Algebraic theory for discrete models in systems biology

Franziska B. Hinkelmann

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mathematics

Reinhard C. Laubenbacher, Chair
Abdul S. Jarrah
Terry Herdman
Brett Tyler

August 1, 2011
Blacksburg, Virginia

Keywords: Mathematical biology, discrete models, systems biology, polynomial dynamical systems, reverse-engineering, finite fields.

Algebraic theory for discrete models in systems biology

Franziska B. Hinkelmann

ABSTRACT

This dissertation develops algebraic theory for discrete models in systems biology. Many discrete model types can be translated into the framework of *polynomial dynamical systems* (PDS), that is, time- and state-discrete dynamical systems over a finite field where the transition function for each variable is given as a polynomial. This allows for using a range of theoretical and computational tools from computer algebra, which results in a powerful computational engine for model construction, parameter estimation, and analysis methods. Formal definitions and theorems for PDS and the concept of PDS as models of biological systems are introduced in section 1.3.

Constructing a model for given time-course data is a challenging problem. Several methods for reverse-engineering, the process of inferring a model solely based on experimental data, are described briefly in section 1.3. If the underlying dependencies of the model components are known in addition to experimental data, inferring a "good" model amounts to *parameter estimation*. Chapter 2 describes a parameter estimation algorithm that infers a special class of polynomials, so called *nested canalyzing functions*. Models consisting of nested canalyzing functions have been shown to exhibit desirable biological properties, namely robustness and stability. The algorithm is based on the parametrization of nested canalyzing functions. To demonstrate the feasibility of the method, it is applied to the cell-cycle network of budding yeast.

Several discrete model types, such as Boolean networks, logical models, and bounded Petri nets, can be translated into the framework of PDS. Section 3 describes how to translate agent-based models into polynomial dynamical systems.

Chapter 4, 5, and 6 are concerned with analysis of complex models. Section 4 proposes a new method to identify steady states and limit cycles. The method relies on the fact that attractors correspond to the solutions of a system of polynomials over a finite field, a long-studied problem in algebraic geometry which can be efficiently solved by computing Gröbner bases. Section 5 introduces a bit-wise implementation of a Gröbner basis algorithm for Boolean polynomials. This implementation has been incorporated into the core engine of Macaulay2. Chapter 6 discusses bistability for Boolean models formulated as polynomial dynamical systems.

Funding for this work was provided through U.S. Army Research Office Grant Nr. W911NF-09-1-0538, National Science Foundation Grant Nr. CMMI-0908201, National Science Foundation Grant Nr. 0755322, Hatcher Fellowship, and Steeneck Fellowship.

To
my mother.

Acknowledgments

I would like to thank my advisor, Reinhard Laubenbacher, who is a wonderful advisor, mentor, and friend, and my committee members, Terry Herdman, Abdul Jarrah, and Brett Tyler, for their support and patience. I would like to thank my co-authors for permission to include co-authored material: Elizabeth Arnold, Greg Blekherman, Madison Brandon, Bonny Guang, Abdul Jarrah, Reinhard Laubenbacher, Rustin McNeill, David Murrugarra, and Alan Veliz-Cuba.

I thank current and former faculty, staff, and graduate students at the Virginia Bioinformatics Institute and at the Department of Mathematics at Virginia Tech for providing an excellent working environment. I'm especially thankful to Greg Blekherman, Abdul Jarrah, Terry Herdman, Peter Haskell, Shernita Lee, Matt Oremland, Betsy Williams, Elena Dimitrova, Brandy Stigler, John Burns, Gene Cliff, Eileen Shugart, Julia Chifman, and Ed Green.

I want to thank Michael Stillman from Cornell University for his help and many hours of joint programming.

I'm thankful to my friends because of whom made my time in Blacksburg was so enjoyable: Julia Bartens, Sven Dorosz, Kevin Pond, Kasie Farlow, Greg Blekherman, Maria Laubenbacher, Thierry Platini, Sayak Mukherjee, Jeremy Archuleta, Christian Wernz, Thomas Pramann and Gabriele Enea.

Last but not least, I would like to thank Stefan Kühnlein and Rudolf Scherer from the Karlsruhe Institute of Technology, who encouraged me to study mathematics and to apply at Virginia Tech.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Systems biology aims to explain how a biological system functions by investigating the interactions of its individual components from a systems perspective. Modeling is a vital tool as it helps to elucidate the underlying mechanisms of the system. Model types include, but are not limited to, continuous, discrete, and statistical models. Discrete models are frequently used in systems biology to reveal qualitative information about systems where detailed quantitative information is not needed or is unavailable [85, 39, 12, 4, 73]. Model types include (probabilistic) Boolean networks, logical networks, Petri nets, cellular automata, and agent-based models [92, 83, 88, 70, 9, 59], to name the most common types.

The state space of discrete models grows exponentially in the number of variables, e.g., the state space of a Boolean model with 60 variables consists of approximately 10^{18} states. This *combinatorial explosion* often prohibits the use of complex models, as it is computationally inefficient to work with them. Many discrete model types can be translated into the framework of *polynomial dynamical systems*, that is, time- and state-discrete dynamical systems over a finite field where the transition function for each variable is given as a polynomial. This allows for using a range of theoretical and computational tools from computer algebra, which results in a powerful computational engine for model construction, parameter estimation, and analysis methods.

1.1 Overview of main results in this dissertation

This dissertation develops mathematical theory for *polynomial dynamical systems* (PDS) as framework for discrete models in systems biology. Formal definitions and theorems for PDS and the concept of PDS as models of biological systems are introduced in section 1.3.

Constructing a model for given time-course data is a challenging problem. Several methods for reverse-engineering, the process of inferring a model solely based on experimental data,

are described briefly in section 1.3. If the underlying dependencies of the model components are known in addition to experimental data, inferring a "good" model amounts to *parameter estimation*. Chapter 2 describes a parameter estimation algorithm that infers a special class of polynomials, so called *nested canalyzing functions*. Models consisting of nested canalyzing functions have been shown to exhibit desirable biological properties, namely robustness and stability. The algorithm is based on the parametrization of nested canalyzing functions. To demonstrate the feasibility of the method, it is applied to the cell-cycle network of budding yeast.

Several discrete model types, such as Boolean networks, logical models, and bounded Petri nets, can be translated into the framework of PDS [98]. Section 3 describes how to translate agent-based models into polynomial dynamical systems.

Chapters 4, 5, and 6 are concerned with analysis of complex PDS. Section 4 proposes a new method to identify steady states and limit cycles. The method relies on the fact that attractors correspond to the solutions of a system of polynomials over a finite field, a long-studied problem in algebraic geometry which can be efficiently solved by computing Gröbner bases. Section 5 introduces a bit-wise implementation of a Gröbner basis algorithm for Boolean polynomials. This implementation has been incorporated into the core engine of Macaulay2 [33]. Chapter 6 discusses bistability for Boolean models formulated as polynomial dynamical systems.

Finally, Chapter 7 contains potential extensions to the framework, which, combined, will result in a powerful computational tool for discrete modeling.

1.2 Manuscripts contained in this dissertation

Most material in this dissertation has been published or is accepted for publication.

1.3 G. Mullen and D. Panario, editors. Handbook on Finite Fields. CRC Press, 2011. Chapter *Finite Fields in Systems Biology*, **F. Hinkelmann** and R. Laubenbacher

2 **F. Hinkelmann** and A. S. Jarrah. *Inferring biologically relevant models: Nested canalyzing functions*. Under review, 2010.

3 **F. Hinkelmann**, D. Murrugarra, A. Jarrah, and R. Laubenbacher. *A mathematical framework for agent based models of complex biological networks*. Bulletin of Mathematical Biology, 73:15831602, 2011.

4 **F. Hinkelmann**, M. Brandon, B. Guang, R. McNeill, A. Veliz-Cuba, G. Blekherman, and R. Laubenbacher. *ADAM: Analysis of Discrete Models of Biological Systems Using Computer Algebra*, BMC Bioinformatics, 12(1):295, 2011.

5 **F. Hinkelmann** and E. Arnold. *Fast Gröbner basis computation for Boolean polynomials*. Journal of Software for Algebra and Geometry: Macaulay2, under revision, 2011.

6 **F. Hinkelmann** and R. Laubenbacher. *Boolean models of bistable biological systems*. Discrete and Continuous Dynamical Systems. Series S, 4(6):14431456, 2011.

Not contained in this dissertation.

- E. Dimitrova, L. D. Garcia-Puente, **F. Hinkelmann**, A. S. Jarrah, R. Laubenbacher, B. Stigler, M. Stillman, and P. Vera-Licona. *Parameter estimation for boolean models of biological networks*. Theoretical Computer Science, 412(26):2816 2826, 2011. Foundations of Formal Reconstruction of Biochemical Networks.

1.3 Polynomial dynamical systems, definitions and examples

The material in this chapter is based on the chapter "Finite Fields in Systems Biology" in the *Handbook of Finite Fields* [65], the chapter is authored jointly with Reinhard Laubenbacher.

1.3.1 Polynomial dynamical systems as framework for discrete models in systems biology

The goal of molecular systems biology is to understand the functionality of biological systems as a whole by studying interactions among the components, e.g., genes, proteins, and metabolites. Modeling is a vital tool in achieving this goal. It allows the simulation of different types of interactions within the system, leading to testable hypotheses, which can then be experimentally validated. This process leads to a better understanding of the underlying biological system.

Discrete models are increasingly used in systems biology [4, 85, 102, 84]. They can reveal valuable insight about the qualitative behavior of a system when it is infeasible to estimate enough parameters accurately to build a quantitative model. Many discrete models can be formulated as polynomial dynamical systems, that is, state and time discrete dynamical systems described by polynomial functions over a finite field. This provides access to the algorithmic theory of computational algebra and the theoretical foundation of algebraic geometry, which helps with all aspects of the modeling process. Polynomial dynamical systems provide a unifying framework for many discrete modeling types. The algebraic framework allows for efficient construction and analysis of discrete models.

1.3.2 Polynomial dynamical systems

Definitions and theorems can be found in [98].

Definition 1.3.1. A *Polynomial Dynamical System (PDS)* is a time-discrete dynamical system $f = (f_1, \ldots, f_n) : \mathbb{F}^n \to \mathbb{F}^n$ where each coordinate function f_i is a function of the n variables $x_1, \ldots, x_n$, each of which takes on values in a finite field $\mathbb{F}$, and each f_i is a polynomial.

Two directed graphs are usually assigned to each such system.

Definition 1.3.2. The *dependency graph* (or *wiring diagram*) $\mathcal{D}(f)$ of f has n vertices $1, \ldots, n$, corresponding to the variables $x_1, \ldots, x_n$ of f. There is a directed edge $i \to j$ if x_i appears in the function f_j. That is, $\mathcal{D}(f)$ encodes the variable dependencies in f.

Definition 1.3.3. The dynamics of f is encoded by its *phase space* (or *state space*), denoted by $\mathcal{S}(f)$. It is the directed graph with vertex set $\mathbb{F}^n$ and a directed edge from u to v if $f(u) = v$.

Definition 1.3.4. A *component* of the phase space $\mathcal{S}(f)$ consists of a limit cycle and all orbits of f that contain it. Hence, the phase space is a disjoint union of components.

Definition 1.3.5. For each $u \in \mathbb{F}^n$, the sequence $\{u, f(u), f^2(u), \ldots\}$ is called the *orbit* of u. If $u = f^t(u)$ and t is the smallest such number, the sequence $\{u, f(u), f^2(u), \ldots, f^{t-1}(u)\}$ is called a *limit cycle* of length t, and u is called a *periodic point* of period t. Since $\mathbb{F}^n$ is finite, every orbit must include a limit cycle.

Definition 1.3.6. The point u is called a *fixed point* (or *steady state*) if $f(u) = u$.

Theorem 1.3.7. Limit Cycle Analysis: *For a polynomial dynamical system* $f = (f_1, \ldots, f_n) : \mathbb{F}^n \to \mathbb{F}^n$, *states in a limit cycle of length* t *are elements of the algebraic variety* $V(f_1^t - x_1, \ldots, f_n^t - x_n)$, *defined by the polynomials* $f_1^t - x_1, \ldots, f_n^t - x_n$, *but not of the variety* $V(f_1^s - x_1, \ldots, f_n^s - x_n)$ *for any* $s < t$. *In particular, fixed points are the points in the variety* $V(f_1 - x_1, \ldots, f_n - x_n)$.

Definition 1.3.8. A polynomial dynamical system can be iterated using different *update schedules*:

- synchronous: all variables are updated at the same time;
- sequential: variables are updated sequentially according to the order specified by an update schedule. A sequential system with a given update schedule can be translated into a synchronous system with the same dynamics;
- asynchronous: at every time step, one variable is chosen according to a probability distribution (usually uniform) to be updated. Asynchronous systems are non-deterministic;
- delays.

Remark 1.3.9. Synchronous and asynchronous systems have the same fixed points.

For visualization and analysis of PDS, the web-based tool ADAM is available, see section 1.3.5 [45]. PDS can be used to represent dynamic biological systems.

Example 1.3.10. The *lac(tose)* operon is a functional unit of three genes, LacZ, LacY, and LacA, transcribed together, responsible for the metabolism of lactose in the absence of glucose in bacteria. In the presence of lactose this genetic machinery is disabled by a repressor protein. The genes in the *lac* operon encode several proteins involved in this process. Lactose permease transports extracellular lactose into the cell, where the protein β-galactosidase breaks the lactose down into glucose, galactose and allolactose. The allolactose binds to

the repressor protein and deactivates it, which results in the transcription of the three *lac* genes, resulting in the production of permease and β-galactosidase. This positive feedback loop allows for a rapid increase of lactose when needed. The result is a bistable dynamical system. This process can be modeled by a polynomial dynamical system. Each variable can take on two states: 0 denotes the absence (or low concentration) of a substrate or the inactive state of a variable, and 1 denotes presence or activity. As there are only two states, the system is over the finite field $\mathbb{F}_2$. Permease (x_P) transports external lactose (x_{eL}) inside the cell, and the update function for intracellular lactose (x_L) is $x_L(t+1) = x_{eL}(t)$ AND $x_P(t)$, or as a polynomial over $\mathbb{F}_2$, $f_L = x_{eL}x_P$. Using x_B for β-galactosidase, x_M for mRNA, x_A for allolactose, the functionality of the *lac* operon can be modeled by the polynomial dynamical system $f = (f_L, f_A, f_M, f_P, f_B) : \mathbb{F}_2^5 \to \mathbb{F}_2^5$:

$$\begin{aligned} f_L &= x_{eL}x_P, \\ f_A &= x_L x_B, \\ f_M &= x_A, \\ f_P &= x_M, \\ f_B &= x_M. \end{aligned}$$

Figure 1.1 denotes the dependency graph and phase space of the above model. [47]

1.3.3 Discrete model types and their translation into PDS

Continuous models of biological systems, such as ordinary or partial differential equations models, rely on exact rate parameters, for which it is oftentimes impossible to obtain exact measurements. When not enough information is available to build a quantitative model, discrete models can give valuable insight about the qualitative behavior of the system. Such models are state- and time-discrete. For example, in the most simple case, one distinguishes only between two states, ON and OFF, or active and inactive, present and absent, etc.

Model types include (probabilistic) Boolean networks, logical networks, Petri nets, cellular automata, and agent-based (individual-based) models, to name the most commonly found ones [92, 83, 88, 70, 9, 59]. All these model types can be translated into PDS [48, 98].

Boolean network models

In a *Boolean network model* every variable is either ON or OFF, and the state of each variable at time $t+1$ is determined by a Boolean expression that involves some or all of the variables at time t. Boolean models were first introduced in 1969 by Kauffman for gene regulatory networks, in which each gene (variable) is either expressed (ON) or not expressed (OFF) at every time step [56].

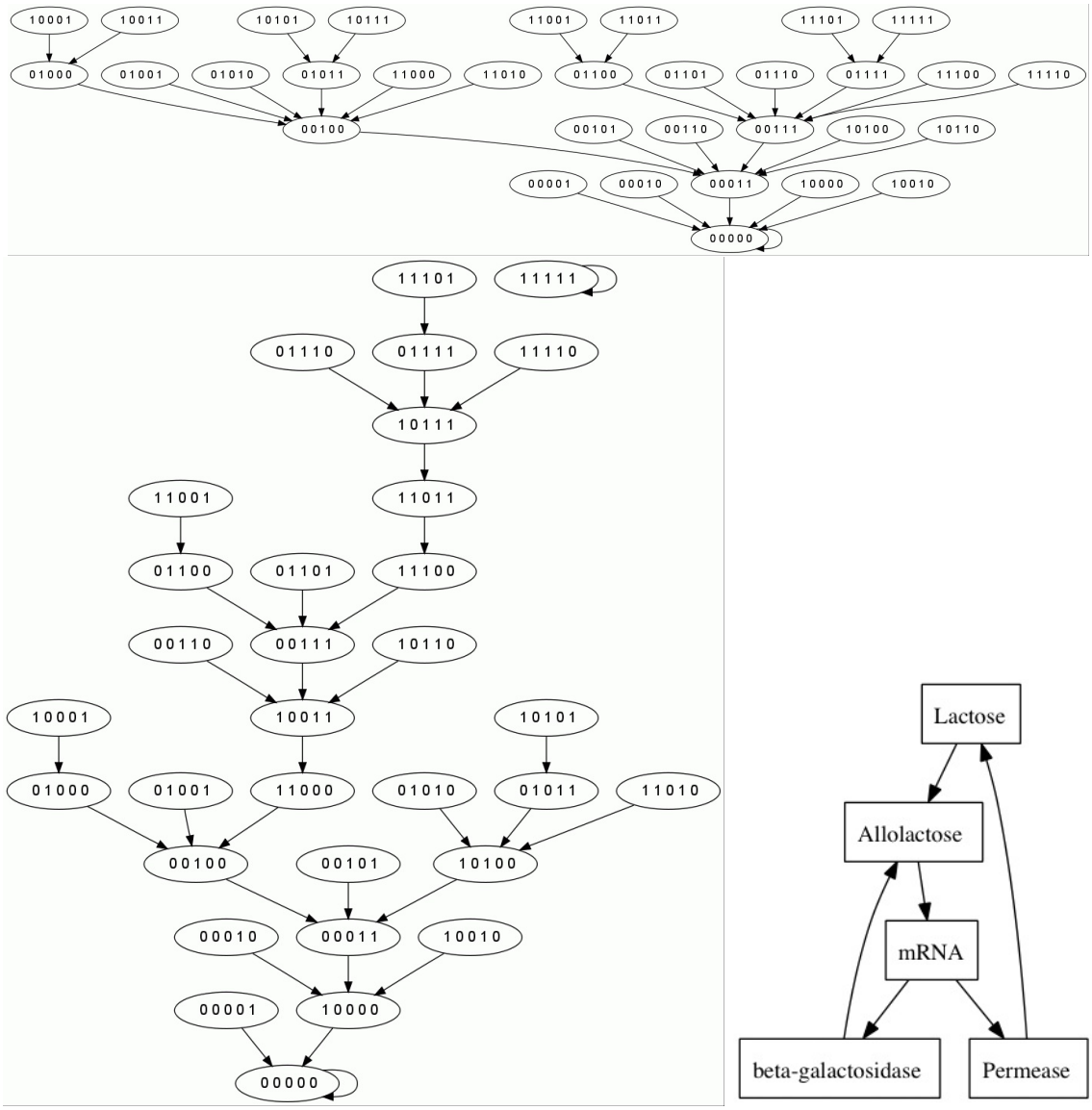


Figure 1.1: PDS for *lac* operon: state space in the absence (top) and presence (bottom left) of glucose, and dependency graph (bottom right). Each 5-tuple represents the states of lactose, allolactose, mRNA, permease, and β-galactosidase, (L, A, M, P, B).

