)
$\mathfrak{f}$	A function
$\mathbb{B}_*$	Set of Boolean values including the undefined value
$x[i \mapsto b]$	A vector x with the i-th element replaced by $\mathfrak{b}$
Networks - common	
$\mathfrak{n}$	Size of a network (number of its nodes – variables)
$\mathcal{G}$	Regulatory network
$\mathbb{V}$	Vertices of a network
$\mathbb{E}$	Edges of a network
S	Subspace of a network
$\mathcal{N}$	Boolean network instance
STG	Asynchronous state transition graph
π	A run (trajectory) of a (partially specified) Boolean network
`variable`	A font for variable names
$\mathcal{E}$(`v1, v2`)	Influence of `v1` on `v2` is observable in STG (also denoted as `v1` $\rightarrow$ `v2`)
$\mathcal{A}$	An attractor of a (partially specified) Boolean network
$\mathbb{A}$	Set of all attractors of a (partially specified) Boolean network
$\mathcal{U}$	An observed character of a network
$\mathcal{T}$	A trait of a network
ϕ	A phenotype of a network
Partially specified Boolean networks	
$\mathcal{M}$	Partially specified Boolean network
$\mathfrak{g}$	A function symbol of a partially specified Boolean network
$\mathbb{G}$	Set of all function symbols of a partially specified Boolean network
$\mathfrak{E}$	An expression of a partially specified Boolean network
I	An interpretation of a partially specified Boolean network
$\mathcal{I}$	A set of interpretations
$\mathbb{I}$	Set of all interpretations of a partially specified Boolean network

Control

Q	A perturbation of a network
$\mathbb{Q}$	A set of all admissible perturbations
ρ	Robustness of a perturbation
$\rho_{\max}$	Maximal robustness of a given set of perturbations
$\rho_{\cup}$	Union robustness of a given set of perturbations

B

DIGITAL ATTACHMENTS

All of the digital attachments related to this thesis are attached in the supplementary files. They are also available online in various repositories. The list of the digital attachments is as follows:

ONE-STEP SEMI-SYMBOLIC SOURCE-TARGET CONTROL

The one-step semi-symbolic source-target control from Section 4.2 is stored in the `src_semisym_control.zip` archive. It is a legacy implementation of `biodivine-pbn-control` library and can be also found in the history of git repository[1].

SOURCE-TARGET AND PHENOTYPE CONTROL

The implementation of methods with symbolic state space representation for source-target (Section 4.3) and phenotype control (Chapter 5) is stored in the `src_symbolic_control.zip` archive. It is a current implementation of `biodivine-pbn-control` library and can be also found in the git repository[2].

MYELOID CASE STUDY

The case study for control of the myeloid differentiation model from Section 7.2 is stored in the `myeloid_case_study.zip` archive. It contains a Jupyter notebook with the code and data for the case study. It can be also found in the git repository[3].

MAPK CASE STUDY

The case study for control of the MAPK model from Section 7.4 is stored in the `mapk_case_study.zip` archive. It contains four Jupyter notebooks with the code and data for the case study and also with a tutorial on how to use AEON.py. It can be also found in the Zenodo archive[4].

1 https://github.com/sybila/biodivine-pbn-control/tree/7d060b6
2 https://github.com/sybila/biodivine-pbn-control
3 https://github.com/sybila/biodivine-aeon-py/tree/main/example/case-study/control
4 https://doi.org/10.5281/zenodo.16886813

OTHERS

The implementation of AEON.py described in Section 6.4 can be found in the git repository[5].

The implementation of the frontend of AEON described in Section 6.3 can be found in the git repository[6].

5 https://github.com/sybila/biodivine-aeon-py
6 https://github.com/sybila/biodivine-aeon-react

BIBLIOGRAPHY

[AL95] Martin Abadi and Leslie Lamport. “Conjoining Specifications.” In: *ACM Transactions on Programming Languages and Systems* 17.3 (1995), pp. 507–534.

[Aku+07] Tatsuya Akutsu, Morihiro Hayashida, Wai-Ki Ching, and Michael K Ng. “Control of Boolean networks: Hardness results and algorithms for tree structured networks.” In: *Journal of theoretical biology* 244.4 (2007), pp. 670–679.

[Alb04] Réka Albert. “Boolean modeling of genetic regulatory networks.” In: *Complex Networks*. Springer, 2004, pp. 459–481.

[Ati+14] Nir Atias et al. “Experimental design schemes for learning Boolean network models.” In: *Bioinformatics* 30.17 (2014), pp. i445–i452.

[Bar21] Lorena A. Barba. “The Python/Jupyter Ecosystem: Today’s Problem-Solving Environment for Computational Science.” In: *Computing in Science & Engineering* 23.3 (2021), pp. 5–9.

[Bar+20] Roberto Barbuti, Roberta Gori, Paolo Milazzo, and Lucia Nasti. “A survey of gene regulatory networks modelling methods: From differential equations, to Boolean and qualitative bioinspired models.” In: *Journal of Membrane Computing* 2.3 (2020), pp. 207–226.

[Bar+17] Jiří Barnat, Nikola Beneš, Luboš Brim, Martin Demko, Matej Hajnal, Samuel Pastva, and David Šafránek. “Detecting attractors in biological models with uncertain parameters.” In: *Computational Methods in Systems Biology*. Vol. 10545. Lecture Notes in Computer Science. Springer, 2017, pp. 40–56.