Algorithm 1.3.1. *For* Boolean network models*, where there are two states (TRUE and FALSE),* $\mathbb{F} = \mathbb{F}_2$*,* 0 *denotes FALSE and* 1 *TRUE. Table 1.1 lists the Boolean expressions and the corresponding polynomials. All Boolean expressions can be translated to polynomials using this correspondence.*

Table 1.1: Correspondence of Boolean expressions and polynomials over $\mathbb{F}_2$

Boolean expression	polynomial
X	x
NOT X	$x+1$
X AND Y	xy
X OR Y	$xy+x+y$

Logical models

Logical models are a generalization of Boolean models, in which variables can take on more than two states, e.g., to represent the three states low, medium, and high concentration of a substrate. The rules governing the temporal evolution are switch-like logical rules for the different states of each variable. Updates in logical models are specified via parameters rather than by a (Boolean) expression.

Definition 1.3.11. A *logical model* is a triple $(\mathcal{V}, \mathcal{E}, \mathcal{K})$, where:

1. $V = \{v_1, \ldots, v_n\}$ is the set of vertices or nodes. Each v_i has a maximum expression level, m_i. The set $S = [0, m_1] \times \ldots \times [0, m_n]$ (Cartesian product) is called the state space of the logical model and its elements are called states.

2. $\mathcal{E}$ is the set of arcs. The elements of $\mathcal{E}$ have the form (v_i, v_j, θ), where $1 \leq \theta \leq m_i$ is called a threshold for (v_i, v_j). Let $\mathcal{I}(j) = \{v : (v, v_j, \theta) \in \mathcal{E}\}$, called input of v_j, be the set of vertices that have an edge ending in v_j. Notice that an arc (v_i, v_j) is allowed to have multiplicity corresponding to different thresholds; the number of thresholds is denoted by $m_{i,j}$. The thresholds are indexed in increasing order, $1 \leq \theta_{i,j,1} < \cdots < \theta_{i,j,m_{i,j}} \leq m_i$; $\theta_{i,j,k}$ is the k-th threshold for the arc (v_i, v_j). By convention, we define $\theta_{i,j,m_{i,j}+1} = m_i + 1$ (actually, $\theta_{i,j,m_{i,j}+1}$ can be defined as any number greater than m_i) and $\theta_{i,j,0} = 0$. Let $x = (x_1, \ldots, x_n)$ be a state; we say that the k-th interaction for input v_i of v_j is active if $\theta_{i,j,k} \leq x_i < \theta_{i,j,k+1}$; we denote this by $\Theta^{i,j}(x_i) = k$.

3. $\mathcal{K} = \{K_i : \prod_{v_j \in \mathcal{I}(i)} [0, m_{j,i}] \to [0, m_i], i = 1, \ldots, n\}$ is the set of parameters.

Example 1.3.12. *Logical Model of Lambda Phage* [77]. Lambda phage is a virus that injects its DNA into a bacterial host. Once injected, it enters either a lytic cycle or a lysogenic cycle. In the lytic cycle, its DNA is replicated, the host cell lyses, and new viruses are

released. In the lysogenic cycle, the virus' DNA is copied into the hosts DNA where it remains without any apparent harm to the host. A number of bacterial and viral genes take part in the decision process between lysis and lysogenisation. The core of the regulatory network that controls the life cycle consists of two regulatory genes, cI and cro. Lysogeny is maintained if cI proteins dominate, the lytic cycle if cro proteins dominate. CI inhibits cro, and vice versa. At high concentrations, cro downregulates its own production, see Figure 1.2. This regulatory network can be encoded in the following logical model: $V = \{\mathrm{c1}, \mathrm{cro}\}$, $\mathcal{E} =$

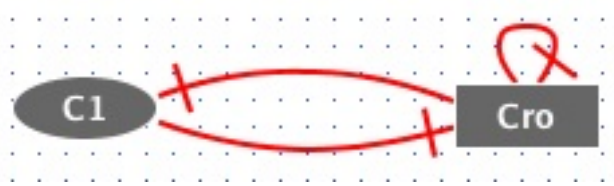


Figure 1.2: Logical Model of Lambda Phage, blunt ended arrows indicate an inhibitory effect.

$\{(\mathrm{c1}, \mathrm{cro}, 1), (\mathrm{cro}, \mathrm{c1}, 1), (\mathrm{cro}, \mathrm{cro}, 2)\}$, and $\mathcal{K} = \{K_{\mathrm{c1}}, K_{\mathrm{cro}}\}$, where $K_{\mathrm{c1}} : \{0,1,2\} \to \{0,1\}$ is defined as $K_{\mathrm{c1}}(0) = 1$, $K_{\mathrm{c1}}(1) = K_{\mathrm{c1}}(2) = 0$ and $K_{\mathrm{cro}} : \{0,1\} \times \{0,1,2\} \to \{0,1,2\}$ as $K_{\mathrm{cro}}(0,2) = K_{\mathrm{cro}}(1,2) = 1$, $K_{\mathrm{cro}}(1,0) = K_{\mathrm{cro}}(1,1) = 0$, $K_{\mathrm{cro}}(0,0) = 1$, and $K_{\mathrm{cro}}(0,1) = 2$.

Algorithm 1.3.2. Logical models *are translated into a PDS by the following algorithm. Let* $(\mathcal{V}, \mathcal{E}, \mathcal{K})$ *be a logical model as in Definition 1.3.11. Choose a prime number* p *such that* $p \geq m_i+1$ *for all* $1 \leq i \leq n$ *(*$[0, m_i]$ *has* m_i+1 *elements), and let* $\mathbb{F} = \mathbb{F}_p = \{0, 1, \ldots, p-1\}$ *be the field with* p *elements. Note that we may consider the set* S *to be a subset of* $\mathbb{F}^n$*. Consider a vertex* v_i *and let* g_i *(or* f_i *if we do not require continuity) be its coordinate function. Our goal is to represent* g_i *as a polynomial in terms of its inputs, say* $x_{i_1}, \ldots, x_{i_r}$ *. That is, we need a polynomial function defined on* $\mathbb{F}^r$ *with values in* $\mathbb{F}$*. Denote* $a \wedge b = min\{a, b\}$*, using the natural order on the set* $\mathbb{F}$*, viewed as integers. To extend the domain of* g *from* $\prod_{v_j \in \mathcal{I}(i)} [0, m_{j,i}]$ *to* $\mathbb{F}^r$*, we define* $g(x_{i_1}, \ldots, x_{i_r}) = g_i(x_{i_1} \wedge m_{i_1}, \ldots, x_{i_r} \wedge m_{i_r})$ *for* $(x_{i_1}, \ldots, x_{i_r}) \in \mathbb{F}^r$*. The polynomial form of* $g_i : \mathbb{F}^r \to \mathbb{F}$ *is then*

$$g_i(x) = \sum_{(c_{i_1}, \ldots, c_{i_r}) \in \mathbb{F}^r} g_i(c_{i_1}, \ldots, c_{i_r}) \prod_{v_j \in \mathcal{I}(i)} (1 - (x_j - c_j)^{p-1}),$$

where the right-hand side is computed modulo p*.*

Example 1.3.13. The logical model of the lambda phage presented in example 1.3.12 corresponds to this PDS over $\mathbb{F}_3$: $\mathrm{c1} = x_1$ and $\mathrm{cro} = x_2$, and the polynomials are

$$f_1 = -x_2^2 + 1$$
$$f_2 = -x_1^2 x_2^2 + x_1^2 x_2 + x_1^2 + x_2^2 - x_2 - 1.$$

The logical model can be converted manually or with the software package ADAM [45]. Instead of analyzing the logical model, the corresponding PDS can be analyzed for its dynamic features.

Petri nets and agent-based models

Petri nets are bipartite graphs, consisting of places and transitions. Places can be marked with tokens, usually representing concentration level or number of molecules present. Transitions fire in a non-deterministic way and move tokens between places. Petri nets have been used extensively to model chemical reaction networks, where places represent species and transitions interactions. Analysis of the Petri net can reveal which species are persistent, i.e., which species can become extinct if all species are present at the initial time. For the translation algorithm of Petri nets into polynomial dynamical systems, see [98].

Agent-based models (ABM) (or *individual-based models*) are computational models consisting of individual agents, each agent having a set of rules that defines how it interacts with other agents and the environment. Simulation is used to assess the evolution of the system as a whole. Sophisticated agent-based models have been published that simulate biological systems including tumor growth and the immune system [25, 5, 63]. For conversion of agent-based models into polynomial dynamical systems, see [48].

1.3.4 Reverse engineering and parameter estimation

A central problem in systems biology is the construction of models based on system-level experimental data and biological input. When the wiring diagram is known but not the combinatorial effect of the different regulatory inputs, the process of identifying models that fit the data is analogous to parameter estimation for continuous models: estimate a function that fits the experimental data and satisfies some optimality criterion.

Typically, the set of possible models is very large. Tools from computer algebra allow the identification of all models that fit a given experimental data set, and furthermore allow one to identify those models that satisfy a given optimality criterion [19, 60]. Section 1.3.4 introduces an algorithm that identifies possible wiring diagrams, with the optimality criterion that every variable is affected by a minimal number of other variables, and sections 1.3.4 and 1.3.6 introduce parameter estimation algorithms.

The minimal-sets algorithm

The *minimal-sets algorithm* constructs the set of all "minimal" wiring diagrams such that a model that fits the experimental data exists for each wiring diagram [53]. Given m observations stating that the inputs t_i result in the state s_i for a variable, i.e., $(s_1, t_1), \ldots, (s_m, t_m)$, where $s_i \in \mathbb{F}^n$ and $t_i \in \mathbb{F}$, the minimal-sets algorithm identifies all inclusion minimal sets $S \subseteq \{1, \ldots, n\}$ such that there exists a polynomial function $f \in \mathbb{F}[\{x_j | j \in S\}]$ with $f(s_i) = t_i$, for all $i \in \{1, \ldots, m\}$. The algorithm is based on the fact that one can define a simplicial complex Δ associated to the experimental data, such that the face ideal for the Alexander

dual of Δ is a square-free monomial ideal M, and the generating sets for the minimal primes in the primary decomposition of the ideal M are exactly the desired minimal sets.

Parameter estimation using the Gröbner fan of an ideal

Typically, there are many models that fit experimental data, even when restricting the model space to minimal models. The *minimal sampling algorithm* is used to sample the subspace of minimal models; it returns a set of weighted functions per node, assigning a higher weight to functions that are candidates for several monomial orders [20]. The algorithm is based on the Gröbner fan of an ideal.

1.3.5 Software for biologists and computer algebra software

The methods described in this section heavily rely on computer algebra systems, e.g., Macaulay2 [33]. A major advantage of translating other discrete modeling types into polynomial systems is that efficient implementations of algorithms such as Gröbner basis calculations or primary decomposition are already implemented, and can be used independently of the underlying model type. On the other hand, an algorithm implemented to analyze a Petri net can not be re-used to analyze a logical model. As many biologists are not familiar with computer algebra systems and the mathematical theory, software packages have been developed that allow the construction and analysis of discrete models using methods described in this section, without requiring understanding of the underlying mathematics [45, 18].

1.3.6 Specific polynomial dynamical systems

The next sections describe specific classes of polynomial dynamical systems, and theorems that relate the structure of the PDS to its dynamics.

Nested canalyzing functions

Certain polynomial functions are very unlikely to represent an interaction in a biological system. For example, in a Boolean system, $x + y$, i.e., the exclusive OR, is unlikely to represent an actual biological process. In addition, there are classes of functions that are biologically more relevant than other functions. One such class consists of so-called nested canalyzing functions, named after the genetic concept of canalization, identified by the geneticist Waddington in the 1940's. Networks consisting of nested canalyzing functions are robust and stable [55, 99]. The following results can be found in [50, 52]

Definition 1.3.14. A Boolean function $f(x_1, \ldots, x_n)$ is *canalyzing* if there exists an index i and a Boolean value a for x_i such that $f(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n) = b$ is constant. That

is, the variable x_i, when given the *canalyzing value* a, determines the value of the function f, regardless of the other inputs. The output value b is called the *canalyzed value*.

Definition 1.3.15. Let f be a Boolean function in n variables.

- Let σ be a permutation on $\{1,\dots,n\}$. The function f is a *nested canalyzing function* (NCF) in the variable order $x_{\sigma(1)},\dots,x_{\sigma(n)}$ with canalyzing input values $a_1,\dots,a_n$ and canalyzed output values $b_1,\dots,b_n$, respectively, if it can be represented in the form

$$f(x_1,x_2,\dots,x_n)=\begin{cases} b_1 & \text{if } x_{\sigma(1)}=a_1,\\ b_2 & \text{if } x_{\sigma(1)}\neq a_1 \text{ and } x_{\sigma(2)}=a_2,\\ b_3 & \text{if } x_{\sigma(1)}\neq a_1 \text{ and } x_{\sigma(2)}\neq a_2 \text{ and } x_{\sigma(3)}=a_3,\\ \vdots & \quad\vdots\\ b_n & \text{if } x_{\sigma(1)}\neq a_1 \text{ and } \cdots \text{ and } x_{\sigma(n-1)}\neq a_{n-1} \text{ and } x_{\sigma(n)}=a_n,\\ b_n+1 & \text{if } x_{\sigma(1)}\neq a_1 \text{ and } \cdots \text{ and } x_{\sigma(n)}\neq a_n. \end{cases}$$

- The function f is nested canalyzing if f is nested canalyzing in the variable order $x_{\sigma(1)},\dots,x_{\sigma(n)}$ for some permutation σ.

Remark 1.3.16. Any Boolean function in n variables is a map $f:\{0,1\}^n\to\{0,1\}$. Denote the set of all such maps by B_n. For any Boolean function $f\in B_n$, there is a unique polynomial $g\in\mathbb{F}_2[x_1,\dots,x_n]$ such that $g(a_1,\dots,a_n)=f(a_1,\dots,a_n)$ for all $(a_1,\dots,a_n)\in\mathbb{F}_2^n$ and such that the degree of each variable appearing in g is equal to 1. Namely,

$$g(x_1,\dots,x_n)=\sum_{(a_1,\dots,a_n)\in\mathbb{F}_2^n} f(a_1,\dots,a_n)\prod_{i=1}^{n}(1-(x_i-a_i)).$$

Definition 1.3.17. Let S be a a non-empty set whose highest element is r_S. The *completion* of S, which is denoted by $[r_S]$, is the set $[r_S]:=\{1,2,\dots,r_S\}$. For $S=\emptyset$, let $[r_\emptyset]:=\emptyset$.

Theorem 1.3.18. *Let f be a Boolean polynomial in n variables, given by*

$$f(x_1,x_2,\dots,x_n)=\sum_{S\subseteq[n]} c_S\prod_{i\in S}x_i. \tag{1.1}$$

The polynomial f is a nested canalyzing function in the order $x_1,x_2,\dots,x_n$ if and only if $c_{[n]}=1$, and for any subset $S\subseteq[n]$,

$$c_S=c_{[r_S]}\prod_{i\in[r_S]\setminus S}c_{[n]\setminus\{i\}}.$$

Corollary 1.3.19. *The set of points in $\mathbb{F}_2^{2^n}$ corresponding to coefficient vectors of nested canalyzing functions in the variable order $x_1, \ldots, x_n$, denoted by V_{id}^{ncf}, is given by*

$$V_{id}^{ncf} = \Big\{ (c_\emptyset, \ldots, c_{[n]}) \in \mathbb{F}_2^{2^n} : c_{[n]} = 1, \\ c_S = c_{[r_S]} \prod_{i \in [r_S] \setminus S} c_{[n]\setminus\{i\}}, \text{ for } S \subseteq [n] \Big\}.$$

Definition 1.3.20. Let σ be a permutation on the elements of the set $[n]$. We define a new order relation $<_\sigma$ on the elements of $[n]$ as follows: $i <_\sigma j$ if and only if $\sigma^{-1}(i) < \sigma^{-1}(j)$.

Let S be a nonempty subset of $[n]$, say $S = \{i_1, \ldots, i_t\}$. Let $r_S^\sigma := \max\{\sigma^{-1}(i_1), \ldots, \sigma^{-1}(i_t)\}$. The *completion of S with respect to the permutation σ*, denoted by $[r_S^\sigma]_\sigma$ is the set $[r_S^\sigma]_\sigma := \{\sigma(1), \ldots, \sigma(r_S^\sigma)\}$.

Corollary 1.3.21. *Let $f \in R$ and let σ be a permutation of the set $[n]$. The polynomial f is a nested canalyzing function in the order $x_{\sigma(1)}, \ldots, x_{\sigma(n)}$, with input values $a_{\sigma(i)}$ and corresponding output values $b_{\sigma(i)}$, $1 \leq i \leq n$, if and only if $c[n] = 1$ and, for any subset $S \in [n]$,*

$$c_S = c_{[r_S^\sigma]_\sigma} \prod_{w \in [r_S^\sigma]_\sigma \setminus S} c_{[n]\setminus\{w\}}.$$

Corollary 1.3.22. *Let σ be a permutation on $[n]$. The set of points in $\mathbb{F}_2^{2^n}$ corresponding to nested canalyzing functions in the variable order $x_{\sigma(1)}, \ldots, x_{\sigma(n)}$, denoted by V_{id}^{σ}, is defined by*

$$V_{id}^{\sigma} = \Big\{ (c_\emptyset, \ldots, c_{[n]}) \in \mathbb{F}_2^{2^n} : c_{[n]} = 1, \\ c_S = c_{[r_S^\sigma]_\sigma} \prod_{w \in [r_S^\sigma]_\sigma \setminus S} c_{[n]\setminus\{w\}}, \text{ for } S \subseteq [n] \Big\}.$$

Corollary 1.3.23. *Let $f \in R$ and let σ be a permutation of the elements of the set $[n]$. If f is a nested canalyzing function in the order $x_{\sigma(1)}, \ldots, x_{\sigma(n)}$, with input values a_i and corresponding output values b_i, $1 \leq i \leq n$, then*

$$\begin{aligned} a_i &= c_{[n]\setminus\{\sigma(i)\}}, \text{ for } 1 \leq i \leq n-1 \\ b_1 &= c_\emptyset + c_{\sigma(1)} c_{[n]\setminus\{\sigma(1)\}}, \\ b_{i+1} - b_i &= c_{[i+1]_\sigma} c_{[n]\setminus\{\sigma(i+1)\}} + c_{[i]_\sigma}, \text{ for } 1 \leq i < n-1 \text{ and} \\ b_n - a_n &= b_{n-1} + c_{[n-1]_\sigma}. \end{aligned}$$

Thus, the family of nested canalyzing polynomials in a given number of variables can be described as an algebraic variety defined by a collection of binomials. This description has several useful implications. Results from this section can be found in [52].

Parameter estimation resulting in nested canalyzing functions

For given time course data and a wiring diagram, one can identify all nested canalyzing functions (1.3.6) that fit this information. Nested canalyzing functions can be parameterized by an ideal, and intersecting the variety of this ideal with the variety of the ideal of all functions that fit the data results in the desired set of models [46].

Linear polynomial dynamical systems

For linear systems, i.e., all polynomials are linear functions, the dynamics of a system can be determined completely from the structure of the polynomials. [97, 24]

Conjunctive/disjunctive networks

Conjunctive (respectively disjunctive) Boolean network models consist of functions constructed using only the AND (respectively OR) operator. For conjunctive or disjunctive networks with strongly connected dependency graph, the cycle structure is completely determined by a formula that depends on the *loop number*, the greatest common divisor of the lengths of the dependency graph's simple (no repeated vertices) directed cycles. For general Boolean conjunctive or disjunctive networks, an upper and lower bound for the number of cycles of a particular length can be calculated [51].

Chapter 2

Inferring biologically relevant models: nested canalyzing functions

This work is authored jointly with Abdul S. Jarrah. FH's contribution is the application and implementation of the algorithm. Hinkelmann was supported by a grant from the U.S. Army Research Office and Jarrah was supported partially by the NSF Grant DMS-0908201. This manuscript is currently under review.

Abstract

Inferring dynamic biochemical networks is one of the main challenges in systems biology. Given experimental data, the objective is to identify the rules of interaction among the different entities of the network. However, the number of possible models fitting the available data is huge and identifying a biologically relevant model is of great interest. Nested canalyzing functions, where variables in a given order dominate the function, have recently been proposed as a framework for modeling gene regulatory networks. Previously we described this class of functions as an algebraic toric variety. In this paper, we present an algorithm that identifies all nested canalyzing models that fit the given data. We demonstrate our methods using a well-known Boolean model of the cell cycle in budding yeast.

2.1 Introduction

Inferring dynamic biochemical networks is one of the main challenges in systems biology. Many mathematical and statistical methods, within different frameworks, have been developed to address this problem, see [91] for a review of some of these methods. Starting from experimental data and known biological properties only, the idea is to infer a "most likely"

model that could be used to generate the experimental data. Here the model could have two parts. The first one is the static network which is a directed graph showing the influence relationships among the components of the network, where an edge from node y to node x implies that changes in the concentration of y could change the concentration of x. The other part of the model is the dynamics of the network, which describes how exactly the concentration of x is affected by that of y. Due to the fact that biological networks are not well-understood and the available data about the network is usually limited, many models end up fitting the available information and the criteria for choosing a particular model are usually not biologically motivated but rather a consequence of the modeling framework.

A framework that has long been used for modeling gene regulatory networks is *time-discrete, finite-space dynamical systems*. This includes Boolean networks [56], Logical models [94], Petri nets [92], and algebraic models [60]. The latter is a straightforward generalization of Boolean networks to multistate systems. Furthermore, in [98], it was shown that logical models as well as Petri nets could be viewed and analyzed as algebraic models. The inference methods we develop here are within the algebraic models framework. To be self-contained, we briefly describe this framework and state some of the known results that we need in this paper, see [60, 52, 50, 98] for more details. Throughout this paper, we will be talking about gene regulatory networks, however, the methods apply for biochemical networks in general.

Suppose that the gene regulatory network that we want to infer has n genes and that we have a set D of r state transition pairs $(\mathbf{s}_j, \mathbf{t}_j)$, $j = 1, ..., r$. The input $\mathbf{s}_j$ and the output $\mathbf{t}_j$ are n-tuples of 0 and 1 encoding the state of genes $x_1, \ldots, x_n$. Real time data points are not Boolean but could be discretized (and in particular, could be made Boolean) using different methods [21]. The goal is to find a model

$$f = (f_1, f_2, \ldots, f_n) : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$$

such that, for $j = 1, \ldots, r$,

$$f(\mathbf{s}_j) = (f_1(\mathbf{s}_j), \ldots, f_n(\mathbf{s}_j)) = \mathbf{t}_j.$$

Notice that, since any function over a finite field is a polynomial, each f_i is a polynomial. An algorithm that finds all models f is presented in [60]. This is done by identifying, for each gene i, the set of all possible functions for f_i. This set can be represented as the coset $f + I$, where f is a particular such function and $I \subset \mathbb{F}_2[x_1, \ldots, x_n]$ is the ideal of all Boolean polynomials that vanish on the input data set, that is, $I = \mathbb{I}(\{\mathbf{s}_1, \ldots, \mathbf{s}_r\})$. The algorithm in [60] then proceeds to find a particular model from the model space $f + I$. The chosen model, which is the normal form of f in the ideal I, depends on the term ordering used in the Gröbner bases computation. So different ordering of the variables (genes) might lead to the selection of different models. This presents a problem as the term ordering, which is a needed for computational reasons, clearly influence the model selection process.

Several modifications have since been presented to address this problem. For example, in [20], using the Gröbner fan of the ideal I, the authors developed a method that produces a

probabilistic model using all possible normal forms. Other improvements on this algorithm can be found in [53, 93].

Another approach toward improving the model selection process is by restricting the model space $f + I$ by requiring not only that the chosen model fits the data but also satisfies some other conditions, such as its network being sparse or scale-free, the polynomials f_i being monomials, the dynamics of the model having some desirable properties such as fixed points are the only limit cycles (that is, starting from any initialization, the model always reaches a steady state), or that the model is robust and stable which could roughly mean that the number of attractors in the phase space is small. In a nutshell, some but not all functions in the model space $f + I$ are biologically relevant and hence restricting the space to only relevant models will improve the model selection process.

By desiring a particular property, several classes of functions have been proposed as biologically relevant functions such as biologically meaningful rules [78], certain post classes of Boolean functions have been studied in [90], and chain functions in [31], to name few. Another class of Boolean functions, which was introduced by S. Kauffman *et al.* [55], is called (nested) canalyzing functions (NCF), where an input to a single variable exclusively determines the value of the function regardless of the values of all other variables. This is a natural characterization of "canalisation" which was introduced by geneticist C. H. Waddington [99] to represent the ability of a genotype to produce the same phenotype regardless of environmental variability. Indeed, known biological functions have been shown to be canalyzing [38, 71], and Boolean nested canalyzing networks to be robust and stable [54, 55, 71].

For the purpose of restricting the model space $f + I$ of all Boolean polynomial models to NCFs only, we previously studied nested canalyzing functions, gave necessary and sufficient conditions on the coefficients of a Boolean polynomial function to be nested canalyzing, and showed that NCFs are nothing but unate cascade functions [52]. Furthermore, in [50], the class of all nested canalyzing functions is parameterized as the rational points of an affine algebraic variety over the algebraic closure of $\mathbb{F}_2$. This variety was shown to be toric, that is, defined by a collection of binomial polynomial equations. In this paper we present an algorithm that restricts the model space to only nested canalyzing functions by identifying all NCFs from the model space $f + I$ that fit the given data set.

In the next section we briefly recall some definitions and results from [52, 50]. Our algorithm is presented in Section 2.3, and its implementation in Singular is discussed in Section 2.4. Before we conclude this paper, we demonstrate the algorithm in Section 2.5, where we identify all nested canalyzing models for the cell cycle in budding yeast using time course data from the Boolean model in [61].

2.2 Nested canalyzing functions: background

We recall some of the definitions and the results from [52, 50] that we need to make this presentation self-contained. Throughout this paper, when we refer to a function on n variables, we mean that h depends on all n variables, that is, for $i = 1, \ldots, n$, there exists $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$ such that $h(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n) \neq h(a_1, \ldots, a_{i-1}, 1 + a_i, a_{i+1}, \ldots, a_n)$.

Definition 2.2.1. Let h be a Boolean function on n variables, i.e., $h : \mathbb{F}_2^n \to \mathbb{F}_2$.

- The function h is a *nested canalyzing function* (NCF) with respect to a permutation σ on the n variables, canalyzing input value a_i and canalyzed output value b_i, for $i = 1, \ldots, n$, if it can be represented in the form

$$h(x_1, x_2, \ldots, x_n) = \begin{cases} b_1 & \text{if } x_{\sigma(1)} = a_1, \\ b_2 & \text{if } x_{\sigma(1)} \neq a_1 \text{ and } x_{\sigma(2)} = a_2, \\ b_3 & \text{if } x_{\sigma(1)} \neq a_1 \text{ and } x_{\sigma(2)} \neq a_2 \text{ and } x_{\sigma(3)} = a_3, \\ \vdots & \quad \vdots \\ b_n & \text{if } x_{\sigma(1)} \neq a_1 \text{ and } \cdots \text{ and } x_{\sigma(n-1)} \neq a_{n-1} \text{ and } x_{\sigma(n)} = a_n, \\ \overline{b_n} & \text{if } x_{\sigma(1)} \neq a_1 \text{ and } \cdots \text{ and } x_{\sigma(n)} \neq a_n. \end{cases} \tag{2.1}$$

- The function h is *nested canalyzing* if h is nested canalyzing with respect to some permutation σ, canalyzing input values $a_1, \ldots, a_n$ and canalyzed output values $b_1, \ldots, b_n$, respectively.

Remark 2.2.2. The definition above has been generalized to multistate functions in [66], where it is also shown that the dynamics of these functions are similar to their Boolean counterparts. In [76], the authors introduce what they called *kinetic models with unate structure*, which are continuous models having the canalization property, and they presented an algorithm for identifying such models.

Using the polynomial form of any Boolean function, the ring of Boolean functions is isomorphic to the quotient ring $R = \mathbb{F}_2[x_1, \ldots, x_n]/J$, where $J = \langle x_i^2 - x_i : 1 \leq i \leq n \rangle$. Indexing monomials by the subsets of $[n] := \{1, \ldots, n\}$ corresponding to the variables appearing in the monomial, the elements of R can be written as

$$R = \{ \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i \, : \, c_S \in \mathbb{F}_2 \}.$$

As a vector space over $\mathbb{F}_2$, R is isomorphic to $\mathbb{F}_2^{2^n}$ via the correspondence

$$R \ni \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i \longleftrightarrow (c_\emptyset, \ldots, c_{[n]}) \in \mathbb{F}_2^{2^n}. \tag{2.2}$$

The main result in [52] is the identification of the set of nested canalyzing functions in R with a subset V^{ncf} of $\mathbb{F}_2^{2^n}$ by imposing relations on the coordinates of its elements.

Definition 2.2.3. Let σ be a permutation of the elements of the set $[n]$. We define a new order relation $<_\sigma$ on the elements of $[n]$ as follows: $\sigma(i) <_\sigma \sigma(j)$ if and only if $i < j$. Let r_S^σ be the maximum element of a nonempty subset S of $[n]$ with respect to the order relation $<_\sigma$. For any nonempty subset S of $[n]$, the *completion of S with respect to the permutation* σ, denoted by $[r_S^\sigma]$, is the set $[r_S^\sigma] = \{\sigma(1), \sigma(2), \ldots, \sigma(r_S^\sigma)\}$.

Note that, if σ is the identity permutation, then the completion is $[r_S] := \{1, 2, \ldots, r_S\}$, where r_S is the largest element of S.

Theorem 2.2.4. *Let $h \in R$ and let σ be a permutation of the set $[n]$. The polynomial h is nested canalyzing with respect to σ, input value a_i and corresponding output value b_i, for $i = 1, \ldots, n$, if and only if $c_{[n]} = 1$ and, for any proper subset $S \subseteq [n]$,*

$$c_S = c_{[r_S^\sigma]} \prod_{\sigma(i) \in [r_S^\sigma] \setminus S} c_{[n] \setminus \{\sigma(i)\}}. \tag{2.3}$$

Corollary 2.2.5. *The set of points in $\mathbb{F}_2^{2^n}$ corresponding to the set of all nested canalyzing functions with respect to a permutation σ on $[n]$, denoted by V_σ^{ncf}, is defined by*

$$V_\sigma^{ncf} = \{(c_\emptyset, \ldots, c_{[n]}) \in \mathbb{F}_2^{2^n} : c_{[n]} = 1,\, c_S = c_{[r_S^\sigma]} \prod_{\sigma(i) \in [r_S^\sigma] \setminus S} c_{[n] \setminus \{\sigma(i)\}},\, \textit{for } S \subseteq [n]\}. \tag{2.4}$$

It was shown in [50] that V_σ^{ncf} is an algebraic variety, and its ideal $\mathbb{I}(V_\sigma^{ncf})$ is a binomial prime ideal in the polynomial ring $\overline{\mathbb{F}}_2[\{c_S : S \subseteq [n]\}]$, where $\overline{\mathbb{F}}_2$ is the algebraic closure of $\mathbb{F}_2$. Namely,

$$I_\sigma = \mathbb{I}(V_\sigma^{ncf}) = \langle c_{[n]} - 1,\, c_S - c_{[r_S^\sigma]} \prod_{\sigma(i) \in [r_S^\sigma] \setminus S} c_{[n] \setminus \{\sigma(i)\}} : S \subset [n] \rangle.$$

Furthermore, the variety of all nested canalyzing functions is

$$V^{ncf} = \bigcup_\sigma V_\sigma^{ncf}$$

and its ideal is

$$\mathbb{I}(V^{ncf}) = \bigcap_\sigma I_\sigma.$$

In the next section, we identify the set $f + I$ with the rational points in an algebraic affine variety. This will allow us to identify all nested canalyzing functions in the model space $f + I$.

2.3 Nested canalyzing models

Recall that we are given the data set $D = \{(\mathbf{s}_1, \mathbf{t}_1), \ldots, (\mathbf{s}_r, \mathbf{t}_r)\} \subset \mathbb{F}_2^n \times \mathbb{F}_2^n$. The model space could be presented by the set $f + I$ where, $f = (f_1, \ldots, f_n)$ and, for $i = 1, \ldots, n$,

$$f_i(x_1, \ldots, x_n) = \sum_{j=1}^{r} t_{j,i} \prod_{e=1}^{n} (1 - (x_e - s_{j,e})). \tag{2.5}$$

In particular, f_i is a polynomial that interpolates the data for gene i and I is the *ideal of points* of $\{\mathbf{s}_1, \ldots, \mathbf{s}_r\}$. Furthermore, the ideal I is a principal ideal in the ring R/J:

$$\begin{aligned}
I &= \mathbb{I}(\{\mathbf{s}_1, \ldots, \mathbf{s}_r\}) && (2.6)\\
&= \bigcap_{j=1}^{r} \mathbb{I}(\{\mathbf{s}_j\}) && (2.7)\\
&= \bigcap_{j=1}^{r} \langle x_1 - s_{j,1}, \ldots, x_n - s_{j,n} \rangle && (2.8)\\
&= \bigcap_{j=1}^{r} \langle 1 - \prod_{e=1}^{n} (1 - (x_e - s_{j,e})) \rangle && (2.9)\\
&= \langle \prod_{j=1}^{r} (1 - \prod_{e=1}^{n} (1 - (x_e - s_{j,e}))) \rangle. && (2.10)
\end{aligned}$$

Now a polynomial $h \in f_i + I$ if and only if $h = f_i + g(x_1, \ldots, x_n) \prod_{j=1}^{r} (1 - \prod_{e=1}^{n} (1 - (x_e - s_{j,e})))$, for some polynomial g, say $g = \sum_{H \subseteq [n]} b_H \prod_{i \in H} x_i$. By expanding the right-hand side and collecting terms, we get that $h = \sum_{S \subseteq [n]} W_S(b_H, \mathbf{s}_j, \mathbf{t}_j) \prod_{l \in S} x_l$, where, for $S \subseteq [n]$, the coefficient $W_S(b_H, \mathbf{s}_j, \mathbf{t}_j)$ is determined by $b_H, \mathbf{s}_j, \mathbf{t}_j$ for all $H \subseteq [n]$ and $j = 1, \ldots, r$.

The proof of the following theorem follows directly from Theorem 2.4.2 in [3].

Theorem 2.3.1. *Consider the ring homomorphism*

$$\Phi : \overline{\mathbb{F}}_2[\{c_S : S \subseteq [n]\}] \longrightarrow \overline{\mathbb{F}}_2[\{b_H : H \subseteq [n]\}]$$

given by, for $S \subseteq [n]$,

$$c_S \;\mapsto\; W_S(b_H, \mathbf{s}_j, \mathbf{t}_j).$$

Then $\ker(\Phi)$ *is the ideal of all polynomials that fit the data set* D*. In particular, the rational points in the variety* $\mathbb{V}(\ker(\Phi))$ *is the set of all models that fit the data set* D*, namely* $f + I$*.*

Since the ideal of all NCFs is $\mathbb{I}(V^{ncf})$, the following corollary is straightforward.

Corollary 2.3.2. *The ideal of all nested canalyzing functions that fit the data set D is* $\mathbb{I}(V^{ncf}) + \ker(\Phi)$.

Remark 2.3.3. It is clear that the model space of Boolean functions is huge, since the number of monomials grows exponentially in the number of variables. For example, if a function has 5 inputs, there are $2^5 = 32$ different monomials in 5 variables, and hence $2^{32} = 4,294,967,296$ different Boolean functions. This clearly shows that a search for NCFs inside the model space is computationally not feasible, which justifies the need for algorithms like the one above.

2.4 Algorithm

In this section we present an algorithm for identifying all nested canalyzing models from the model space of a given data set.

Input

A wiring diagram, i.e., a square matrix of dimension n, describing the influence relationships among the n genes in the network. For each variable x_i, a table consisting of the rows $(s_{j,i_1}, \ldots, s_{j,i_s}, t_{j,i})$, $j = \{1, \ldots, r\}$, where $i_1, \ldots, i_s$ are the indices of the genes that affect x_i, as specified in the wiring diagram.

Output

For each variable, the complete list of all nested canalyzing functions interpolating the given data set on the given wiring diagram. A function is in the output if it is nested canalyzing in at least one variable order. If needed, the code can easily be modified to find only nested canalyzing functions of a particular variable order.

Algorithm

It is a well known fact, that a Gröbner basis for the kernel of Φ is a basis for $\langle c_S - W_S : S \subseteq [n]\}\rangle$ intersected with the ring $\overline{\mathbb{F}}_2[\{c_S : S \subseteq [n]\}]$ [3, Theorem 2.4.2] Using a similar notation as above, the algorithm is outlined as follows:

Use ring $\mathbb{F}_2[x_1, \ldots, x_n, b_S, c_S : S \subseteq [n]\}]$
Define $\mathbb{I}(V^{ncf})$ as ideal in $\mathbb{F}_2[c_S : S \subseteq [n]\}]$
Define $h = \sum_{H \subseteq [n]} b_H \prod_{i \in H} x_i$
Define $q = \sum_{H \subseteq [n]} c_H \prod_{i \in H} x_i$

Compute the polynomial p that generates I as in (2.10)
For each variable x_i do
1. Compute f_i as in ((2.5))
2. Let $g = f_i + h * p$; its coefficients are the same as W_S above
3. Compute a Gröbner basis G for the ideal generated by the coefficients of $g - q$ using any elimination order to eliminate all b_S from G
4. Concatenate generators of G and $\mathbb{I}(V^{ncf})$
5. Compute the primary decomposition of $G + \mathbb{I}(V^{ncf})$ to obtain necessary and sufficient conditions on the coefficients of all NCFs fitting the data set D

End

An implementation of this algorithm is available as a Singular library [42, 34].

2.5 Application: inferring the cell cycle network in budding yeast

Li *et al.* [61] constructed a Boolean threshold model of the cell cycle in budding yeast. The network of the model has the key known regulators of the cell cycle process and the known interactions among these regulators in the literature. The Boolean function at each node, however, is a threshold function, which is completely determined by the numbers of active activators and active inhibitors, and is not necessarily biologically motivated. However, this model captures the known features of the global dynamics of the cell cycle, it is robust and stable, and the trajectory of the known cell-cycle sequence is a stable and attracting trajectory as it has 1764 states out of the total number of 2048 states. The remaining states are distributed into 6 small trajectories.

In this Section, we use the time course corresponding to the biological cell-cycle sequence, see Table 2.1, to infer nested canalyzing models of the cell cycle. That is, assuming the same wiring diagram as the threshold model above, we use our algorithm to identify, for each gene in the network, all nested canalyzing functions that fit the cell cycle sequence.