[Bar+09] Jiří Barnát, Luboš Brim, Ivana Černá, Sven Dražan, Jana Fabriková, Jan Láník, David Šafránek, and Ma Hongwu. “BioDiVinE: A Framework for Parallel Analysis of Biological Models.” In: *Computational Models for Cell Processes*. EPTCS, 2009, pp. 31–45.

[Bau+19] Alexis Baudin, Soumya Paul, Cui Su, and Jun Pang. “Controlling large Boolean networks with single-step perturbations.” In: *Bioinformatics* 35.14 (2019), pp. i558–i567.

[Ben+16] Nikola Beneš, Luboš Brim, Martin Demko, Samuel Pastva, and David Šafránek. “Parallel SMT-based parameter synthesis with application to piecewise multi-affine systems.” In: *International Symposium on Automated Technology for Verification and Analysis*. Springer. 2016, pp. 192–208.

[Ben+24] Nikola Beneš, Luboš Brim, Ondřej Huvar, Samuel Pastva, and David Šafránek. “BNClassifier: Classifying Boolean Models by Dynamic Properties.” In: *International Conference on Computational Methods in Systems Biology*. Springer. 2024, pp. 19–26.

[Ben+22a] Nikola Beneš, Luboš Brim, Ondřej Huvar, Samuel Pastva, David Šafránek, and Eva Šmijáková. “AEON. py: Python library for attractor analysis in asynchronous Boolean networks.” In: *Bioinformatics* 38.21 (2022), pp. 4978–4980.

[Ben+19] Nikola Beneš, Luboš Brim, Samuel Pastva, Jakub Poláček, and David Šafránek. “Formal Analysis of Qualitative Long-Term Behaviour in Parametrised Boolean Networks.” In: *Formal Methods and Software Engineering*. Ed. by Yamine Ait-Ameur and Shengchao Qin. Vol. 11852. Lecture Notes in Computer Science. Springer. 2019, pp. 353–369.

[Ben+20] Nikola Beneš, Luboš Brim, Samuel Pastva, and David Šafránek. “AEON: Attractor bifurcation analysis of parametrised Boolean networks.” In: *International Conference on Computer Aided Verification*. Vol. 12224. Lecture Notes in Computer Science. Springer, 2020, pp. 569–581.

[Ben+21] Nikola Beneš, Luboš Brim, Samuel Pastva, and David Šafránek. “Computing bottom SCCs symbolically using transition guided reduction.” In: *International Conference on Computer Aided Verification*. Springer. 2021, pp. 505–528.

[Ben+22b] Nikola Beneš, Luboš Brim, Samuel Pastva, and David Šafránek. “BDD-based algorithm for SCC decomposition of edge-coloured graphs.” In: *Logical Methods in Computer Science* 18 (2022).

[Ben+23a] Nikola Beneš, Luboš Brim, Ondřej Huvar, Samuel Pastva, and David Šafránek. “Boolean Network Sketches: A Unifying Framework for Logical Model Inference.” In: *Bioinformatics* (2023).

[Ben+17] Nikola Beneš, Luboš Brim, Martin Demko, Samuel Pastva, and David Šafránek. “Pithya: A Parallel Tool for Parameter Synthesis of Piecewise Multi-Affine Dynamical Systems.”

In: *Computer Aided Verification*. Vol. 10426. Lecture Notes in Computer Science. Springer, 2017, pp. 591–598.

[Ben+22c] Nikola Beneš, Luboš Brim, Jakub Kadlecaj, Samuel Pastva, and David Šafránek. "Exploring attractor bifurcations in Boolean networks." eng. In: *BMC Bioinformatics* 23 (2022). ISSN: 1471-2105.

[Ben+23b] Nikola Beneš, Luboš Brim, Samuel Pastva, David Šafránek, and Eva Šmijáková. "Phenotype Control of Partially Specified Boolean Networks." eng. In: *Computational Methods in Systems Biology*. Ed. by Joachim Niehren Jun Pang. Luxembourg City, Luxembourg: Springer Cham, 2023, pp. 18–35. ISBN: 978-3-031-42696-4.

[Blo+18] Peter Bloomingdale, Van Anh Nguyen, Jin Niu, and Donald E Mager. "Boolean network modeling in systems pharmacology." In: *Journal of pharmacokinetics and pharmacodynamics* 45 (2018), pp. 159–180.

[BD21] Enrico Borriello and Bryan C Daniels. "The basis of easy controllability in Boolean networks." In: *Nature Communications* 12.1 (2021), pp. 1–15.

[BD10] R.A. Bradshaw and E.A. Dennis. *Handbook of Cell Signaling*. Handbook of Cell Signaling. Elsevier/Academic Press, 2010.

[Bri+15] Luboš Brim, Milan Češka, Martin Demko, Samuel Pastva, and David Šafránek. "Parameter synthesis by parallel coloured CTL model checking." In: *International Conference on Computational Methods in Systems Biology*. Springer. 2015, pp. 251–263.

[Bri+21] Luboš Brim, Samuel Pastva, David Šafránek, and Eva Šmijáková. "Parallel one-step control of parametrised Boolean networks." In: *Mathematics* 9.5 (2021), p. 560.

[Bri+23] Luboš Brim, Samuel Pastva, David Šafránek, and Eva Šmijáková. "Temporary and permanent control of partially specified Boolean networks." In: *Biosystems* 223 (2023), p. 104795. ISSN: 0303-2647.

[Bry86] Randal E. Bryant. "Graph-based algorithms for Boolean function manipulation." In: *IEEE Transactions on Computers* 35.8 (1986), pp. 677–691.

[Cal+10] Laurence Calzone, Laurent Tournier, Simon Fourquet, Denis Thieffry, Boris Zhivotovsky, Emmanuel Barillot, and Andrei Zinovyev. "Mathematical Modelling of Cell-Fate Decision in Response to Death Receptor Engagement." In: *PLOS Computational Biology* 6.3 (2010), pp. 1–15.

[Car05] Luca Cardelli. “Abstract Machines of Systems Biology.” In: *Transactions on Computational Systems Biology III*. Springer Berlin Heidelberg, 2005, pp. 145–168. ISBN: 978-3-540-31446-2.

[CAS05] Madalena Chaves, Reka Albert, and Eduardo D Sontag. “Robustness and fragility of Boolean models for genetic regulatory networks.” In: *Journal of theoretical biology* 235.3 (2005), pp. 431–449.

[CQL10] Daizhan Cheng, Hongsheng Qi, and Zhiqiang Li. *Analysis and control of Boolean networks: a semi-tensor product approach*. Springer Science & Business Media, 2010.

[Che+16] K.C. Cheng, S.R. Katz, A.Y. Lin, X. Xin, and Y. Ding. “Chapter Four - Whole-Organism Cellular Pathology: A Systems Approach to Phenomics.” In: *Genetics, Genomics and Fish Phenomics*. Vol. 95. Academic Press, 2016, pp. 89–115.

[CD12] Anne B. C. Cherry and George Q. Daley. “Reprogramming cellular identity for regenerative medicine.” In: *Cell* 148.6 (2012), pp. 1110–1122.

[Che+25] Stéphanie Chevalier, Julia Becker, Yujuan Gui, Vincent Noël, Cui Su, Sascha Jung, Laurence Calzone, Andrei Zinovyev, Antonio Del Sol, Jun Pang, et al. “Data-driven inference of Boolean networks from transcriptomes to predict cellular differentiation and reprogramming.” In: *npj Systems Biology and Applications* 11.1 (2025), p. 105.

[Cho+18] Sang-Mok Choo, Byunghyun Ban, Jae Il Joo, and Kwang-Hyun Cho. “The phenotype control kernel of a biomolecular regulatory network.” In: *BMC systems biology* 12.1 (2018), pp. 1–15.

[CF23] Laura Cifuentes Fontanals. “Methods for control strategy identification in Boolean networks.” PhD thesis. 2023.

[CFTS20] Laura Cifuentes Fontanals, Elisa Tonello, and Heike Siebert. “Control Strategy Identification via Trap Spaces in Boolean Networks.” In: *Computational Methods in Systems Biology*. Vol. 12314. Lecture Notes in Computer Science. Springer. 2020, pp. 159–175.

[CFTS22] Laura Cifuentes Fontanals, Elisa Tonello, and Heike Siebert. “Control in Boolean Networks with Model Checking.” In: *Frontiers in Applied Mathematics and Statistics* 8 (Apr. 2022), p. 838546.

[Cla97] Edmund M Clarke. “Model checking.” In: *Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17*. Springer. 1997, pp. 54–56.

[CE81] Edmund M Clarke and E Allen Emerson. "Design and synthesis of synchronization skeletons using branching time temporal logic." In: *Workshop on logic of programs*. Springer. 1981, pp. 52–71.

[Coh+15] David PA Cohen, Loredana Martignetti, Sylvie Robine, Emmanuel Barillot, Andrei Zinovyev, and Laurence Calzone. "Mathematical modelling of molecular pathways enabling tumour cell invasion and migration." In: *PLOS Computational Biology* 11.11 (2015), pp. 1–29.

[Cor+18] Rion B Correia, Alexander J Gates, Xuan Wang, and Luis M Rocha. "CANA: a python package for quantifying control and canalization in Boolean networks." In: *Frontiers in physiology* 9 (2018), p. 1046.

[DL05] Eric Davidson and Michael Levin. "Gene regulatory networks." In: *Proceedings of the National Academy of Sciences* 102.14 (2005), pp. 4935–4935.

[Daw04] Conrado Daws. "Symbolic and parametric model checking of discrete-time Markov chains." In: *International Colloquium on Theoretical Aspects of Computing*. Springer. 2004, pp. 280–294.

[DC14] Jennifer A Doudna and Emmanuelle Charpentier. "The new frontier of genome engineering with CRISPR-Cas9." In: *Science* 346.6213 (2014), p. 1258096.

[Faf+22] B. Fafilek et al. "Expanding horizons of achondroplasia treatment: current options and future developments." In: *Osteoarthritis and Cartilage* 30.4 (2022), pp. 535–544. ISSN: 1063-4584.

[Faf+18] Bohumil Fafilek et al. "The inositol phosphatase SHIP2 enables sustained ERK activation downstream of FGF receptors by recruiting Src kinases." In: *Science signaling* 11.548 (2018), eaap8608.

[FCZ03] Enrique Fernández-Cara and Enrique Zuazua. "Control theory: History, mathematical achievements and perspectives." In: *Boletín de la Sociedad Española de Matemática Aplicada* 26 (2003), pp. 79–140.

[Fie+13] Bernold Fiedler, Atsushi Mochizuki, Gen Kurosawa, and Daisuke Saito. "Dynamics and control at feedback vertex sets. I: Informative and determining nodes in regulatory networks." In: *Journal of Dynamics and Differential Equations* 25.3 (2013), pp. 563–604.

[Fir+98] Andrew Fire, SiQun Xu, Mary K Montgomery, Steven A Kostas, Samuel E Driver, and Craig C Mello. "Potent and specific genetic interference by double-stranded RNA in Caenorhabditis elegans." In: *nature* 391.6669 (1998), pp. 806–811.