We start by describing the network. It consists of 11 proteins and a start signal. The proteins are members of the following three classes: cyclins (Cln1,2, Cln3, Clb1,2, and Clb5,6), inhibitors, degraders, and competitors of cyclin complexes (Sic1, Cdh1, Cdc20 and Cdc14), and transcription factors (SBF, MBF, Swi5, and Mcm1/SFF). This simplified network (Figure 2.1) is almost identical to the network in [61] where the only difference is that we do not force self-degradation, as it was added to some nodes in the network because they did not have inhibitors, but without biological justification [61]. Furthermore, we do not impose activation or inhibition in the network. As we do not use threshold functions but more general Boolean functions, a variable can both increase and decrease the concentration of another substrate, depending on the concentrations of other proteins.

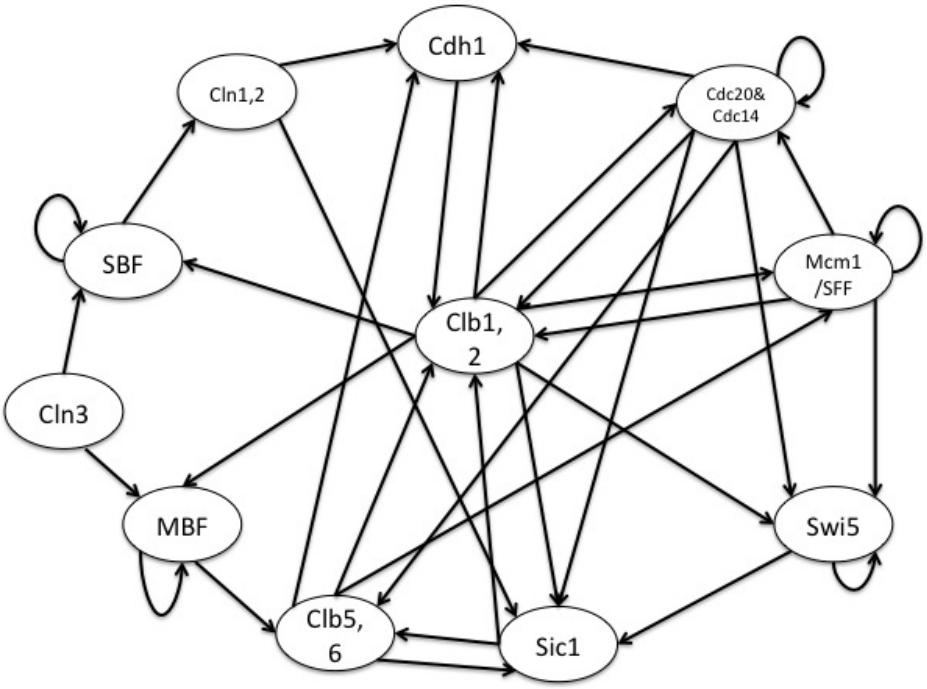


Figure 2.1: The simplified cell cycle network in budding yeast, which is based on the model in [61].

Li *et al.* [61] use their model to generate a time course of temporal evolution of the cell-cycle network, shown in Table 2.1. This time course is in agreement with the behavior of the cell-division process cycling through the four distinct phases G_1, S (Synthesis), G_2, and M (Mitosis).

We used this time course along with the network as the only input to the algorithm to obtain all nested canalyzing functions that interpolate the time course. In the fifth column of Table 2.2 we list the number of NCFs for each protein. By requiring the Boolean function to be nested canalyzing, we have significantly reduced the number of possible functions for each protein as it is evident when comparing the numbers in the third and fifth columns of Table 2.2. However, even after this reduction, there are $330,559,488$ possible nested canalyzing models that fit the time course in Table 2.1. To reduce the this number more, one needs to use additional time courses or request that the models incorporate additional biological information about yeast cell cycle.

Table 2.1: The temporal evolution of the Boolean cell-cycle model in [61]; corresponding to the biological cell-cycle sequence.

Time	Cln3	MBF	SBF	Cln1,2	Cdh1	Swi5	Cdc14,20	Clb5,6	Sic1	Clb1,2	Mcm1/SFF
1	1	0	0	0	1	0	0	0	1	0	0
2	0	1	1	0	1	0	0	0	1	0	0
3	0	1	1	1	1	0	0	0	1	0	0
4	0	1	1	1	0	0	0	0	0	0	0
5	0	1	1	1	0	0	0	1	0	0	0
6	0	1	1	1	0	0	0	1	0	1	1
7	0	0	0	1	0	0	1	1	0	1	1
8	0	0	0	0	0	1	1	0	0	1	1
9	0	0	0	0	0	1	1	0	1	1	1
10	0	0	0	0	0	1	1	0	1	0	1
11	0	0	0	0	1	1	1	0	1	0	0
12	0	0	0	0	1	1	0	0	1	0	0
13	0	0	0	0	1	0	0	0	1	0	0

2.5.1 Dynamics

To analyze the dynamics of the resulting nested canalyzing models, we randomly sampled 2000 models and analyzed them. The average number of basins of attraction (components) per network is 3.09, and the average size of the component containing the given trajectory is 1889. In only 6 models, the trajectory in Table 2.1 is not in the largest component, however, the average size of the component containing the trajectory is 833.5.

These results clearly show that nested canalyzing models for the cell cycle network are in agreement with the original threshold model of Li *et al.*, and since such models are known to be robust and stable, any of these models could be used as a model for the cell cycle in budding yeast. Furthermore, especially when there is no evidence for choosing a particular type of functions, a nested canalyzing function has an advantage over other possible choices.

2.5.2 Comparison with random networks

To understand the effect of the network itself on its dynamics, we sampled 2000 models on the same network where the local function of each gene in each one of these models is chosen randomly from all possible functions in the model space. We found that the given cell cycle trajectory has oftentimes much smaller basin of attraction, and hence random functions on the cell cycle network could not in general produce the desired dynamics. A comparison of the statistics from the sampled networks is shown in Figures 2.2 and 2.3.

Table 2.2: For each protein i, we list the number of inputs, the number of possible Boolean functions (the cardinality of $f_i + I$), the number of nested canalyzing functions with the given number of inputs, and finally the number of nested canalyzing functions in the model space $f_i + I$.

Protein (i)	inputs	$f_i + I$	NCFs	NCFs in $f_i + I$
Cln3	1	1	2	1
MBF	3	8	64	2
SBF	3	8	64	2
Cln1,2	1	1	2	1
Cdh1	4	2048	736	12
Swi5	4	2048	736	14
Cdc20&Cdc14	3	8	64	4
Clb5,6	3	8	64	3
Sic1	5	2^{24}	10,634	336
Clb1,2	5	2^{24}	10,634	61
Mcm1/SFF	3	8	64	2

2.6 Conclusion

In this paper we have presented an algorithm for identifying all Boolean nested canalyzing models that fit a given time course or other input-output data sets. Our algorithm uses methods from computational algebra to present the model space as an algebraic variety. The intersection of this variety with the variety of all NCFs, which was parameterized in [50], gives us the set of all NCFs that fit the data. We demonstrate our algorithm by finding all nested canalyzing models of the cell cycle network form Li *et al.* [61]. We then showed that the dynamics of almost any of these models is strikingly similar to that of the original threshold model. Unless the chosen model is required to meet other conditions, and in that case the model space will be reduced further, any one of the models that our algorithm found is an acceptable model of the cell cycle process in the Budding yeast.

One limitation of the current algorithm, which we left for future work, is that it does not distinguish between activation and inhibition in the network as we do not have a systematic method of knowing when a given variable in a (nested canalyzing) polynomial is an activator or inhibitor.

As our algorithm relies heavily on different Gröbner based computations, the current implementation in Singular allows a given gene to have at most 5 regulators. This is due to the fact that the number of monomials then is 32 which is already a burden especially when the primary decomposition of an ideal is what we are after. We are working on a better implementation so that we can infer larger and denser networks.

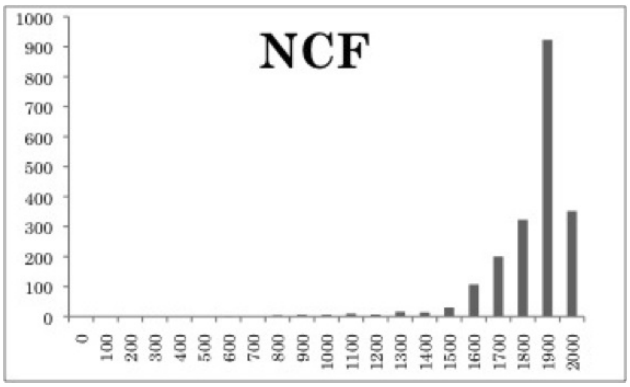


Figure 2.2: Nested canalyzing functions with the wiring diagram in Figure 2.1 interpolating the time course in Table 2.1. x-axis: size of basin of attraction for given trajectory, y-axis: number of networks observed, out of 2000.

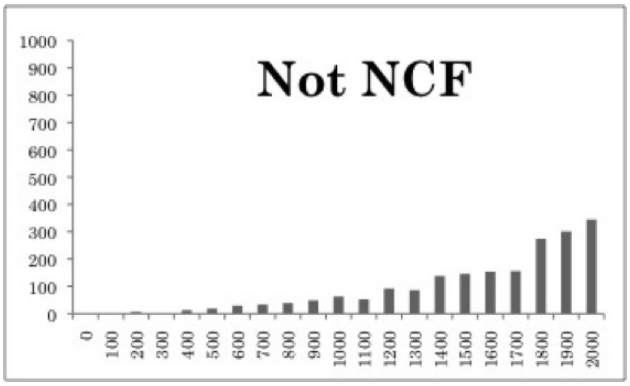


Figure 2.3: Not nested canalyzing functions with the wiring diagram in Figure 2.1 interpolating the time course in Table 2.1. x-axis: size of basin of attraction for given trajectory, y-axis: number of networks observed, out of 2000.

Chapter 3

A mathematical framework for agent-based models of complex biological networks

This manuscript is published in the *Bulletin of Mathematical Biology*. This work is authored jointly with David Murrugarra, Abdul Salam Jarrah, and Reinhard Laubenbacher. FH's contributions are the algorithm and applications.

Abstract

Agent-based modeling and simulation is a useful method to study biological phenomena in a wide range of fields, from molecular biology to ecology. Since there is currently no agreed-upon standard way to specify such models it is not always easy to use published models. Also, since model descriptions are not usually given in mathematical terms, it is difficult to bring mathematical analysis tools to bear, so that models are typically studied through simulation. In order to address this issue, Grimm et al. proposed a protocol for model specification, the so-called ODD protocol, which provides a standard way to describe models. This paper proposes an addition to the ODD protocol which allows the description of an agent-based model as a dynamical system, which provides access to computational and theoretical tools for its analysis. The mathematical framework is that of algebraic models, that is, time-discrete dynamical systems with algebraic structure. It is shown by way of several examples how this mathematical specification can help with model analysis. This mathematical framework can also accommodate other model types such as Boolean networks and the more general logical models, as well as Petri nets.

3.1 Introduction

The arsenal of modeling tools in mathematical biology has grown to include a spectrum of methods beyond the traditional and very successful continuous models with the introduction of Boolean network models in the 1960s and the more general so-called logical models in the 1980s [95]. Since then other methods have been added, in particular Petri nets (see, e.g., [37]) as models for metabolic and molecular regulatory networks. More recently, agent-based, or individual-based models, long popular in social science, have been used increasingly in areas ranging from molecular to population biology. Discrete models such as these have many useful features. Qualitative models of molecular networks such as logical models, do not require kinetic parameters but can still provide information about network dynamics and serve as tools for hypothesis generation. Agent-based models can capture the fact that in some biological systems, such as a growing tumor, system dynamics emerges from interactions at the local level, such as cell-cell interactions in the case of a tumor. Discrete models also tend to be more intuitive than models based on differential equations, so they have added appeal for researchers without a strong mathematical background.

The flip side of the coin is the relative lack of mathematical analysis tools for discrete models. While methods like bifurcation, sensitivity, and stability analysis are available for differential equations models, the principal tool in the discrete case is simulation. While this is very effective for small models, it becomes impossible for larger models, since the size of the phase space is exponential in the number of variables in the model. Thus, problems like the identification of steady states for a Boolean network model becomes problematic once the model contains many more than 20 or 30 nodes, unless one makes use of high performance computation capabilities. An added complication is the heterogeneity of the different discrete model types so that tools developed for one type are unlikely to apply to another one.

One possible approach to this problem is to find a mathematical framework that is general enough so that most or all types of discrete models can be formulated within this framework and is rich enough to provide practically useful theoretical and computational tools for model analysis. This approach was taken in [98], where it was shown that any logical model and any k-bounded Petri net can be translated into a time-discrete dynamical system over a finite state space. The transition function can be described in terms of polynomial functions. This makes model analysis amenable to the computational tools and theoretical results of computer algebra, a theoretically rich area that has taken advantage in the last decade of increasingly powerful symbolic computation capabilities. In this setting, the computation of steady states of a model, for instance, turns into the problem of solving a system of polynomial equations, for which there are several good software implementations. In this paper we show that one can make a similar translation for many agent-based models, thereby covering a large part of the discrete models available in the literature. One added benefit of this common mathematical framework is that one can use it to easily compare models of different types.

Many complex biological systems can be modeled effectively as multiagent systems in which the constituent entities (agents) interact with each other. For instance, processes unfolding in a non-homogeneous spatial environment can be modeled in this way, or processes that are inherently discrete, such as individual immune cells interacting with each other in a volume of tissues. Examples include [13, 26, 75, 100, 74]. Often, the models include large numbers of agents that can be in one of finitely many different states and interact with each other and their environment based on a set of deterministic or stochastic rules. The global dynamics emerge from the local interactions among the agents. The advantage of increased realism of agent-based models (ABMs) is counter-balanced by the relative lack of mathematical tools for their development and analysis.

One key obstacle is the lack of a formal description of ABMs in a way that makes them amenable to mathematical tools for analysis and optimal control. An important step in this direction is the work of Grimm et al., which provides a protocol for model specification. In [35] the authors point out that agent or individual based models are "more difficult to analyze, understand, and communicate" than traditional analytical models because they are not "formulated in the general language of mathematics." They proceed to develop the so-called ODD protocol for the specification of such models. The basic mathematical nature of many ABMs is that of a time-discrete dynamical system on a finite state space (either deterministic or stochastic). A state of the model can be specified as a vector of values, one for each of the model variables. In addition, a function, either deterministic or stochastic, is specified that transforms a given model state into another state. Model dynamics is generated by iteration of this function. There are other model types, such as discrete event simulations, that can also be viewed in this framework. Even hybrid models that contain continuously varying quantities, such as temperature, can sometimes be treated as discrete models in practice. The ODD protocol provides essentially a standard template for specifying the state space of the model and the update function.

Little is gained in terms of mathematical power by viewing an ABMs as a function from the set of states to itself, without any additional mathematical structure, however. And it would still be difficult to verify, in many cases, whether the update function has been specified completely and unambiguously, an important aim of the ODD protocol. Both problems could be remedied by the introduction of additional mathematical structure that provides access to mathematical tools and theoretical results, and which at the same time respects the fundamental property of ABMs that global dynamics emerges from local interactions. And the additional mathematical structure should be "benign," in the sense that it introduces few or no mathematical artifacts into the model properties, in particular its dynamics. Furthermore, it should be computationally tractable, allowing easy model comparison, for instance. The addition of such structure to the ODD protocol in a way that takes a model specified in the current ODD protocol and translates it automatically into a mathematical object would place little burden on the user, while giving access to mathematical tools. In this paper we propose such a mathematical structure and demonstrate its use via some examples.

A natural way to approximate ABMs by state space models that are grounded in a richer

mathematical theory and satisfies the constraints discussed above is to construct an algebraic model specification, that is, a discrete time, discrete state dynamical system whose state space represents exactly the dynamic properties of the ABMs. Algebraic models can be described by polynomial functions over finite fields, which provides access to the rich algorithmic theory of computer algebra and the theoretical foundation of algebraic geometry.

We demonstrate the added value that is gained from such a mathematical description through a collection of examples. The first example illustrates the fact that it is easy to check by comparing polynomials whether two different implementations of the same model are identical, using a published ABM of butterfly migration [74]. The second example consists of an epidemiological model in the form of a cellular automaton. Here one can use theoretical mathematical results to determine all periodic points of the model and their period without resorting to simulation. Finally, we show how to compute all steady states of Conway's Game of Life using computer algebra algorithms for the solution of systems of polynomial equations. In a forthcoming paper we will illustrate the use of the mathematical framework proposed here for the purpose of designing optimal control methods for agent-based models.

3.2 Algebraic models

The basic idea underlying our approach is very similar to the idea that allowed geometers to move from synthetic geometry to analytic geometry, namely the introduction of a coordinate system. That is, we need to impose an algebraic structure of addition and multiplication on the set of possible states of the model variables, so that we obtain a field. (This assumption has long been made in the case of Boolean networks, where the choice of underlying field is the Galois field $\mathbb{F}_2 = \{0, 1\}$.) This is possible whenever the number of states for a given variable is a power of a prime number. In practice, it is easy to accomplish that all variables take values in the same finite field $\mathbb{F}$, either by choosing an appropriate number of levels for a given variable at the outset or by introducing duplicates of one or more states. As with coordinate systems on real-valued spaces, there will generally be several different choices that all lead to equivalent outcomes. Once we choose such an algebraic structure $\mathbb{F}$, then the set function description of an ABM turns into a mapping between vector spaces over the finite field $\mathbb{F}$, which can be described in terms of polynomial coordinate functions. We briefly describe this class of dynamical systems and then provide a detailed description of how to translate an ABM specified in the ODD protocol into such a polynomial dynamical system.

Let $x_1, \ldots, x_n$ be a collection of variables, which take values in $\mathbb{F}$. The variables represent the entities in the system being modeled and the elements of $\mathbb{F}$ represent all possible variable states. Each variable x_i has associated to it a "local update function" $f_i : \mathbb{F}^n \longrightarrow \mathbb{F}$, where "local" refers to the fact that f_i takes inputs from the variables in the "neighborhood" of x_i. Here, "neighborhood" refers to an appropriately defined directed graph encoding the

variable dependencies. These functions assemble to a dynamical system

$$f = (f_1, \ldots, f_n) : \mathbb{F}^n \longrightarrow \mathbb{F}^n,$$

with the dynamics generated by iteration of f. Iteration can be performed by updating the variables synchronously or asynchronously.

The dynamics of f is usually represented as a directed graph on the vertex set $\mathbb{F}^n$, called the *state space* of f. There is a directed edge from $\mathbf{v} \in \mathbb{F}^n$ to $\mathbf{u} \in \mathbb{F}^n$ if and only if $f(\mathbf{v}) = \mathbf{u}$. It is easy to show [62] that any local function $f_i : \mathbb{F}^n \longrightarrow \mathbb{F}$ can be expressed as a polynomial in the variables $x_1, \ldots, x_n$. This observation has many useful consequences, since polynomial functions have been studied extensively and many analytical tools are available.

We discuss a simple example. Let $f : \mathbb{F}_3^2 \longrightarrow \mathbb{F}_3^2$ be given by $f(x_1, x_2) = (1 - x_1x_2, 1 + 2x_2)$. The state space of f has two components, containing two limit cycles: one of length two and one of length three. See Figure 3.1 (right). The dependency relations among the variables are encoded in the dependency graph in Figure 3.1 (left).

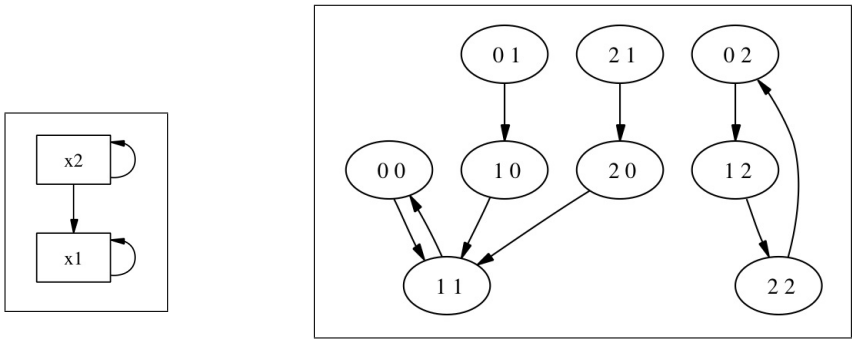


Figure 3.1: The dependency graph (left) and the state space $\mathcal{P}(f)$ (right) of the polynomial dynamical system in the above example.

It is discussed in [59] that the framework of algebraic models is particularly suitable for the study of agent-based simulations, since many agent-based simulations naturally map to this mathematical setting. Furthermore, it grounds the investigation firmly in the mathematical fields of dynamical systems and polynomial algebra, both of which provide a rich set of tools and concepts.

3.3 Polynomial form of ODD models

In [35], the authors propose a standard protocol, named ODD after its key components *Overview, Design concepts, and Details*, for describing individual based and agent-based models. The main motivation was to better enable the communication of such models. They state: "We conclude that what we badly need is a standard protocol for describing

IBMs which combines two elements: (1) a general structure for describing IBMs, thereby making a model's description independent of its specific structure, purpose and form of implementation [...] and (2) the language of mathematics, thereby clearly separating verbal considerations from a mathematical description of the equations, rules, and schedules that constitute the model." In this section we address the second element and first review the key features of the ODD protocol, using the categories from [35]. We will make two assumptions on the models this section applies to.

- All state variables in the model are updated in discrete time steps, either explicitly in the model or in the way state variable updates are computed.
- All state variables can take on only finitely many different states. (This includes state variables that include probabilities, etc., since in practice these are represented by only finitely many choices.)

While these assumptions exclude some models, they are satisfied for many ABMs found in the literature. We next describe the different parts of an ODD model and how they relate to algebraic models.

3.3.1 Purpose

This part contains a verbal description of the process the model is intended to capture and the questions one hopes to answer using the model.

3.3.2 State variables and scales

The term "state variable" refers to the low-level variables that characterize the low-level entities of the model, such as individuals or spatial entities. Another class of variables to be considered are aggregated variables such as population size or average food density. These auxiliary variables contain information that is deduced from low-level state variables. Thus, the state variables represent the fundamental components of the system, the parts whose interactions create the whole. Aggregate variables contain information about the system by aggregating information about the state variables. State variables can be grouped according to type, e.g., individuals, spatial state variables, etc.:

$$x_1, \ldots, x_n; y_1, \ldots, y_m; \ldots; z_1, \ldots, z_r.$$

Each state variable x can take on values from a prescribed set X. For instance, an individual could be described by the state vector (age, sex, location), so that X consists of a set of triples with a mixture of numerical and symbolic entries. A spatial location y could be described by the state vector (number of cars occupying the location, traffic flow), so that its state set Y contains a set of pairs with numerical entries. An *agent* is described by a collection of state variables.

3.3.3 Process overview and scheduling

This part contains a verbal description of the specific processes to be captured by the model. The scheduling aspect is very important for our purposes. Is time modeled using discrete time steps, or continuous time, or both? Are there different time scales involved, e.g., slow and fast, and which variables fall into which category? What is the update order for the different state variables, synchronous or asynchronous, with a fixed schedule or in random order? That is, the specification needs to give a complete description of the update schedule for all state variables.

3.3.4 Design concepts

For our purposes, the important aspects addressed here are:

- *A*daptation. Do the state variables change the way they interact with other state variables, either individuals or spatial state variables, as a result of changes in their environment? What are the environmental variables they sense? Do they have a fitness objective that drives adaptation?
- *I*nteraction. What are the dependencies of the state variables and what are the rules for their update?
- *S*tochasticity. Do the state variables follow deterministic or stochastic rules to update their state?

3.3.5 Input

It is necessary to specify all necessary inputs defining the state of all state variables and for computing a state update for each variable.

3.3.6 Submodels

This part contains a detailed description of model equations and rules, as well as all model parameters. It should also include a detailed justification for the choices made.

Mathematical specification. As Grimm et al. point out, the goal has to be to obtain a model description that is complete and as mathematical as possible. We now rephrase these features in a more mathematical way. The fundamental components of the model are as follows.

- The state variables. We will denote these by $x_1, \ldots, x_n$, without taking into account the different groupings based on domain-specific notions such as individual or spatial entity, etc.

- Each state variable x_i has a set of states X_i that it can be in. Thus, a state of the model is given by an element of the Cartesian product $X = X_1 \times \cdots \times X_n$. Note that for the purpose of mathematical specification it is not important that there are different types of state variables. This information is implicit in their set of possible states.

- Each state variable x_i is assigned a finite collection of rules to update its state. At each step, each state variable chooses a rule, either deterministically or stochastically, which takes as input the states of all or some other state and environmental variables, and assigns a new state to x_i. The choice of rule might involve aggregated variables and/or random choices. Note that this rule needs to provide complete information about how to determine the new state, given any admissible input state of the state variable. For instance, a bacterium might have several metabolic modes depending on the environment it finds itself in and the density of other bacteria present. The update rule chosen will then depend on the relevant environmental variables, and possibly others.

- We are given a complete specification of the order in which state variables are to be updated. That is, to compute a new state of the model, we update some variables before others, some variables simultaneously, and for some variables we choose a random update order. Different time scales can be implemented by updating faster variables several times before updating slower ones.

Observe that each rule for the update of a state variable can be expressed as a function $f_i : X \longrightarrow X_i$. We can now assemble these components to represent the model as a time discrete dynamical system

$$f = (f_1, \ldots, f_n) : X \longrightarrow X,$$

with dynamics generated by iteration. We describe the most general case of models that allow state variables to evolve and choose different update rules depending on environmental conditions. In this case, each state variable x_i has associated to it a probability space P_i of rules/update functions $f_i : X \longrightarrow X_i$, which represent its different "modes of action," depending on the environment. The probability distribution on P_i can be computed with information provided as part of the ODD. For instance, a state variable may choose an update function based on the state of one or more aggregated variables that describe its environment, such as *food density*, or the states of other state variables in its environment. For a given model update, each state variable chooses one update function from this probability space. The details of the construction can be found in Appendix 3.6.1. The key step in the construction is the replacement of each X_i with a finite field $\mathbb{F}$, so that $X = \mathbb{F}^n$.

The end result is that we can now describe the ABM as a dynamical system

$$f = (f_1, \ldots, f_n) : \mathbb{F}^n \longrightarrow \mathbb{F}^n,$$

with all $f_i \in \mathbb{F}[x_1, \ldots, x_n]$ polynomials. So what have we accomplished? Translating a model specified in the ODD protocol into a polynomial dynamical system has several advantages:

- Models are stored in a unified mathematical way.
- Ambiguities in the verbal description and incomplete information can be detected in the translation to an equation-based model.
- It is easy to implement an existing model and modify it.
- There exists a body of mathematical tools to analyze models, such as computing all steady states, which amounts to solving a certain system of polynomial equations. Also, there is a framework for optimal control in this context, which we describe in a future paper.
- It is easy to compare models.
- It is easy to incorporate variables describing the global environment, such as temperature, market price, pH value, as external parameters into the polynomial functions.

It is also worth mentioning that the mathematical framework we have created for ABMs in this way is "minimal," in the sense that all we have done is to impose a mathematical structure on the state space of the model. We have not changed or approximated the rules used to update state variables, and we have not changed the way in which the states of the model are updated. That is, we still have an exact representation of the ABM, but in precise mathematical terms.

Polynomials are neither intuitive nor are they simple functions. But they provide an exact representation of the dynamics of the model that is more compact than the state space, which is not feasible to describe for most realistic models. Any computer algebra system can be used to analyze a polynomial system, independent of a particular software or implementation. The polynomials can be generated in an almost automatic way: we provide a simple script to generate the polynomials that interpolates a given truth table [41], and tables are easily generated from the description of the model. We illustrate this process with an example of a model specified in the ODD protocol, taken from the text book [36].

3.4 Examples

We now show three examples, one of which demonstrates how to translate a model specified in the ODD protocol to its algebraic representation, and the other two show how the algebraic

representation can be used to analyze the global dynamics of the system without simulating it. We want to point out that these examples are meant as "proof of concept" illustrative demonstrations. A deeper analysis of each of them to show what the algebraic language can and cannot do is beyond the scope of this paper, and for this purpose these examples might not be the best choices. The first model was chosen because it is a key example in the expository book [36]. The second example affords an easy way to demonstrate how one might use theoretical results about algebraic models for the purpose of analyzing dynamics of models that are much too large to study thoroughly through simulation. And the third example is a widely known cellular automaton that has rarely been studied from the point of view of a dynamical system.

3.4.1 From ODD to algebraic model: virtual corridors of butterflies

We demonstrate how to formulate the algebraic description for a model given in the ODD protocol and show how this process provides guidelines to the modeler for including all relevant details and formulating the model such that it is suitable for the purpose it was built for. The example we use is a model analyzing the emergence of virtual corridors in the movements of butterflies navigating a landscape based on an elevation gradient [74]. This model was used in [36] as an example of how to specify an ABM in the ODD protocol.

We model the "hilltopping" behavior of butterflies, as they try to reach the highest point in their spatial environment for mating. The model is initialized with 500 butterflies on a landscape discretized into 150×150 square patches. A butterfly moves with probability q to the highest patch of its 8 surrounding patches, and it moves randomly with probability $1 - q$. Initially, all butterflies start out on the same patch, and simulations are run for 1000 iterations.

The *purpose*, see Section 3.3.1, of the model is to understand what conditions lead to virtual corridors and how the uncertainty of butterflies to sense the elevation gradient correctly affects these virtual corridors. This clearly stated purpose forces us to use patches as agents, i.e., use a variable x_i for each patch. Butterflies are the "acting agents," so they have to be represented as variables, too. Since the butterflies are assumed to be homogeneous the state of a butterfly only consists of its position. We enumerate the patches so every state corresponds to a different position. Patches differ by their elevation (which is fixed during a simulation) and the number of butterflies on them. At the end of a simulation, one can compare the number of butterflies on the patches to detect virtual corridors.

Algebraic model

Each butterfly is a state variable, $x_1, \ldots, x_{500}$, and so is each patch, $x_{501}, \ldots, x_{23000}$, we let x denote a state of the system, $x = (x_1, \ldots, x_{23000})$. The state of a butterfly is its position $(1, \ldots 22500)$, the state of a patch is the number of butterflies on it $(0, \ldots, 500)$. We enumerate the patches and assume that an elevation map of the modeled area is given. Updates are synchronous probabilistic updates with fixed probabilities. From the elevation map we can create 9 different tables: the first table assigns to every patch its most elevated neighbor, the remaining 8 tables assign to every patch its north (north-east, east, south-east, ..., north-west) neighbor.

Since there are 22500 different states for a butterfly and we want to work over an algebraic field, we choose $p = 22501$, the next highest prime number, as described in Appendix 3.6.1. The algebraic model is then a system of equations over $\mathbb{F}_{22501}$.

The first table has two rows. The two entries in a column, labeled a_i in the first row and b_i in the second row, for $i \in \{1, \ldots, 22500\}$ indicates that the patch adjacent to a_i with the highest elevation is b_i. We generate the polynomial $g_{j,1}$ that updates a butterfly x_j for $j \in \{1, \ldots, 500\}$ in the following way.

$$g_{j,1}(x) = \sum_{i=1}^{22500} (1 - (a_i - x_j)^{p-1}) b_i.$$

Note that for all butterflies the polynomials $g_{j,1}$ are the same because all butterflies have exactly the same hilltopping strategy and ability. This function is generated as follows. The state of a butterfly is equal to the number of the patch the butterfly is on. Therefore $(a_i - x_j)^{p-1}$ is equal to 1, except when $x_j = a_i$, i.e., the butterfly is on patch a_i, then it is 0. So $(1 - (a_i - x_j)^{p-1})$ is equal to 0, unless $x_j = a_i$, in which case it is 1, and we multiply by b_i. So $g_{j,1}(x)$ interpolates the first table, it only depends on the state of butterfly x_j, not on the other butterflies or their distribution over the patches. It is straightforward to generate $g_{j,2}(x), \ldots, g_{j,9}(x)$ for the remaining 8 tables. If a butterfly detects the correct elevation with probability q, then the probabilistic update function for butterfly x_j is

$$f_j(x) = \begin{cases} g_{j,1}(x) & \text{with probability } q \\ g_{j,2}(x) & \text{with probability } \frac{1-q}{8} \\ g_{j,3}(x) & \text{with probability } \frac{1-q}{8} \\ g_{j,4}(x) & \text{with probability } \frac{1-q}{8} \\ g_{j,5}(x) & \text{with probability } \frac{1-q}{8} \\ g_{j,6}(x) & \text{with probability } \frac{1-q}{8} \\ g_{j,7}(x) & \text{with probability } \frac{1-q}{8} \\ g_{j,8}(x) & \text{with probability } \frac{1-q}{8} \\ g_{j,9}(x) & \text{with probability } \frac{1-q}{8}, \end{cases}$$

where $g_{j,i}$ interpolates table i.

The functions for the state of the patches, f_j, $j \in \{501, \ldots, 23000\}$, are built the following way:

$$f_j(x) = \sum_{i=1}^{500} (1 - (x_i - j)^{p-1}).$$

As before, each summand is 1 or 0, depending on whether x_i is equal to j or not, respectively. An increment of 1 is added to the state of patch j whenever a butterfly x_i is in state j, i.e., on patch j. The algebraic system that describes the complete butterfly model is

$$f : \mathbb{F}_{22501}^{500+22500} \to \mathbb{F}_{22501}^{500+22500},$$

$$(x_1, \ldots, x_{23000}) \mapsto (f_1(x), \ldots, f_{23000}(x)).$$

This example demonstrates how to formalize a model given in the ODD protocol as an algebraic model, which fully captures all details of the butterfly hilltopping behavior. The algebraic framework also provides further advantages to the modeler: algebraic equations eliminate any ambiguity inherent in verbal descriptions. The functions are specified for all possible cases, some of which are often left out when specifying a function verbally.

One benefit of this description is the ease of comparison of different model implementations. We illustrate this with an example. In the model description in [36], a butterfly "moves uphill", i.e., "to the neighbor patch that has the highest elevation," with probability q. But what if a butterfly is already on the highest patch? All neighboring patches have a lower elevation, so any movement is a *downhill* movement. Two different interpretations are possible:

1. If a butterfly is on the highest patch, it stays on the same patch with probability q, to not contradict the "move uphill" instruction;
2. butterflies can always move, even if this means that "move uphill" is actually a downhill movement.

Clearly, these two interpretations lead to different update functions (which may or may not affect the qualitative dynamics of the model). The difference in a verbal description or implementation might be hard to notice, but it is easy to see the difference by comparing the resulting polynomials. Using standard computer algebra tools, the terms of each polynomial are sorted in a unique way, and the complexity of the comparison of the individual terms is linear in the number of terms. Thus, formulated as algebraic models, their difference becomes clear, a fact that otherwise might go unnoticed and lead to discrepancies in model dynamics.

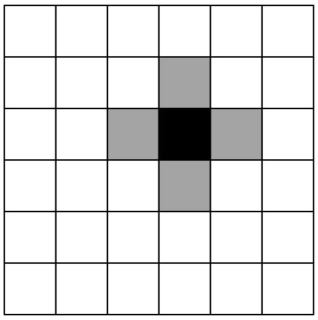
Questions about global dynamical behavior can now be formulated in terms of solving systems of polynomial equations, which come naturally from the algebraic model. Due to the large number of variables involved in the above example, the resulting polynomial systems

can not be solved easily by direct computation on a standard desktop computer with today's methods and software. However, we believe that with the advancement of algorithms and computational techniques, it will very soon be possible to analyze such models with symbolic computation techniques.

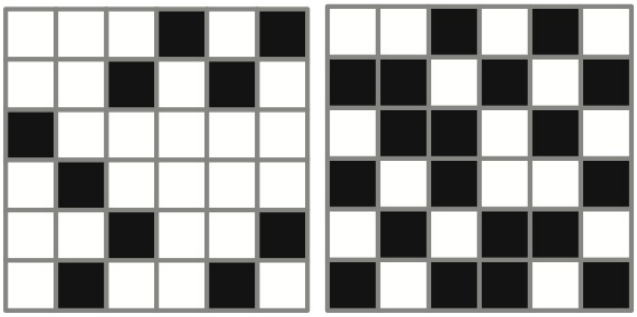
Currently, there exist no feasible mathematical methods to analyze the dynamics of general large stochastic models without iterating over the entire state space, i.e., simulating the system, in whatever format. However, it is possible in many cases to approximate a stochastic model with a deterministic system that captures the main dynamical features. Next we show two examples where mathematical theory can be used to analyze the dynamics of agent-based systems.

3.4.2 Simple infection model

Consider the following model for the spread of an infection: agents are cells on a square grid, every cell can be healthy or infected. Healthy cells are depicted in white, infected cells in black. Figure 3.2(a) shows the layout of the grid with a cell and its four neighbors. Here, we are considering the so-called von Neumann neighborhood of a cell.



(a) Grid with a cell (black) and its 4 neighbors



(b) Random infection of cells and the grid after one iteration

Figure 3.2: Infection model

The systems evolves according to the following rules: A cell acquires a healthy state, if its four neighbors are healthy, otherwise infected. Figure 3.2(b) shows a randomly infected grid and its state after one iteration.

This agent-based model can easily be translated into an algebraic model. Variables represent the cells, healthy cells have state ON (1), infected cells are OFF (0). Since there are two states for each variable, we choose $\mathbb{F}_2$ as base field. The update rule is homogeneous for all cells, and for a cell x with neighbors y_1, y_2, y_3, and y_4 it is given by

$$f_x(y_1, y_2, y_3, y_4) = y_1y_2y_3y_4.$$

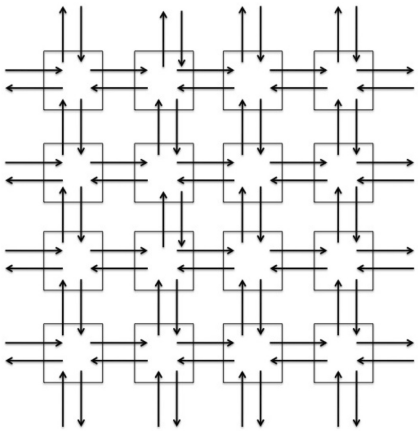


Figure 3.3: Dependency graph for infection model

One easily sees that $f_x = 0$, i.e., infected, unless all four neighbors are healthy (1). The resulting polynomial dynamical system is a so-called conjunctive Boolean network, as described in [51]. The dependency graph of this network is shown in Figure 3.3 and clearly consists of one strongly connected component, that is, any node can be reached from any other node by a directed path. We briefly describe one of the results in [51] that applies here.