[Flo62] JJ Flokentin. "Partial observability and optimal control." In: *International Journal of Electronics* 13.3 (1962), pp. 263–279.

[FT+12] Silvie Foldynova-Trantirkova et al. "Sixteen years and counting: The current understanding of FGFR3 signaling in skeletal dysplasias." In: *Human Mutation* 33.1 (2012), pp. 29–41.

[FLTS22] Cifuentes Fontanals Laura, Elisa Tonello, and Heike Siebert. "Computing trap space-based control strategies for Boolean networks using answer set programming." In: *AIP Conference Proceedings*. Vol. 2611. 1. AIP Publishing. 2022.

[Gat+21] Alexander J Gates, Rion Brattig Correia, Xuan Wang, and Luis M Rocha. "The effective graph reveals redundancy, canalization, and control pathways in biochemical regulation and signaling." In: *Proceedings of the National Academy of Sciences* 118.12 (2021), e2022598118.

[GR16] Alexander J Gates and Luis M Rocha. "Control of complex networks requires both structure and dynamics." In: *Scientific reports* 6.1 (2016), p. 24456.

[GGC16] Liesbet Geris and David Gomez-Cabrero. "An introduction to uncertainty in the development of computational models of biological processes." In: *Uncertainty in Biology*. Springer, 2016, pp. 3–11.

[Gje+20] Enio Gjerga et al. "Converting networks to predictive logic models from perturbation signalling data with CellNOpt." In: *Bioinformatics* 36.16 (2020), pp. 4523–4524.

[Gol19] Michael S. Goligorsky. "New Trends in Regenerative Medicine: Reprogramming and Reconditioning." In: *Journal of the American Society of Nephrology* 30.11 (2019), pp. 2047–2051.

[GIP13] Diana-Elena Gratie, Bogdan Iancu, and Ion Petre. "ODE Analysis of Biological Systems." In: *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Vol. 7938. Lecture Notes in Computer Science. 2013, pp. 29–62.

[Gri+13] Luca Grieco, Laurence Calzone, Isabelle Bernard-Pierrot, François Radvanyi, Brigitte Kahn-Perles, and Denis Thieffry. "Integrative modelling of the influence of MAPK network on cancer cell fate decision." In: *PLOS Computational Biology* 9.10 (2013), e1003286.

[Gro+93] Robert L Grossman, Anil Nerode, Anders P Ravn, and Hans Rischel. *Hybrid systems*. Vol. 736. Springer, 1993.

[Haj+16] Matej Hajnal, David Šafránek, Martin Demko, Samuel Pastva, Pavel Krejčí, and Luboš Brim. "Toward Modelling and Analysis of Transient and Sustained Behaviour of Signalling Pathways." In: *Hybrid Systems Biology*. Ed. by Eugenio Cinquemani and Alexandre Donzé. Springer, 2016, pp. 57–66.

[HIB01] Justin E Harlow III and Franc Brglez. "Design of experiments and evaluation of BDD ordering heuristics." In: *International Journal on Software Tools for Technology Transfer* 3.2 (2001), pp. 193–206.

[HKH11] Carla A. Herberts, Marcel SG Kwa, and Harm PH Hermsen. "Risk factors in the development of stem cell therapy." In: *Journal of Translational Medicine* 9.1 (2011).

[Her+12] Franziska Herrmann, Alexander Groß, Dao Zhou, Hans A Kestler, and Michael Kühl. "A boolean model of the cardiac gene regulatory network determining first and second heart field identity." In: *PloS one* 7.10 (2012), e46798.

[Hoc+13] Gal Hochma, Michael Margaliot, Ettore Fornasini, and Maria Elena Valcher. "Symbolic dynamics of Boolean control networks." In: *Automatica* 49.8 (2013), pp. 2525–2530.

[Hun+02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits Vaandrager. "Linear parametric model checking of timed automata." In: *The Journal of Logic and Algebraic Programming* 52 (2002), pp. 183–220.

[Ito+12] Nobuhisa Ito, Go Kuwahara, Yuta Sukehiro, and Hiromitsu Teratani. "Segmental arterial mediolysis accompanied by renal infarction and pancreatic enlargement: a case report." In: *Journal of Medical Case Reports* 6.1 (2012), pp. 1–5.

[JBB11] Chris Jopling, Stephanie Boue, and Juan Carlos Izpisua Belmonte. "Dedifferentiation, transdifferentiation and reprogramming: Three routes to regeneration." In: *Nature Reviews Molecular Cell Biology* 12.2 (2011), pp. 79–89.

[KEN83] JOHN T. KENT. "Information gain and a general measure of correlation." In: *Biometrika* 70.1 (Apr. 1983), pp. 163–173. ISSN: 0006-3444.

[Kal60] Rudolf E Kalman. “On the general theory of control systems.” In: *Proceedings first international conference on automatic control, Moscow, USSR*. 1960, pp. 481–492.

[Kam+13] Roland Kaminski, Torsten Schaub, Anne Siegel, and Santiago Videla. “Minimal intervention strategies in logical signaling networks with ASP.” In: *Theory and Practice of Logic Programming* 13.4-5 (2013), pp. 675–690.

[Kau69] S.A. Kauffman. “Metabolic stability and epigenesis in randomly constructed genetic nets.” In: *Journal of Theoretical Biology* 22.3 (1969), pp. 437–467. ISSN: 0022-5193.

[Kau04] Stuart Kauffman. “A proposal for using the ensemble approach to understand genetic regulatory networks.” In: *Journal of theoretical biology* 230.4 (2004), pp. 581–590.