The *loop number* of a directed, strongly connected graph is defined as follows: choose a vertex (the loop number is independent of the vertex chosen) and consider all directed loops at this vertex. The loop number is the greatest common divisor of the lengths (number of edges) of all such loops [51]. It is easily seen that the infection model has loop number 2. Theorem 3.8 in [51] states that the length of any limit cycle of the system has to divide the loop number, so that there are only steady states and limit cycles of period 2. Furthermore, the theorem gives an example formula for the number of limit cycles of a given length according to which the system has exactly two steady states and one limit cycle of length 2, and no other limit cycles. The two states of periodicity 2 are shown in figure 3.4. By the nature of the update rule the disease is fast spreading, a single infected neighbor is sufficient for a cell to be infected. It is interesting and counterintuitive that there is a limit cycle of length 2 with only half the cells infected. The basin of attraction of this cycle for an $n \times n$ grid is of size $2^{\frac{n^2}{2}+1} - 2$, any combination of at least half the cells being healthy and arranged as in Figure 3.4. There are two obvious steady states, given by all cells healthy or all cells infected.

Although it might be easy to find the two steady states and the limit cycle by studying the agent-based model and not its algebraic representation, one would still need to prove that these are the only fixed points and that there is exactly one limit cycle of length 2, and no limit cycles of greater length. Finding the fixed points for such a system by simulation is computationally not feasible: on a 500 by 500 grid, there are 2^{250000} different states in the state space that one would have to test.

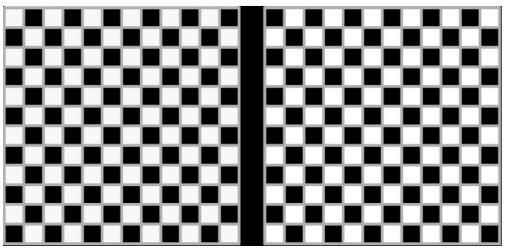


Figure 3.4: Two states of periodicity 2

The beauty of this algebraic model is, that the theorem in [51] applies to an arbitrarily large finite grid. An elegant way of dealing with boundary cells is to project the grid onto a torus by connecting the cells on the right edge to their counterparts on the left edge, and similarly for the top and bottom cells.

3.4.3 Conway's game of life

Our third example is Conway's Game of Life [30], a 2-dimensional cellular automaton (CA), using the 8 neighbors of a cell, that is, the Moore neighborhood of a cell, to compute the next state. While most ABMs are not in the form of a cellular automaton, Conway's Game of Life has some of the same characteristics as many ABMs. It is also interesting for our purpose in that it poses an interesting computational challenge for our framework. The rules of this CA have to cover many cases so that the polynomials expressing the rules are very dense, containing almost all possible terms. This affects the computational complexity of the algorithms significantly.

Cells are in one of two states, either LIVE (1) or DEAD (0), with the following behavioral rules:

1. Any live cell with fewer than two live neighbors dies, as if caused by underpopulation.
2. Any live cell with more than three live neighbors dies, as if by overcrowding.
3. Any live cell with two or three live neighbors lives on to the next generation.
4. Any dead cell with exactly three live neighbors becomes a live cell.

Although the dynamics of the system are determined by very simple rules, the emerging patterns are fascinating and have been studied extensively. Questions that are natural to ask are what steady states or oscillators can occur. We will show how to answer these questions by using an algebraic model of the Game of Life.

Naturally, the variables or agents in this system are the cells. There are only 2 possible states for an agent, DEAD or ALIVE. Therefore, we can describe its behavior with polynomials over $\mathbb{F}_2$. Every agent x has 8 neighbors, $x_1, \ldots, x_8$. The function f_x that describes the transition of agent x, is

$$f_x(x, x_1, \ldots, x_8) = \begin{cases} 0 & : \sum x_i < 2 \\ 0 & : \sum x_i = 2 \text{ and } x = 0 \\ 1 & : \sum x_i = 2 \text{ and } x = 1 \\ 1 & : \sum x_i = 3 \\ 0 & : \sum x_i > 3 \end{cases}$$

For function f_x in polynomial form, see Appendix 3.6.2. The algebraic model for the Game of Life is a system

$$f = (f_1, \ldots, f_{n\times n}) : \mathbb{F}_2^{n\times n} \longrightarrow \mathbb{F}_2^{n\times n}$$
$$x_i \mapsto f_i(x_1, \ldots, x_{n\times n}),$$

where n is the dimension of the square grid. To calculate all the fixed points, i.e., still lives of this system, we solve the system of polynomial equations:

$$f_i(x) - x_i = 0, \qquad i = 1 \ldots n \times n.$$

One can use a computer algebra system like *Macaulay2* [33] to compute a Gröbner basis of the ideal that is generated by the equations for the fixed points of the system, from which one obtains the fixed points.

For example, on a 4×4 grid with periodic boundary conditions, the fixed points of the model are the solutions to the system

$$\begin{aligned} f_1(x) &= x_1 \\ &\vdots \\ f_{16}(x) &= x_{16}. \end{aligned}$$

First we compute a Groebner basis in lexicographic order for $I = \langle f_1(x) - x_1, \ldots, f_{16}(x) - x_{16}\rangle$ over the quotient ring $\mathbb{F}_2[x_1, \ldots, x_{16}]/J$, where $J = \langle x_1^2 - x_1, \ldots, x_{16}^2 - x_{16}\rangle$. From a factorization of the Gröbner basis, one can easily read off the solutions. There are 53 fixed points.

One should remark, that all fixed points can easily be found by updating every possible initialization and checking whether it is a fixed point. For a 4×4 grid, there are only

$2^{16} = 65536$ states, so finding all fixed points is no computational challenge that would require computer algebra. As the grid increases, the complexity of this brute force approach increases exponentially, whereas the number of variables and equations increases linearly in the algebraic model. For example, for a 10×10 grid one has to check 2^{100} states. With the algebraic model though, one has to compute a Gröbner basis in a ring with 100 indeterminates for an ideal with 100 generators. Depending on the polynomials describing the update rules, this is a fast computation. Admittedly, for the functions that describe the Game of Life, we were not able to compute the solutions for a system with more than 16 variables, neither with *Macaulay2* [33] nor *Singular* [34]. However, with the rapid improvement of hardware and computer algebra software, solving the polynomial system to analyze the dynamics will soon become feasible for larger grid sizes and also faster than simulating every state which requires exhaustive enumeration of the state space.

3.5 Discussion

At this time there is no broadly agreed-upon mathematical framework which can serve as a standard for the specification and analysis of agent-based models and which preserves the key feature of this model type that global dynamics emerges from local interactions. In this paper we propose such a framework, which preserves all features of agent-based models and provides access to mathematical analysis tools. It is conceived as an extra step in the framework of the ODD protocol, which represents a first step toward a standardized protocol for the specification of agent-based models. The extra step can be automated, so that users do not need familiarity with the underlying mathematical concepts. The mathematical framework is that of polynomial dynamical systems over a finite field, which provides access to theoretical and computational tools from computer algebra and discrete mathematics.

We have presented examples of how this extra step of model specification works in practice, and we have presented examples of how the mathematical specification provides added value by allowing access to theoretical and computational tools for model analysis. We emphasize that algebraic model specification is an addition to the ODD protocol, not a replacement. The model must be explained in ODD to be understood by others. An algebraic system is an additional resource that can be used to distribute and re-use the model. It eliminates any ambiguity created by a verbal description, and it is a compact format that can run on any system independent of software implementation, so parameters and rules are easily modified for further simulations. The algebraic representation allows easy comparison between two models. The rigorous mathematical language is another advantage of the framework, rich algorithmic theory from computer algebra and the theoretical foundation of algebraic geometry are available to analyze algebraic models. We are making available a basic tool to automatically generate a polynomial from data in ODD [41] to ease the process of creating an algebraic model. Because of its many advantages, we hope that modelers will extend their model description to the algebraic description and store their models in a central location so

that models can easily be found and re-used. The polynomial systems framework unifies the representation of three important discrete model types: agent-based models, logical models (including Boolean network models), and Petri net models, allowing direct comparison of different models.

The translation algorithm presented in this paper only applies to deterministic agent-based models. And the algorithms in [98] also deal only with deterministic models. However, the majority of published ABMs in biology are stochastic. Fortunately, there are stochastic versions of several of these model types available on the mathematical side, that can be used for the purpose of modeling stochastic ABMs. The most suitable model type is that of probabilistic Boolean networks [88], a multi-state generalization of Boolean networks that is stochastic in two ways: each node of the network has attached a probability space of update functions rather than a single function and, secondly, the update order can be stochastic. It is easy to represent these models in the polynomial dynamical system framework, which we have done as part of our software package ADAM (Analysis of Dynamic Algebraic Models), available at http://adam.vbi.vt.edu as a web service.

Finally, we comment on the computational complexity of the analysis of agent-based models by the methods proposed in this paper. While a significant number of published ABMs are well within the computational reach of our methods, there are many ABMs that completely overwhelm them. This is of course similar to the situation for continuous models and parameter estimation methods, bifurcation analysis, etc. There too, models are becoming too large to be analyzed by anything other than more or less through simulation. For some ABMs even simulation represents a challenge because of their size. In these cases new methods for model reduction are the only viable approach to a mathematical analysis, no matter what methods are available. For instance, in the case of a multi-scale ABM one possible approach might be to construct phenomenological models for the higher scales that accurately model the aggregate dynamics of the lower scales without explicitly representing these. This is the subject of ongoing work by the authors.

3.6 Appendix

3.6.1 Translation from ODD to polynomial model

Here we describe the details of constructing a polynomial model from ABM information stored in the ODD protocol format.

Update schedule

The scheduling information provided allows the assembly of a complete update order for all the state variables that need to be updated, possibly involving a mixture of sequential,

parallel, deterministic and random updates of subgroups of the state variables. The scheduling information can be assembled to a probability space P, which has as elements the set of words in the letters $u_1, \ldots, u_n; v_1, \ldots, v_n$. Each word $u_i u_j u_k \cdots v_a v_b \cdots u_c \cdots$ translates into an update order $x_i x_j x_k \cdots (x_a x_b \cdots) x_c \cdots$, which is to be interpreted as updating first $x_i, x_j, x_k, \ldots$ sequentially in this order, then updating $x_a, x_b, x_c, \ldots$ in parallel, then update $x_c, \ldots$ sequentially in this order, etc. The probability distribution on the space P can be computed from the scheduling information which indicates the variables to be updated randomly and with which probability distribution. The resulting dynamical system will be denoted as

$$f = (P_1, \ldots, P_n; P) : X \longrightarrow X.$$

At this point we have described the dynamical system as a set function, which is rather limiting. For instance, if we want to compare whether two such models are identical, we need to evaluate both at all possible inputs and compare the outputs. It would be useful if we could describe the function f via equations. Now, sometimes, the model specification will already be given to us in the form of equations or mathematical functions. We now describe a procedure that allows us to represent f by mathematical functions of a unified type, in all circumstances. This procedure is analogous to the introduction of a Cartesian coordinate system to transition from synthetic geometry to analytic geometry. The fundamental observation is the following.

Observation

Let $\mathbb{F}$ be a finite field, such as $\mathbb{Z}/p$ and let $f : \mathbb{F}^n \longrightarrow \mathbb{F}$ be any function. Then there exists a unique polynomial $g \in \mathbb{F}[x_1, \ldots, x_n]$, such that each variable in g appears to a power less than $|\mathbb{F}|$, and $f(a_1, \ldots, a_n) = g(a_1, \ldots, a_n)$ for all $(a_1, \ldots, a_n) \in \mathbb{F}^n$.

Thus, in the case that all state variables take on states in the same state set, which furthermore can be given the structure of a finite field, then the dynamical system $f : X \longrightarrow X$ above can be described via polynomials. This is in fact always possible, and we briefly outline the process. It is similar to a construction described in detail in [98]. There, we show how to translate a so-called logical model into a polynomial dynamical system. Logical models are specified in a way that is very similar to the rules for state variables in an ABM. First consider one state variable and suppose that one of its states is described by an r-tuple from a set $X_1 \times \cdots \times X_r$. In each component X_i we choose an element and duplicate it enough times so that the number of elements in X_i becomes equal to the smallest prime number that is greater than or equal to the orders of all the X_j. If X_i contains ordered numerical values, for instance, say $X_i = \{1, 2, 3, 4\}$, then we can add a state $4'$ to obtain a set with 5 elements. After carrying this construction out for all X_i we have obtained a set $X_1 \times \cdots \times X_r$ in which each component has the same number of elements, equal to the power of a prime number p. It is now straightforward to see that one can endow this set with a field structure so that it becomes isomorphic to the Galois field $\mathbb{F}_{p^r}$. A similar construction can now be carried out

to assure that the order p^r is the same for the Galois fields for all state variables. The end result is that all state variables take values in the same finite field $\mathbb{F}$. The last step is to extend the update functions f_i for each state variable to the larger state set. This is done by assigning the same value to a new state as the state that was duplicated to obtain it.

3.6.2 Behavioral rule in polynomial form

For a 4 by 4 grid, we obtain for agent x_1 with neighbors $x_2 \ldots, x_9$ the following polynomial:
$f = x_1*x_2*x_3*x_4*x_5*x_6*x_7*x_8+x_1*x_2*x_3*x_4*x_5*x_6*x_7*x_9+x_1*x_2*x_3*x_4*x_5*x_6*x_8*x_9+$
$x_1*x_2*x_3*x_4*x_5*x_7*x_8*x_9+x_1*x_2*x_3*x_4*x_6*x_7*x_8*x_9+x_1*x_2*x_3*x_5*x_6*x_7*x_8*x_9+x_1*$
$x_2*x_4*x_5*x_6*x_7*x_8*x_9+x_1*x_3*x_4*x_5*x_6*x_7*x_8*x_9+x_1*x_2*x_3*x_4*x_5*x_6*x_7+x_1*x_2*x_3*$
$x_4*x_5*x_6*x_8+x_1*x_2*x_3*x_4*x_5*x_7*x_8+x_1*x_2*x_3*x_4*x_6*x_7*x_8+x_1*x_2*x_3*x_5*x_6*x_7*x_8+$
$x_1*x_2*x_4*x_5*x_6*x_7*x_8+x_1*x_3*x_4*x_5*x_6*x_7*x_8+x_2*x_3*x_4*x_5*x_6*x_7*x_8+x_1*x_2*x_3*x_4*$
$x_5*x_6*x_9+x_1*x_2*x_3*x_4*x_5*x_7*x_9+x_1*x_2*x_3*x_4*x_6*x_7*x_9+x_1*x_2*x_3*x_5*x_6*x_7*x_9+x_1*$
$x_2*x_4*x_5*x_6*x_7*x_9+x_1*x_3*x_4*x_5*x_6*x_7*x_9+x_2*x_3*x_4*x_5*x_6*x_7*x_9+x_1*x_2*x_3*x_4*x_5*$
$x_8*x_9+x_1*x_2*x_3*x_4*x_6*x_8*x_9+x_1*x_2*x_3*x_5*x_6*x_8*x_9+x_1*x_2*x_4*x_5*x_6*x_8*x_9+x_1*x_3*$
$x_4*x_5*x_6*x_8*x_9+x_2*x_3*x_4*x_5*x_6*x_8*x_9+x_1*x_2*x_3*x_4*x_7*x_8*x_9+x_1*x_2*x_3*x_5*x_7*x_8*$
$x_9+x_1*x_2*x_4*x_5*x_7*x_8*x_9+x_1*x_3*x_4*x_5*x_7*x_8*x_9+x_2*x_3*x_4*x_5*x_7*x_8*x_9+x_1*x_2*x_3*$
$x_6*x_7*x_8*x_9+x_1*x_2*x_4*x_6*x_7*x_8*x_9+x_1*x_3*x_4*x_6*x_7*x_8*x_9+x_2*x_3*x_4*x_6*x_7*x_8*x_9+$
$x_1*x_2*x_5*x_6*x_7*x_8*x_9+x_1*x_3*x_5*x_6*x_7*x_8*x_9+x_2*x_3*x_5*x_6*x_7*x_8*x_9+x_1*x_4*x_5*x_6*$
$x_7*x_8*x_9+x_2*x_4*x_5*x_6*x_7*x_8*x_9+x_3*x_4*x_5*x_6*x_7*x_8*x_9+x_1*x_2*x_3*x_4+x_1*x_2*x_3*x_5+$
$x_1*x_2*x_4*x_5+x_1*x_3*x_4*x_5+x_1*x_2*x_3*x_6+x_1*x_2*x_4*x_6+x_1*x_3*x_4*x_6+x_1*x_2*x_5*x_6+x_1*$
$x_3*x_5*x_6+x_1*x_4*x_5*x_6+x_1*x_2*x_3*x_7+x_1*x_2*x_4*x_7+x_1*x_3*x_4*x_7+x_1*x_2*x_5*x_7+x_1*x_3*$
$x_5*x_7+x_1*x_4*x_5*x_7+x_1*x_2*x_6*x_7+x_1*x_3*x_6*x_7+x_1*x_4*x_6*x_7+x_1*x_5*x_6*x_7+x_1*x_2*x_3*$
$x_8+x_1*x_2*x_4*x_8+x_1*x_3*x_4*x_8+x_1*x_2*x_5*x_8+x_1*x_3*x_5*x_8+x_1*x_4*x_5*x_8+x_1*x_2*x_6*x_8+$
$x_1*x_3*x_6*x_8+x_1*x_4*x_6*x_8+x_1*x_5*x_6*x_8+x_1*x_2*x_7*x_8+x_1*x_3*x_7*x_8+x_1*x_4*x_7*x_8+x_1*$
$x_5*x_7*x_8+x_1*x_6*x_7*x_8+x_1*x_2*x_3*x_9+x_1*x_2*x_4*x_9+x_1*x_3*x_4*x_9+x_1*x_2*x_5*x_9+x_1*x_3*$
$x_5*x_9+x_1*x_4*x_5*x_9+x_1*x_2*x_6*x_9+x_1*x_3*x_6*x_9+x_1*x_4*x_6*x_9+x_1*x_5*x_6*x_9+x_1*x_2*x_7*$
$x_9+x_1*x_3*x_7*x_9+x_1*x_4*x_7*x_9+x_1*x_5*x_7*x_9+x_1*x_6*x_7*x_9+x_1*x_2*x_8*x_9+x_1*x_3*x_8*x_9+$
$x_1*x_4*x_8*x_9+x_1*x_5*x_8*x_9+x_1*x_6*x_8*x_9+x_1*x_7*x_8*x_9+x_1*x_2*x_3+x_1*x_2*x_4+x_1*x_3*x_4+$
$x_2*x_3*x_4+x_1*x_2*x_5+x_1*x_3*x_5+x_2*x_3*x_5+x_1*x_4*x_5+x_2*x_4*x_5+x_3*x_4*x_5+x_1*x_2*x_6+x_1*$
$x_3*x_6+x_2*x_3*x_6+x_1*x_4*x_6+x_2*x_4*x_6+x_3*x_4*x_6+x_1*x_5*x_6+x_2*x_5*x_6+x_3*x_5*x_6+x_4*x_5*$
$x_6+x_1*x_2*x_7+x_1*x_3*x_7+x_2*x_3*x_7+x_1*x_4*x_7+x_2*x_4*x_7+x_3*x_4*x_7+x_1*x_5*x_7+x_2*x_5*$
$x_7+x_3*x_5*x_7+x_4*x_5*x_7+x_1*x_6*x_7+x_2*x_6*x_7+x_3*x_6*x_7+x_4*x_6*x_7+x_5*x_6*x_7+x_1*x_2*$
$x_8+x_1*x_3*x_8+x_2*x_3*x_8+x_1*x_4*x_8+x_2*x_4*x_8+x_3*x_4*x_8+x_1*x_5*x_8+x_2*x_5*x_8+x_3*x_5*$
$x_8+x_4*x_5*x_8+x_1*x_6*x_8+x_2*x_6*x_8+x_3*x_6*x_8+x_4*x_6*x_8+x_5*x_6*x_8+x_1*x_7*x_8+x_2*x_7*$
$x_8+x_3*x_7*x_8+x_4*x_7*x_8+x_5*x_7*x_8+x_6*x_7*x_8+x_1*x_2*x_9+x_1*x_3*x_9+x_2*x_3*x_9+x_1*x_4*$
$x_9+x_2*x_4*x_9+x_3*x_4*x_9+x_1*x_5*x_9+x_2*x_5*x_9+x_3*x_5*x_9+x_4*x_5*x_9+x_1*x_6*x_9+x_2*x_6*$
$x_9+x_3*x_6*x_9+x_4*x_6*x_9+x_5*x_6*x_9+x_1*x_7*x_9+x_2*x_7*x_9+x_3*x_7*x_9+x_4*x_7*x_9+x_5*x_7*$
$x_9+x_6*x_7*x_9+x_1*x_8*x_9+x_2*x_8*x_9+x_3*x_8*x_9+x_4*x_8*x_9+x_5*x_8*x_9+x_6*x_8*x_9+x_7*x_8*x_9.$

Chapter 4

ADAM: analysis of discrete models of biological systems using computer algebra

This work is authored jointly with Madison Brandon, Bonny Guang, Rustin McNeill, Grigoriy Blekherman, Alan Veliz-Cuba, and Reinhard Laubenbacher. The manuscript has is published in BMC Bioinformatics [44].