[KPC13] Junil Kim, Sang-Min Park, and Kwang-Hyun Cho. “Discovery of a kernel for controlling biomolecular regulatory networks.” In: *Scientific reports* 3.1 (2013), pp. 1–9.

[Kim+17] Yunseong Kim, Sea Choi, Dongkwan Shin, and Kwang-Hyun Cho. “Quantitative evaluation and reversion analysis of the attractor landscapes of an intracellular regulatory network for colorectal cancer.” In: *BMC systems biology* 11 (2017), pp. 1–11.

[Kla+18] Hannes Klarner, Frederike Heinitz, Sarah Nee, and Heike Siebert. “Basins of attraction, commitment sets, and phenotypes of Boolean networks.” In: *IEEE/ACM transactions on computational biology and bioinformatics* 17.4 (2018), pp. 1115–1124.

[KSS17] Hannes Klarner, Adam Streck, and Heike Siebert. “PyBoolNet: a Python package for the generation, analysis and visualization of Boolean networks.” In: *Bioinformatics* 33.5 (2017), pp. 770–772.

[KB05] Konstantin Klemm and Stefan Bornholdt. “Stable and unstable attractors in Boolean networks.” In: *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 72.5 (2005), p. 055101.

[KH11] Koichi Kobayashi and Kunihiko Hiraishi. “Optimal control of asynchronous Boolean networks modeled by Petri nets.” In: *Biological Process & Petri Nets*. CEUR-WS, 2011, pp. 7–20.

[KS14] Stepan Kochemazov and Alexander Semenov. “Using Synchronous Boolean Networks to Model Several Phenomena of Collective Behavior.” In: *PLOS ONE* 9.12 (Dec. 2014), pp. 1–28.

[KT09] Clemens Kreutz and Jens Timmer. “Systems biology: experimental design.” In: *The FEBS journal* 276.4 (2009), pp. 923–942.

[Kri59] Saul A Kripke. “A completeness theorem in modal logic1.” In: *The journal of symbolic logic* 24.1 (1959), pp. 1–14.

[Kri63] Saul A Kripke. “Semantical considerations on modal logic.” In: *Acta philosophica fennica* 16 (1963).

[Kru+11] Jan Krumsiek, Carsten Marr, Timm Schroeder, and Fabian J. Theis. “Hierarchical Differentiation of Myeloid Progenitors Is Encoded in the Transcription Factor Network.” In: *PLOS ONE* 6.8 (2011), pp. 1–10.

[Kum+23] Santhana Kumar et al. “Discovery of a small molecule ligand of FRS2 that inhibits invasion and tumor growth.” In: *Cellular Oncology* 46.2 (2023), pp. 331–356.

[LJ09] Christopher James Langmead and Sumit Kumar Jha. “Symbolic approaches for finding control strategies in Boolean networks.” In: *Journal of Bioinformatics and Computational Biology* 7.02 (2009), pp. 323–338.

[Lev+18] Nicolas Levy, Aurélien Naldi, Céline Hernandez, Denis Thieffry, Andrei Zinovyev, Laurence Calzone, and Loïc Paulevé. “Prediction of mutations to control pathways enabling tumor cell invasion with the CoLoMoTo interactive notebook (tutorial).” In: *Frontiers in physiology* 9 (2018), p. 370902.

[Lex+14] Alexander Lex et al. “UpSet: Visualization of Intersecting Sets.” In: *IEEE TACG* 20.12 (2014), pp. 1983–1992.

[LSB13] Yang-Yu Liu, Jean-Jacques Slotine, and Albert-László Barabási. “Observability of complex systems.” In: *Proceedings of the National Academy of Sciences* 110.7 (2013), pp. 2460–2465.

[Liu+17] Yang Liu, Bowen Li, Hongwei Chen, and Jinde Cao. “Function perturbations on singular Boolean networks.” In: *Automatica* 84 (2017), pp. 36–42.

[Man19] Hugues Mandon. “Algorithms for Cell Reprogramming Strategies in Boolean Networks.” Theses. Université Paris-Saclay, 2019.

[Man+19] Hugues Mandon, Cui Su, Stefan Haar, Jun Pang, and Loïc Paulevé. “Sequential Reprogramming of Boolean Networks Made Practical.” In: *Computational Methods in Systems Biology*. Vol. 11773. Lecture Notes in Computer Science. Springer. 2019, pp. 3–19.

[MA03] Shmoolik Mangan and Uri Alon. “Structure and function of the feed-forward loop network motif.” In: *Proceedings of the National Academy of Sciences* 100.21 (2003), pp. 11980–11985.

[Mar+16] Alberto JM Martin, Calixto Dominguez, Sebastián Contreras-Riquelme, David S Holmes, and Tomas Perez-Acle. “Graphlet based metrics for the comparison of gene regulatory networks.” In: *PLOS ONE* 11.10 (2016).

[Max68] James Clerk Maxwell. “I. On governors.” In: *Proceedings of the Royal Society of London* 16 (1868), pp. 270–283.

[Mel+10] Bence Melykuti, Elias August, Antonis Papachristodoulou, and Hana El-Samad. “Discriminating between rival biochemical network models: three approaches to optimal experiment design.” In: *BMC systems biology* 4 (2010), pp. 1–16.

[ML+17] Luis Fernando Méndez-López, Jose Davila-Velderrain, Elisa Domínguez-Hüttinger, Christian Enríquez-Olguín, Juan Carlos Martínez-García, and Elena R Alvarez-Buylla. “Gene regulatory network underlying the immortalization of epithelial cells.” In: *BMC systems biology* 11 (2017), pp. 1–15.