FH led the algorithm and software development. BG, MB, and RM implemented the user interface and attractor analysis, executed benchmarking calculations, and drafted the initial manuscript under FH's direction. AV implemented the translation for logical algorithms to PDS used by *ADAM*. GB participated in the software design effort and algorithm development. RL conceived of the study, provided overall leadership of the project, and secured funding for it. He also contributed to the writing and editing of the manuscript. All authors read and approved the final manuscript.

Abstract

Background: Many biological systems are modeled qualitatively with discrete models, such as probabilistic Boolean networks, logical models, Petri nets, and agent-based models, to gain a better understanding of them. The computational complexity to analyze the complete dynamics of these models grows exponentially in the number of variables, which impedes working with complex models. There exist software tools to analyze discrete models, but they either lack the algorithmic functionality to analyze complex models deterministically or they are inaccessible to many users as they require understanding the underlying algorithm and implementation, do not have a graphical user interface, or are hard to install. Efficient analysis methods that are accessible to modelers and easy to use are needed.

Results: We propose a method for efficiently identifying attractors and introduce the web-based tool Analysis of Dynamic Algebraic Models (*ADAM*), which provides this and other analysis methods for discrete models. *ADAM* converts several discrete model types automatically into polynomial dynamical systems and analyzes their dynamics using tools from computer algebra. Specifically, we propose a method to identify attractors of a discrete model that is equivalent to solving a system of polynomial equations, a long-studied problem in computer algebra. Based on extensive experimentation with both discrete models arising in systems biology and randomly generated networks, we found that the algebraic algorithms presented in this manuscript are fast for systems with the structure maintained by most biological systems, namely sparseness and robustness. For a large set of published complex discrete models, *ADAM* identified the attractors in less than one second.

Conclusions: Discrete modeling techniques are a useful tool for analyzing complex biological systems and there is a need in the biological community for accessible efficient analysis tools. *ADAM* provides analysis methods based on mathematical algorithms as a web-based tool for several different input formats, and it makes analysis of complex models accessible to a larger community, as it is platform independent as a web-service and does not require understanding of the underlying mathematics.

4.1 Background

Mathematical modeling is a crucial tool in understanding the dynamic behavior of complex biological systems. In addition to the popular ordinary differential equations (ODE) models, discrete models are now increasingly used for this purpose [85, 39, 12]. Model types include (probabilistic) Boolean networks, logical networks, Petri nets, cellular automata, and agent-based (individual-based) models, to name the most commonly found ones [92, 83, 88, 70, 9, 59]. While discrete models tend to be more intuitive than those based on differential equations, they do not have the broad range of mathematical analysis tools available that have been developed for ODE models. For small models, exhaustive enumeration of all possible state transitions of the model is the method of choice. But since the size of the state space grows exponentially in the number of model variables, this method is very limited in its applicability. For larger models sampling methods can be used to get some information about model dynamics. There are several existing sophisticated software tools available that allow users to analyze and simulate discrete networks, focused on a particular model type. These tools use a variety of computational and analytical tools for analysis purposes, with a range of different user interfaces; see, e.g., [68, 67, 81, 103, 89, 22, 82]. They will be discussed in detail in a later section.

The software tool introduced in this paper, *Analysis of Dynamic Algebraic Models (ADAM)* complements existing software packages in several ways. By translating models into the rich

mathematical framework of polynomial dynamical systems over a finite number system, we can bring to bear a variety of theoretical results, computational algorithms, and available software packages from computer algebra and computational algebraic geometry on the analysis of model dynamics. For this purpose we provide implemented algorithms that import models created in with other packages, so that the user does not need to learn a new mathematical framework [98, 48]. The basic computational workhorse underlying our software tool is the (symbolic) solution of systems of (nonlinear) polynomial equations over a finite number system. This is a well-studied problem in computer algebra and sophisticated algorithms are implemented for this purpose, which we make use of. An efficient computational implementation results in the ability to analyze model dynamics for quite large discrete models without having to resort to heuristic algorithms. We offer *ADAM* as a web service, avoiding the problems associated with software downloads and different computational platforms.

4.2 Results and Discussion

In this manuscript, we present the web-based tool *ADAM*, Analysis of Dynamic Algebraic Models [45], a tool to study the dynamics of a wide range of discrete models. *ADAM* provides efficient analysis methods based on mathematical algorithms as a web-based tool for several different input formats, and it makes analysis of complex models accessible to a larger community, as it is platform independent as a web-service and does not require understanding of the underlying mathematics. *ADAM* is the successor to DVD, Discrete Visualizer of Dynamics [58], a tool to visualize the temporal evolution of small polynomial dynamical systems.

As the underlying computational approach, we propose a novel method to identify attractors of a discrete model. This method relies on the fact that many discrete models can be translated into the algebraic framework of polynomial dynamical systems. Using these polynomials, one can construct a system of polynomial equations, such that its solutions correspond to fixed points or limit cycles. Thus, the problem of identifying attractors becomes equivalent to solving a system of polynomial equations over a finite field. This is a long-studied problem in computer algebra, and can usually be solved efficiently by using Gröbner basis methods [16]. We emphasize that this method is not a new mathematical algorithm to solve polynomial equations, but a novel approach to the analysis of discrete dynamical systems that uses a novel encoding of the periodic points of such a system as the solutions of polynomial systems derived from the model when expressed in the algebraic framework. *ADAM* allows users unfamiliar with polynomial dynamical systems or Gröbner bases to benefit from this efficient algorithm. We tested the method on several examples and had an average run time of less than one second, comparable to the performance of other software tools; and we were able to identify limit cycles of systems with more than 32 variables in less than one second.

In addition to providing access to mathematical theory for efficient analysis, algebraic models

are a unifying framework and systematic approach for several model types. This allows for an effective comparison of heterogeneous models, such as a Boolean network model and an agent-based model. For community integration in the biological sciences, *ADAM* contains a model repository of previously published models available in *ADAM* specific format [40]. This allows new users to familiarize themselves quickly with *ADAM* and to validate and experiment with existing models. In the following section, we discuss general features of *ADAM* briefly and explain new features in more detail.

4.2.1 General Features of ADAM

ADAM is a tool for analyzing different types of discrete models. It automatically converts discrete models into polynomial dynamical systems, that is, time and state discrete dynamical systems described by polynomials over a finite field (see Appendix 4.5.1 for definition and example). The dynamics of the models is then analyzed by using various computational algebra techniques. Even for large systems, *ADAM* computes key dynamic features, such as steady states, in a matter of seconds. *ADAM* is available online and free of charge. It is platform independent and does not require the installation of software or a computer algebra system.

ADAM translates the following **inputs** into (probabilistic) polynomial dynamical systems and can then analyze them.

- Logical models generated with GINsim [68]
- polynomial dynamical systems
- Boolean networks
- probabilistic polynomial dynamical systems, probabilistic Boolean networks (PBN) [88].

ADAM also translates Petri nets generated with Snoopy and we plan to implement analysis methods for Petri nets in future versions.

ADAM's main application is the analysis of the dynamic features of a model, which includes the identification of stable attractors. These are either steady states, i.e., time-invariant states, or limit cycles, i.e., time-invariant sets of states. *ADAM* is capable of identifying all steady states and limit cycles of length up to a user-specified length m. The process of finding long limit cycles is quite slow for large models. However, in biological models limit cycles are likely to be short, so that m can be chosen to be small in general, i.e., less than 10.

The temporal evolution of the model can be visualized by the *state transition graph*, the directed graph of all possible states and edges indicating their transitions, also called the

state space. For small enough models, i.e., less than eleven variables, *ADAM* generates a graph of the complete state space; for larger models, *ADAM* uses algebraic algorithms to determine dynamic properties. Independent of network size, *ADAM* generates a *wiring diagram*. The wiring diagram, also known as *dependency graph*, shows the static relationship between the variables. All edges in *ADAM*'s wiring diagrams are functional edges, that is, there exists at least one state such that a change in the input variable causes a change in the output variable (see Appendix 4.5.2 for more details). This means that *ADAM* determines all non-functional edges, which is oftentimes of interest.

With *ADAM*, one can also study the temporal evolution of user-specified initial states. The trajectory of a state describes the state's evolution, and it can be computed by repeatedly applying the transition function until an attractor is reached.

All of these features can be computed assuming synchronous updates or sequential updates according to an update-schedule specified by the user. Note that the steady states are the same independent of the update schedule. This is due to the fact that updating any variable at a steady state does not change its value. It is irrelevant for a steady state analysis whether updates are considered to happen sequentially or simultaneously.

For probabilistic networks, i.e., models in which each variable has several choices of local update rules, *ADAM* can generate a graph of all possible updates. This means that states in the phase space can have out-degree greater than one, since different transitions are possible. *ADAM* can find all true steady states, in the context of probabilistic networks, meaning all states that are time-invariant independent of the choice of update function. For further information of probabilistic networks, see [88].

For Boolean networks, *ADAM* calculates all functional circuits (see Appendix 4.5.2). Positive functional circuits are a necessary condition for multi-stationarity. For a certain class of Boolean networks, namely conjunctive/disjunctive networks, *ADAM* computes a complete description of the phase space as described in [51]. For further details on conjunctive networks, see Appendix 4.5.4.

In summary, *ADAM* can generate the following **outputs**.

- wiring diagram
- phase space for small models
- steady states (for deterministic and probabilistic systems)
- limit cycles of specified length m
- trajectories originating from a given initial state until a stable attractor is found
- dynamics for synchronous or sequential updates
- functional circuits for Boolean networks

Cell 1	SLP	wg	WG	en	EN	hh	HH	ptc	PTC	PH	SMO	ci	CI	CIA	CIR
	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15
Cell 2	SLP	wg	WG	en	EN	hh	HH	ptc	PTC	PH	SMO	ci	CI	CIA	CIR
	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25	x26	x27	x28	x29	x30
Cell 3	SLP	wg	WG	en	EN	hh	HH	ptc	PTC	PH	SMO	ci	CI	CIA	CIR
	x31	x32	x33	x34	x35	x36	x37	x38	x39	x40	x41	x42	x43	x44	x45
Cell 4	SLP	wg	WG	en	EN	hh	HH	ptc	PTC	PH	SMO	ci	CI	CIA	CIR
	x46	x47	x48	x49	x50	x51	x52	x53	x54	x55	x56	x57	x58	x59	x60

Table 4.1: Correspondence of genes and variable names. Genes and proteins in [4] and their corresponding variable names $x_1, \ldots, x_{60}$.

- a complete description of the phase space for conjunctive/disjunctive networks.

4.2.2 Applications

We show how to use *ADAM* on a well-understood model of the expression pattern of the segment polarity genes in Drosophila melanogaster. Albert and Othmer developed a model for embryonic pattern formation in the fruit fly Drosophila melanogaster [4]. Their Boolean model consists of 60 variables, resulting in a state space with more than 10^{18} states. They analyze the model for steady states by manually solving a system of Boolean equations. They also analyze the temporal evolution of a specific initial state corresponding to the wild type expression pattern by repeatedly applying the Boolean update rules until a steady state is found. The update schedule of the model is synchronous with the exception of activation of SMO and the binding of PTC to HH (activation of PH), which are assumed to happen instantaneously. This can be accounted for by substituting the equations for SMO and PH into the update rules for other genes and proteins, rather than using SMO and PH themselves.

To analyze the model, we first rename the variables in the Boolean rules given in [4] such as wg_i or SLP_i to $x_1 \ldots x_{60}$, to standardize their format. The variables x_i and their corresponding genes are listed in Table 4.1. Then we use *ADAM*: the model type is *Polynomial Dynamical Systems*, the number of states in a Boolean model is 2, representing 'present' or 'absent'. One can choose *Boolean*, and enter the Boolean rules in the text-area or upload a text file with the Boolean rules. Alternatively, one can first convert the Boolean rules to polynomials over $\mathbb{F}_2$, and enter the polynomials with the choice *Polynomial*. The file with the polynomial equations for the model can be accessed at [40].

The rules in the model file are specified in *Polynomial* form. Once the polynomials are uploaded, we need to set the *Analysis* type. The model with 60 variables is too complex for exhaustive enumeration, and we choose *Algorithm*. This means that instead of exhaustive enumeration of the state space, analysis of the dynamics is done via computer algebra by

solving systems of equations. In *Options*, we set *Limit cycle length* to 1 because we are interested in the steady states, i.e., time-invariant states. We chose *Synchronous* as updating scheme. Once these choices have been made, we obtain the steady states by clicking *Analyze*. *ADAM* returns a link to the *wiring diagram* (or *dependency graph*), which captures the static relations between the different variables. In addition, *ADAM* returns the number of steady states and the steady states themselves: see Figure 4.1. These steady states are identical to those found in [4], half of which have been observed experimentally.

00000000100110100000000100110110000000100110110000000001001101
00011110001000000011110001000011100001111111011100001111111 0
00000001111111000011110001000011100001111111010000000100 1101
01100001111111000011110001000011100001111111010000000100 1101
00000001111111000011110100000011100001111111010000000100 1101
01100001111111000011110100000011100001111111010000000100 1101
00011110001000000000001111111010000000100110111100001111111 0
00011110100000000000001111111010000000100110111100001111111 0
00011110001000001100001111111010000000100110111100001111111 0
00011110100000001100001111111010000000100110111100001111111 0

Figure 4.1: *ADAM*: Analysis of steady states of Drosophila model. Each row in the table corresponds to a stable attractor. Attractors are written as binary strings, where 0 represents non-expression of a gene (or low concentration of a protein), and 1 expression (or high concentration)

Each row in the table in Figure 4.1 corresponds to a stable attractor. Attractors are written as binary strings, where 0 represents non-expression of a gene (or low concentration of a protein), and 1 expression (or high concentration); e.g., the binary string

$$\begin{aligned}(&0001111000100000000000111111 0\\ &1000000100110111100001111111 0)\end{aligned} \tag{4.1}$$

corresponds to the genes (and proteins) being expressed (or present in high concentration) in four cells from anterior to posterior compartments (compartment 1 to 4). The string can be translated back to a list of genes that are expressed in this stable attractor; see Table 4.2.

This is the steady state obtained in [4] when starting the system with an initial state representing the experimental observations of stage 8 embryos.

ADAM can also generate trajectories for a given initial state. For example, we can choose the initial state that was used in [4] representing stage 8 embryos. Again, we enter *Polynomial*

compartment 1	en, EN, hh, HH, SMO
compartment 2	ptc, PTC, PH, SMO, ci, CI, CIA
compartment 3	SLP, PTC, ci, CI, CIR
compartment 4	SLP, wg, WG, ptc, PTC, PH, SMO, ci, CI, CIA

Table 4.2: Genes and proteins present in steady state. Genes and proteins present in steady state corresponding to binary string (4.1).

compartment 1	en, hh
compartment 2	ptc, ci
compartment 3	SLP, ptc, ci
compartment 4	SLP, wg, ptc, ci

Table 4.3: Genes and proteins present in initial state corresponding to binary string (4.2).

Dynamical Systems with 2 as the number of states and upload the polynomials describing the model. Instead of *Algorithms*, we now choose *Simulation*. Since we are not interested in the number of steady states or the complete phase space, but in a single trajectory originating from a specific initial state, we choose *One trajectory starting at an initial state* as the simulation option. We enter the state corresponding to the initial state shown in table 4.3 as a binary string:

$$(00010100000000000000010001000 \\ 10000010001000110000010001000). \tag{4.2}$$

By clicking *Analyze*, we obtain the temporal evolution of this particular state until it reaches a steady state; see Figure 4.2. As predicted in [4], the steady state is the state corresponding to the state shown in Table 4.2.

Initial State	00010100000000000000010001000100000010001000110000010001000
after 1 iteration	00001010001100000000000101110010000000101110011100000010111 00
after 2 iteration	00010100001010000000000011111010000000100110111100000011111 0
after 3 iteration	00011010001101000000001001111010000000100110111100001001111 0
after 4 iteration	00011100001010000000001101111010000000100110111100001101111 0
after 5 iteration	00011110001001000000001100111010000000100110111100001100111 0
6 - Fixed Point	00011110001000000000001111111010000000100110111100001111111 0

Figure 4.2: *ADAM*: Trajectory of Drosophila model

To summarize, *ADAM* correctly identified the steady states in less than one second. All steady states have been determined previously in [4] by labor-intensive manual investigation of the system.

Furthermore, we used *ADAM* to verify that there are no limit cycles of length two or three. The model has not been analyzed previously for limit cycles. The absence of two- and three-cycles strengthens confidence in the model, since oscillatory behavior has not been observed experimentally. Computations for limit cycles of length greater than three have not been conducted, as composing the system several times with itself is computationally complex. The model file in *ADAM* format can be accessed at [40].

4.2.3 Benchmark Calculations

We analyzed logical models available in the GINsim model repository [69] as of August 2010. The repository consists of models in GINsim XML format previously published in peer-reviewed journals. We converted all but two models into polynomial dynamics systems. For these 26 models we computed the steady states. All calculations finished in less than 1,5 seconds; see Figure 4.3.

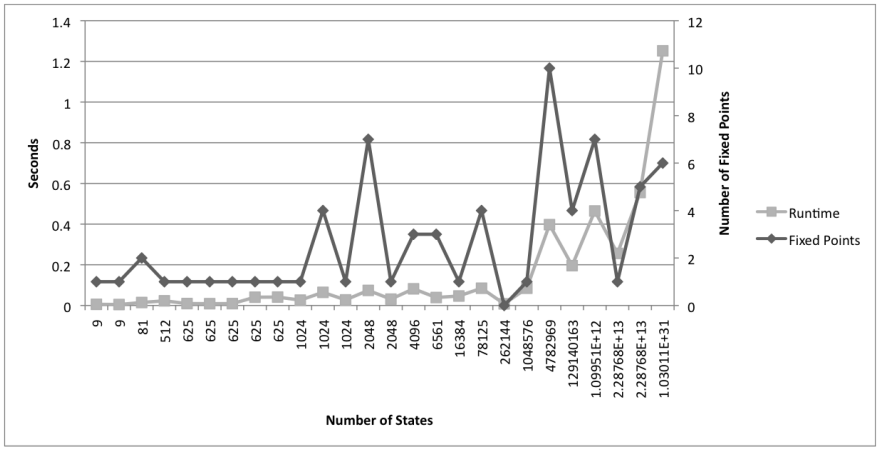


Figure 4.3: Runtime of steady state calculations of several logical models from [69]. Executed on a 2.7 GHz computer.