[Mil+23] Marija Milacic et al. “The Reactome Pathway Knowledgebase 2024.” In: *Nucleic Acids Research* 52.D1 (2023), pp. D672–D678.

[MZ+13] Natasa Miskov-Zivanov, Michael S. Turner, Lawrence P. Kane, Penelope A. Morel, and James R. Faeder. “The Duration of T Cell Stimulation Is a Critical Determinant of Cell Fate and Plasticity.” In: *Science Signaling* 6.300 (2013).

[MGFA19] Mohammad Moradi, Sama Goliaei, and Mohammad-Hadi Foroughmand-Araabi. “A Boolean network control algorithm guided by forward dynamic programming.” In: *PLOS ONE* 14.5 (2019), pp. 1–21.

[Mot+08] Adilson E Motter, Natali Gulbahce, Eivind Almaas, and Albert-László Barabási. “Predicting synthetic rescues in metabolic networks.” In: *Molecular Systems Biology* 4.1 (2008), p. 168.

[Nal18] Aurélien Naldi. “BioLQM: a java toolkit for the manipulation and conversion of logical qualitative models of biological networks.” In: *Frontiers in physiology* 9 (2018), p. 1605.

[Nal+15] Aurélien Naldi et al. “Cooperative development of logical modelling standards and tools with CoLoMoTo.” In: *Bioinformatics* 31.7 (Jan. 2015), pp. 1154–1159. ISSN: 1367-4803.

[PAU23] Loïc PAULEVÉ. “Boolean Networks: Formalism, Semantics and Complexity.” In: *Symbolic Approaches to Modeling and Analysis of Biological Systems* (2023), p. 151.

[PRA25] Kyu Hyong Park, Jordan C Rozum, and Réka Albert. “Automated model refinement using perturbation-observation pairs.” In: *npj Systems Biology and Applications* 11.1 (2025), p. 65.

[Pas22] Samuel Pastva. “Digital Bifurcation Analysis: On the Qualities of Long-term Behaviour in Discrete Systems.” SUPERVISOR : Luboš Brim. Disertační práce. Masarykova univerzita, Fakulta informatiky, Brno, 2022.

[PPS18a] Soumya Paul, Jun Pang, and Cui Su. “On the Full Control of Boolean Networks.” In: *Computational Methods in Systems Biology*. Vol. 11095. Lecture Notes in Computer Science. Springer, 2018, pp. 313–317.

[PPS18b] Soumya Paul, Jun Pang, and Cui Su. “Towards the Existential Control of Boolean Networks: A Preliminary Report.” In: *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*. Springer. 2018, pp. 142–149.

[Pau+18] Soumya Paul, Cui Su, Jun Pang, and Andrzej Mizera. “A Decomposition-Based Approach towards the Control of Boolean Networks.” In: *ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. Association for Computing Machinery, 2018, 11––20.

[Pau+20] Loïc Paulevé, Juri Kolčák, Thomas Chatain, and Stefan Haar. “Reconciling qualitative, abstract, and scalable modeling of biological networks.” In: *Nature communications* 11.1 (2020), p. 4256.

[Pau23] Loïc Paulevé. “Marker and source-marker reprogramming of Most Permissive Boolean networks and ensembles with BoNesis.” In: *Peer Community Journal* (2023).

[Pnu77] Amir Pnueli. “The temporal logic of programs.” In: *18th annual symposium on foundations of computer science (sfcs 1977)*. ieee. 1977, pp. 46–57.

[Por17] Arnaud Poret. “Qualitative modeling of biological networks for therapeutic innovation.” PhD thesis. Université Claude Bernard-Lyon I, 2017.

[PB14] Arnaud Poret and Jean-Pierre Boissel. “An in silico target identification using Boolean network attractors: Avoiding pathological phenotypes.” In: *Comptes Rendus. Biologies* 337.12 (2014), pp. 661–678.

[PG18] Arnaud Poret and Carito Guziolowski. “Therapeutic target discovery using Boolean network attractors: improvements of kali.” In: *Royal Society open science* 5.2 (2018), p. 171852.

[Rai+22] Dhruv Raina et al. “Intermittent ERK oscillations downstream of FGF in mouse embryonic stem cells.” In: *Development* 149.4 (2022), dev199710.

[RW87] Peter J Ramadge and W Murray Wonham. “Supervisory control of a class of discrete event processes.” In: *SIAM journal on control and optimization* 25.1 (1987), pp. 206–230.

[Riv+23] Sara Riva, Jean-Marie Lagniez, Gustavo Magaña López, and Loïc Paulevé. “Tackling universal properties of minimal trap spaces of boolean networks.” In: *International Conference on Computational Methods in Systems Biology*. Springer. 2023, pp. 157–174.

[Rol+15] Andrea Roli, Marco Villani, Roberto Serra, Stefano Benedettini, Carlo Pinciroli, and Mauro Birattari. “Dynamical Properties of Artificially Evolved Boolean Network Robots.” In: *AI*IA 2015 Advances in Artificial Intelligence*. Ed. by Marco Gavanelli, Evelina Lamma, and Fabrizio Riguzzi. Springer, 2015, pp. 45–57.

[RP22] Théo Roncalli and Loïc Paulevé. “Variable-Depth Simulation of Most Permissive Boolean Networks.” In: *International Conference on Computational Methods in Systems Biology*. Springer. 2022, pp. 138–157.

[Roz+22] Jordan C Rozum, Dávid Deritei, Kyu Hyong Park, Jorge Gómez Tejeda Zañudo, and Réka Albert. “pystablemotifs: Python library for attractor identification and control in Boolean networks.” In: *Bioinformatics* 38.5 (2022), pp. 1465–1466.

[Saa+11] Assieh Saadatpour, Rui-Sheng Wang, Aijun Liao, Xin Liu, Thomas P. Loughran, István Albert, and Réka Albert. “Dynamical and Structural Analysis of a T Cell Survival Network Identifies Novel Candidate Therapeutic Targets for Large Granular Lymphocyte Leukemia.” In: *PLOS Computational Biology* 7.11 (2011), pp. 1–15.

[Sah+09] Özgür Sahin, Holger Fröhlich, Christian Löbke, Ulrike Korf, Sara Burmester, Meher Majety, Jens Mattern, Ingo Schupp, Claudine Chaouiya, Denis Thieffry, et al. “Modeling ERBB receptor-regulated G1/S transition to find novel targets for de novo trastuzumab resistance.” In: *BMC Systems Biology* 3.1 (2009), pp. 1–20.

[Sch+20] Julian D. Schwab, Silke D. Kühlwein, Nensi Ikonomi, Michael Kühl, and Hans A. Kestler. "Concepts in Boolean network modeling: What do they all mean?" In: *Computational and Structural Biotechnology Journal* 18 (2020), pp. 571–582.

[Sha38] Claude E Shannon. "A symbolic analysis of relay and switching circuits." In: *Electrical Engineering* 57.12 (1938), pp. 713–723.

[Sha40] Claude Elwood Shannon. "A Symbolic Analysis of Relay and Switching Circuits." Advisor: Frank L. Hitchcock. Includes bibliographical references (leaf 69). M.S. Thesis. Cambridge, MA: Massachusetts Institute of Technology, 1940.

[Shm+02] Ilya Shmulevich, Edward R Dougherty, Seungchan Kim, and Wei Zhang. "Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks." In: *Bioinformatics* 18.2 (2002), pp. 261–274.

[SD09] Amar M. Singh and Stephen Dalton. "The cell cycle and Myc intersect with mechanisms that regulate pluripotency and reprogramming." In: *Cell Stem Cell* 5.2 (2009), pp. 141–149.

[Šmi+20] Eva Šmijáková, Samuel Pastva, David Šafránek, and Luboš Brim. "Parallel Parameter Synthesis for Multi-affine Hybrid Systems from Hybrid CTL Specifications." In: *Computational Methods in Systems Biology*. Springer. 2020, pp. 280–297. ISBN: 978-3-030-60327-4.

[Sor68] HW Sorenson. "Controllability and Observability of Linear, Stochastic, Time Discrete Control Sytems." In: *Advances in Control Systems*. Vol. 6. Elsevier, 1968, pp. 95–158.

[Sta+12] Anthony Stanislaus et al. "Knockdown of PLC-gamma-2 and calmodulin 1 genes sensitizes human cervical adenocarcinoma cells to doxorubicin and paclitaxel." In: *Cancer cell international* 12 (2012), pp. 1–8.

[Su20] Cui Su. "Scalable Control of Asynchronous Boolean Networks." PhD thesis. Université du Luxembourg, 2020.

[SP20a] Cui Su and Jun Pang. "A dynamics-based approach for the target control of Boolean networks." In: *ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*. Association for Computing Machinery, 2020, pp. 1–8.

[SP20b] Cui Su and Jun Pang. “Sequential Temporary and Permanent Control of Boolean Networks.” In: *Computational Methods in Systems Biology*. Vol. 12314. Lecture Notes in Computer Science. Springer. 2020, pp. 234–251.

[SP21] Cui Su and Jun Pang. “CABEAN: A software for the control of asynchronous Boolean networks.” In: *Bioinformatics* 37.6 (2021), pp. 879–881.

[SPP19a] Cui Su, Soumya Paul, and Jun Pang. “Controlling large Boolean networks with temporary and permanent perturbations.” In: *International Symposium on Formal Methods*. Vol. 11800. Lecture Notes in Computer Science. Springer, 2019, pp. 707–724.

[SPP19b] Cui Su, Soumya Paul, and Jun Pang. “Scalable Control of Asynchronous Boolean Networks.” In: *Computational Methods in Systems Biology*. Vol. 11773. Lecture Notes in Computer Science. Springer. 2019, pp. 364–367.

[Su+24] X. Su, H. Zhang, C. Luo, L. Xu, and S. Alghamdi. “Controllability of generalized asynchronous Boolean networks with periodical impulsive control.” In: *Communications in Nonlinear Science and Numerical Simulation* 128 (2024), p. 107653. ISSN: 1007-5704.

[TY06] Kazutoshi Takahashi and Shinya Yamanaka. “Induction of Pluripotent Stem Cells from Mouse Embryonic and Adult Fibroblast Cultures by Defined Factors.” In: *Cell* 126.4 (2006), pp. 663–676.

[Tho73] René Thomas. “Boolean formalization of genetic control circuits.” In: *Journal of theoretical biology* 42.3 (1973), pp. 563–585.

[Tri+24] Giang Trinh, Belaid Benhamou, Samuel Pastva, and Sylvain Soliman. “Scalable enumeration of trap spaces in Boolean networks via answer set programming.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 9. 2024, pp. 10714–10722.

[THB22] Van-Giang Trinh, Kunihiko Hiraishi, and Belaid Benhamou. “Computing attractors of large-scale asynchronous boolean networks using minimal trap spaces.” In: *CM International Conference on Bioinformatics, Computational Biology and Health Informatics*. Association for Computing Machinery, 2022.

[UD+16] SM Minhaz Ud-Dean et al. “Optimal design of gene knockout experiments for gene regulatory network inference.” In: *Bioinformatics* 32.6 (2016), pp. 875–883.

[Vaq+14] José P Vaqué et al. “PLCG1 mutations in cutaneous T-cell lymphomas.” In: *Blood* 123.13 (2014), pp. 2034–2043.

[Ves+25] Vojtěch Veselỳ, Eva Šmijáková, Samuel Pastva, Nikola Beneš, and David Šafránek. "AEON 2025: Robust Control of Partially-Specified Boolean Networks." In: *International Conference on Computational Methods in Systems Biology*. Springer. 2025, pp. 61–68.

[Ves] Vojtěch Veselý. *Design and Implementation of the AEON Tool Frontend*. Bachelor's thesis. SUPERVISOR : David Šafránek.

[Vid14] Santiago Videla. "Reasoning on the response of logical signaling networks with answer set programming." PhD thesis. Université de Rennes; Universität Postdam (Allemagne), 2014.

[Vid+17] Santiago Videla, Julio Saez-Rodriguez, Carito Guziolowski, and Anne Siegel. "caspo: a toolbox for automated reasoning on the response of logical signaling networks families." In: *Bioinformatics* 33.6 (2017), pp. 947–950.

[Vid+15] Santiago Videla et al. "Designing experiments to discriminate families of logic models." In: *Frontiers in bioengineering and biotechnology* 3 (Feb. 2015), p. 131.

[Vil19] Alejandro F Villaverde. "Observability and structural identifiability of nonlinear biological systems." In: *Complexity* 2019.1 (2019), p. 8497093.

[VEB09] Nguyen Xuan Vinh, Julien Epps, and James Bailey. "Information theoretic measures for clusterings comparison: is a correction for chance necessary?" In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 1073–1080.

[Waa+14] Ashley J Waardenberg, Mirana Ramialison, Romaric Bouveret, and Richard P Harvey. "Genetic networks governing heart development." In: *Cold Spring Harbor perspectives in medicine* 4.11 (2014), a013839.

[WSA12] Rui-Sheng Wang, Assieh Saadatpour, and Reka Albert. "Boolean modeling in systems biology: An overview of methodology and applications." In: *Physical biology* 9.5 (2012).

[Wer+07] Marius Wernig, Alexander Meissner, Ruth Foreman, Tobias Brambrink, Manching Ku, Konrad Hochedlinger, Bradley E. Bernstein, and Rudolf Jaenisch. "In vitro reprogramming of fibroblasts into a pluripotent ES-cell-like state." In: *Nature* 448.7151 (2007), pp. 318–324.

[Wie48] Norbert Wiener. *Cybernetics: or Control and Communication in the Animal and the Machine*. 2nd ed. Cambridge, MA: MIT Press, 1948.

[XD07] Yufei Xiao and Edward R Dougherty. “The impact of function perturbations in Boolean networks.” In: *Bioinformatics* 23.10 (2007), pp. 1265–1273.

[Yor+16] Boyan Yordanov et al. “A method to identify and analyze biological programs through automated reasoning.” In: *NPJ systems biology and applications* 2.1 (2016), pp. 1–16.

[You11] Richard A. Young. “Control of the Embryonic Stem Cell State.” In: *Cell* 144.6 (2011), pp. 940–954.

[ZA13] Jorge GT Zañudo and Réka Albert. “An effective network reduction approach to find the dynamical repertoire of discrete dynamic networks.” In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 23.2 (2013).

[ZA15] Jorge G. T. Zañudo and Réka Albert. “Cell Fate Reprogramming by Control of Intracellular Network Dynamics.” In: *PLOS Computational Biology* 11.4 (2015), pp. 1–24.

[ZGC15] Yin Zhao, Bijoy K Ghosh, and Daizhan Cheng. “Control of large-scale Boolean networks via network aggregation.” In: *IEEE Transactions on Neural Networks and Learning Systems* 27.7 (2015), pp. 1527–1536.

[Zhe+13] Desheng Zheng, Guowu Yang, Xiaoyu Li, Zhicai Wang, Feng Liu, and Lei He. “An Efficient Algorithm for Computing Attractors of Synchronous And Asynchronous Boolean Networks.” In: *PLOS ONE* 8.4 (2013), e60593.

[Zhu+16] Peican Zhu, Hamidreza Montazeri Aliabadi, Hasan Uludağ, and Jie Han. “Identification of potential drug targets in cancer signaling pathways using stochastic logical models.” In: *Scientific reports* 6.1 (2016), p. 23078.

[Zhu+15] Yanbei Zhu, Rongze Yang, John McLenithan, Daozhan Yu, Hong Wang, Yaping Wang, Devinder Singh, John Olson, Carole Sztalryd, Dalong Zhu, et al. “Direct conversion of human myoblasts into brown-like adipocytes by engineered super-active PPARγ.” In: *Obesity* 23.5 (2015), pp. 1014–1021.

[Zou13] Yi Ming Zou. “Boolean networks with multiexpressions and parameters.” In: *Transactions on Computational Biology and Bioinformatics* 10 (2013), pp. 584–592.

[Šm19] Eva Šmijáková. “Parallel Parameter Synthesis for Hybrid Systems.” Master’s thesis. Brno: Masarykova univerzita, Fakulta informatiky, 2019.

[Šm21] Eva Šmijáková. *Control of Parametrised Boolean Networks*. Rigorous thesis. SUPERVISOR : Luboš Brim. 2021.