In addition to the published models in [69], we analyzed randomly generated networks that have the same structure that we expect from biological systems, namely sparse, i.e., while the number of nodes in a biological network may be quite large, each node is affected only by a small number of other nodes, and robust, i.e., small number of attractors. We tested a total of 50 networks with 50-150 nodes ($10^{15} - 10^{45}$ states) and an average of average in-degrees of 1.6848. The steady state calculations took less than half a second for each network on a 2.7 GHz computer.

	Steady State Analysis	Limit Cycle Analysis	Input Format	System Requirements
ADAM	Yes‡	Yes◇	Boolean (or polynomial) functions Logical Models (GINsim)	None, web based
GINsim	Yes‡	For small models	Parameters (non-zero truth tables) Logical Model	Java virtual machine◦
BoolNet R package	For small † models	For small † models	Boolean functions	R statistics software
DDLab	For small models	For small models	Logical tables	◦
BN/PBN Matlab Toolbox	For small models	For small models	Logical tables	Matlab

Table 4.4: Comparison of different software tools regarding attractor analysis: ‡ less than 1 second on published gene regulatory networks with up to 72 variables; ◇ only for short limit cycles; † heuristic methods are available for larger networks; ◦ installation necessary, available for common operating systems.

4.2.4 Comparison to Other Systems

In this section, we describe the functionality of several state-of-the-art software tools for the analysis of discrete models of biological systems. They are all capable of identifying steady states and limit cycles by exhaustive enumeration of the state space for small models (less than 32 variables) [68, 67, 103, 89, 22, 82]. For larger models, GINsim is capable of analyzing models for steady states, and several tools provide heuristic analysis methods. None of them identifies limit cycles deterministically for models with more than 32 variables. It is important to stress that ADAM provides a web-interface and does not require local installation as all the other tools do, which makes them less accessible to users. Table 4.4 summarizes the features of the different software tools, which we will now explain in detail.

GINsim (Gene Interaction Network simulation) is a package designed for the analysis of gene regulatory networks [68]. As input, it accepts logical models. Logical models are an extension of Boolean models; they consist of similar switch-like rules, but allow for a finer data discretization with more than two states per variable, e.g., *low*, *medium*, and *high*. Logical models can be updated synchronously or asynchronously. For the latter, the temporal evolution of a logical model is non-deterministic because the variables are updated randomly in an asynchronous fashion. In either case, updates of every variable are *continuous*, meaning that no variable changes its value by more than one unit in one time-step, see section *Remarks about Logical Models* for a detailed discussion.

GINsim provides algorithms that use binary decision diagrams (BDD) for the determination of steady states [70]. Analysis of limit cycles is executed by simulating every trajectory, i.e., generating the complete state space, called state transition graph in GINsim, and therefore limited by network size. We tested GINsim on logical models with up to 72 variables;

determining the steady states took less than one second. More complex logical networks were not available to us.

BoolNet R package provides methods for inference and analysis of synchronous, asynchronous, and probabilistic Boolean networks [67]. It is a package for the free statistics software *R*, and it is run via the *R* command-line. It is helpful, if the user is already familiar with *R*. Steady state analysis is implemented as exhaustive search of the state space, heuristic search, random walk, or Markov Chain analysis [88]. Non-heuristic analysis is limited to networks with 29 variables. For larger networks, steady states can be inferred heuristically, which does not guarantee that all steady states are identified.

DDLab is an interactive graphics software for discrete models, including cellular automata, Boolean and multi-valued networks [103]. As it is mainly a visualization tool, analysis is based on exhaustive enumeration of the state space, and model size is limited to 31 variables.

BN/PBN Toolbox is a toolbox written in *Matlab* [89]. It uses the state transition matrix to compute attractors. Statistics for networks with more than 27 variables cannot be computed ("Maximum variable size allowed by the program is exceeded"). In addition to analyzing deterministic Boolean networks, the toolbox can analyze probabilistic Boolean networks and calculate statistics such as numbers and sizes of attractors, basins, transient lengths, Derrida curves, percolation on 2-D lattices, and influence matrices.

Remarks about Logical Models

In this manuscript, we distinguish between three different update types: synchronous, sequential according to an update schedule, and asynchronous. *ADAM* allows for synchronous or sequential updates according to a given update schedule. In models with synchronous updates, all variables are updated simultaneously at every time step. In models with sequential updates according to an update schedule, all variables are updated at every time-step in the order given by the schedule. Both these model types are deterministic. In models with asynchronous updates, as is common for logical models, one variable is updated at random at every time step, which results in a non-deterministic model. Models with sequential updates according to an update schedule produce dynamics that is different from that of models with asynchronous updates, i.e., logical models.

In GINsim, all models are *continuous* in the sense that at each time-step, each variable increases or decreases by at most one unit. Though logical models are discrete, there are no jumps skipping intermediate states. For example, in a model with three states, low, medium, and high, no variable can drop from high to low in a single update step. This interpretation is different from the common meaning of continuous, which usually refers to models of ordinary or partial differential equations. The parameters entered in GINsim specify the target value towards which the variable changes, i.e., the value increases by

next state of x_2	low x_2	medium x_2	high x_2
x_1 absent	low x_2	medium x_2	high x_2
x_1 present	medium x_2	high x_2	high x_2
extension x_1 present	medium x_2	high x_2	high x_2

Table 4.5: Multi-valued models. Updates for variable x_2 in a logical model, where x_2 depends on x_1 and itself. The states 0 and 1 represent absent and present for the Boolean variable x_1; 0, 1, and 2 represent low, medium, and high for the multi-valued variable x_2. The last row is introduced in the polynomial dynamical system such that all variables are defined over $\mathbb{F}_3$. The extra states $(2,0),(2,1),(2,2)$ in the state space should be ignored when interpreting the dynamics.

one, decreases by one, or remains constant if the target value is larger, smaller, or equal than the initial value, respectively. The state transition graph generated with *ADAM* might differ from the state transition graph generated in GINsim. To obtain the exact same state transition graph, every variable in the logical model must contain an explicit self-loop, and all parameters must be entered such that the target value differs by at most one from the value of the variable to be updated. Any logical model can be specified in this way without changing its state transition graph. Boolean models are always continuous.

In multi-valued logical models, variables can have different maximum values. In an algebraic model, all variables are defined over the same algebraic field, i.e., have the same maximum value. When a multi-valued logical model is translated into an algebraic model, extraneous states might be introduced such that all variables are defined over the same field. An example of such an extension is given in Table 4.5, the extra states are the states in the last row, which are given the same values as the states above to extend the model in a meaningful way. The extra states should be ignored when analyzing the dynamics. For more details, see [98].

4.2.5 Architecture

ADAM is available as a web-based tool that does not require any software installation. *ADAM*'s user interface is implemented in HTML. We use JavaScript to generate a dynamic website that adapts as the user makes various choices. This simplifies the process of entering a model. For example, after defining the model type, i.e., Polynomial Dynamical System, Probabilistic Network, Petri net, and Logical Model the next line changes to the number of states, k-bound, or nothing, appropriately. Input can be entered directly into the text area, or uploaded as a text document.

All mathematical algorithms are programmed in Macaulay2 [33]. Macaulay2 is a powerful computer algebra system. The routines for which fast execution is crucial are implemented

in C/C++ as part of the Macaulay2 core. Logical Models and Petri nets in XML format are parsed using Ruby's XmlSimple library. The interplay between HTML and Macaulay2 is also programmed in Ruby.

Output graphs are generated with Graphviz's *dot* command. When *Simulation* is chosen as analysis method, Graphviz's *ccomps - connected components filter for graphs* is used to count the connected components. A Perl script directs the execution of the Graphviz commands.

4.2.6 Model Repository

A model repository is part of the *ADAM* website [40]. The repository consists of a collection of several previously published models in *ADAM* format. The models are extracted from publications, and rewritten in *ADAM* specific format, i.e., all variables are renamed to x_i and the update rules from the original publication are reformulated as Boolean rules or polynomials. The central repository with models in a unified format allows for quick verification and experimentation with published models. By changing parameters or initial states, users can gain a better understanding of the models.

New users can also use the repository to quickly familiarize themselves with the main functionalities of *ADAM*. In addition to the model itself, the database entries contain a short summary of the biological system and relevant graphs, together with an analysis of dynamic features determined by *ADAM* and their biological explanation. The repository is work in progress by researchers from several institutions generating more entries for the repository. We invite all interested researchers to submit their models.

Because of their intuitive nature, discrete models are an excellent introduction to mathematical modeling for students of the life sciences. *ADAM*'s model repository is a great starting point to familiarize students with the abstraction of discrete models such as Boolean networks.

4.3 Conclusions

Discrete modeling techniques are a useful tool for analyzing complex biological systems and there is a need in the biological community for easy to use analysis software. *ADAM* provides efficient methods as a web-based tool and will allow a larger community to use complex modeling techniques, as it is platform independent and does not require the user to understand the underlying mathematics. Upon translating discrete models, such as logical networks, Petri nets, or agent-based models into algebraic models, rich mathematical theory becomes available for model analysis, e.g., for steady state and limit cycle analysis.

After extensive experimentation with both discrete models arising in systems biology and randomly generated networks, we found that the algebraic algorithms presented in this

manuscript are fast for sparse systems with few attractors, a structure maintained by most biological systems. All algorithms have been included in the software package *ADAM*[45], which is user-friendly and available as a free web-based tool. *ADAM* is highly suitable to be used in a classroom as a first introduction to discrete models because students can use it without going through an installation process.

ADAM provides methods to analyze the key dynamic features, such as steady states and limit cycles, for large-scale (probabilistic) Boolean networks and logical models. *ADAM* unifies different modeling types by providing analysis methods for all of them and thus can be used by a larger community.

We hope to expand *ADAM* to a more comprehensive Discrete Toolkit which incorporates new analysis methods, better visualization, and automatic conversion for more model types. We also hope to analyze controlled algebraic models and expand theory to stochastic systems.

4.4 Methods

Logical models, Petri nets, and Boolean networks are converted automatically into the corresponding polynomial dynamical system as described in [98], so that algorithms from computational algebra can be used to analyze the dynamics. In polynomial dynamical systems over a finite field, states of a variable are assigned to values in the field, and the update (or transition) rule for each variable is given as a polynomial rather than a Boolean or logical expression. For more details, see Appendix 4.5.1. Using these polynomials, one can construct systems of polynomial equations, such that their solutions correspond to fixed points or limit cycles. Thus, the problem of identifying attractors becomes equivalent to solving a system of polynomial equations over a finite field. This is a long-studied problem in computer algebra, and can usually be solved efficiently by Gröbner basis methods.

Gröbner basis calculation is for polynomial systems what Gauss-Jordan elimination is for linear systems: a structured way to transform the original system to triangular shape without changing its solution space. The triangular shape of the resulting systems allows for stepwise retrieval of the solutions of the system. For a more in depth discussion of Gröbner bases, see for example [16].

In the worst case, computing Gröbner bases for a set of polynomials has complexity doubly exponential in the number of solutions to the system. However, in practice, Gröbner bases are computable in a reasonable time. It has been suggested, that in robust gene regulatory networks genes are regulated by only a handful of regulators [80]. Thus, the polynomial dynamical systems representing such biological networks are sparse, i.e., each function depends only on a small subset of the model variables. From our experience, a Gröbner basis calculation for sparse systems with few attractors, a structure common for biological systems, is actually quite fast.

4.5 Appendix

4.5.1 Polynomial Dynamical Systems

To be self-contained, we briefly explain polynomial dynamical systems and their key features.

A **polynomial dynamical system** (PDS) [53] over a finite field k is a function

$$f = (f_1, \ldots, f_n) : k^n \to k^n,$$

with coordinate functions $f_i \in k[x_1, \ldots, x_n]$, the ring of polynomials in the variables x_i, with coefficients in k. Iteration of f results in a time-discrete dynamical system. A PDS can be used to describe the dynamic behavior of a biological system: every variable x_i corresponds to a biological substrate, for example a protein or gene, and the polynomials f_i describe the evolution of x_i depending on the previous state of the variables $x_1, \ldots, x_n$.

PDS have several dynamic features of biological relevance. These include the number of components, component sizes, steady states, limit cycles, and limit cycle lengths.

Example 4.5.1. Let $k = \mathbb{F}_2$ and $f = (f_1, f_2, f_3) : \mathbb{F}_2^3 \to \mathbb{F}_2^3$ with

$$\begin{aligned} f_1 &= x_1x_2x_3 + x_1x_2 + x_2x_3 + x_2 \\ f_2 &= x_1x_2x_3 + x_1x_2 + x_1x_3 + x_1 + x_2 \\ f_3 &= x_1x_2x_3 + x_1x_3 + x_2x_3 + x_1 + x_2. \end{aligned}$$

The wiring diagram of f, which shows the static interaction of the three variables, is depicted in Figure 4.4 (left) along with its phase space in Figure 4.4 (right). The state transition graph shows the temporal evolution of the system. Each state is represented as a vector of the values of the three variables (x_1, x_2, x_3). The PDS described by f has two stable attractors: a steady state, (000), and a limit cycle of length three, consisting of the states (010), (111), and (011).

A **probabilistic PDS** over a finite field k is a collection of functions

$$f = (\{f_{1,1}, \ldots, f_{1,r_1}\}, \ldots, \{f_{n,1}, \ldots, f_{n,r_n}\}) : k^n \to k^n,$$

together with a probability distribution for every coordinate that assigns the probability that a specific function is chosen to update that coordinate. The coordinate functions $f_{i,j}$ are elements in $k[x_1, \ldots, x_n]$. Probabilistic PDS, specifically Boolean probabilistic networks (PBN), have been studied extensively in [88]. *ADAM* analyzes probabilistic PDS. It can simulate the complete state transition graph for sufficiently small models, by generating every possible transition and labeling the edge with its probability according to the distribution. If no distribution is given, *ADAM* assumes a uniform distribution on all functions. For large networks, *ADAM*'s *Algorithm* choice computes steady states of probabilistic networks.

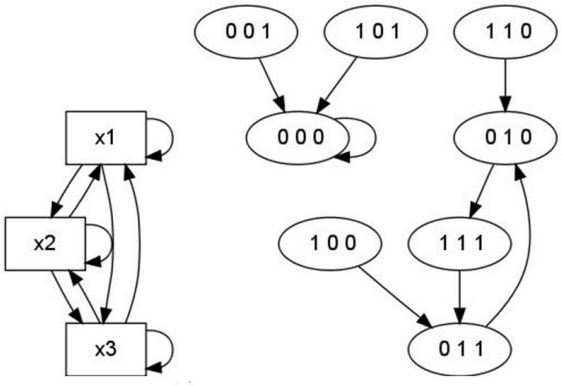


Figure 4.4: (left) Wiring diagram: static relationship between variables (right) Phase space: temporal evolution of the system.

4.5.2 Functional Edges

An edge in the wiring diagram from x_i to x_j is considered functional, if there exists a state $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_n)$ such that $f_j(\hat{x}_1, \ldots, a, \ldots \hat{x}_n) \neq f_j(\hat{x}_1, \ldots, b, \ldots \hat{x}_n)$, where a and b are values for x_i, in other words, if there is at least one state, such that changing only x_i but keeping all other values fixed, changes the next state of x_j. In *ADAM*, all edges in the wiring diagram are functional. For Boolean networks, *ADAM* identifies all functional elementary circuits. An elementary circuit is a finite closed path in the wiring diagram in which all the nodes are distinct. The existence of functional circuits is a necessary condition for multi-stationarity and limit cycles. For a further discussion of functional circuits, see [70]. For multivalued networks, circuit analysis has not yet been implemented.

4.5.3 Analysis of stable attractors

Every attractor in a PDS is either a steady state or a limit cycle. For small models, *ADAM* determines the complete phase space by enumeration, for large models, *ADAM* computes steady states and limit cycles of a given length. A state is a steady state, if it transitions to itself after one update of the system. A state is part of a limit cycle of length m, if, after m updates, it results in itself. Any steady state of a PDS satisfies the equation $f(x) = x$, as no coordinate of x is changing as it is updated. Similarly, states of a limit cycle of length m satisfy the equation $f^m(x) = x$. *ADAM* computes all steady states by solving the system $f_i(x) - x_i = 0$ for $i \in \{1, \ldots, n\}$ simultaneously. To efficiently solve the resulting systems of polynomial equations, we first compute the Gröbner basis in lexicographic order for the ideal generated by the equations. Choosing a lexicographic order allows to easily obtain the solutions [16]. We use the Gröbner basis algorithms distributed with Macaulay2, version 1.3.1.1, and found that for quotient rings over a finite field the implementation 'Sugarless' is more efficient than the default algorithm with 'Sugar' [33, 32]. For limit cycles of length m,

the solutions of $f^m(x) = x$ are found and then grouped into cycles, by applying f to each of the solutions.

Example 4.5.2. Fixed points of the system shown in the example in 4.5.1 are solutions in $\mathbb{F}_2^3$ of the system $f(x) = x$:

$$\begin{aligned} x_1x_2x_3 + x_1x_2 + x_2x_3 + x_2 &= x_1 \\ x_1x_2x_3 + x_1x_2 + x_1x_3 + x_1 + x_2 &= x_2 \\ x_1x_2x_3 + x_1x_3 + x_2x_3 + x_1 + x_2 &= x_3. \end{aligned}$$

The only solution to this systems is the point $(x_1, x_2, x_3) = (0, 0, 0)$. This is in accordance with the state transition graph depicted in Figure 4.4 : $(0, 0, 0)$ is the only steady state. To investigate limit cycles of length two, one has to look at the system $f^2(x) = x$,

$$\begin{aligned} g_1(x) &= f_1(f_1(x), f_2(x), f_3(x)) \\ &= x_1 * x_2 + x_2 * x_3 = x_1 \\ g_2(x) &= f_2(f_1(x), f_2(x), f_3(x)) \\ &= x_1 * x_2 * x_3 + x_1 * x_2 + x_1 * x_3 + x_1 + x_2 = x_2 \\ g_3(x) &= f_3(f_1(x), f_2(x), f_3(x)) \\ &= x_1 * x_2 * x_3 + x_2 = x_3. \end{aligned}$$

Again, $(0, 0, 0)$ is the only solution, which means that there are no limit cycles of length two.

Investigating $f^3(x) = x$,

$$\begin{aligned} f_1(g_1(x), g_2(x), g_3(x)) &= x_1 \\ f_2(g_1(x), g_2(x), g_3(x)) &= x_2 \\ f_3(g_1(x), g_2(x), g_3(x)) &= x_3, \end{aligned}$$

results in the solutions $(0, 0, 0), (0, 1, 0), (0, 1, 1), (1, 1, 1)$. $(0, 0, 0)$ is a steady state, and $(0, 1, 0), (0, 1, 1), (1, 1, 1)$ are elements of a limit cycle of length 3. For all $m > 3$, $f^m(x) = x$ has no solutions, that means the system f has exactly two attractors, a steady state a a limit cycle of length 3.

4.5.4 Conjunctive/Disjunctive Networks

Some classes of networks have a certain structure that can be exploited to achieve faster calculations. Jarrah et al. show that for conjunctive (disjunctive) networks key dynamic features can be found with almost no computational effort [51]. Conjunctive (respectively disjunctive) networks consist of functions using only the AND (respectively OR) operator. *ADAM* comes with an implementation of this algorithm to analyze dynamics in the case of conjunctive (disjunctive) networks. Currently, this option is only implemented for networks with strongly connected dependency graphs.

Acknowledgements

Dimitrova, Clemson University; J. Adeyeye, Winston-Salem State University; B. Stigler, Southern Methodist University; R. Isokpehi, Jackson State University are currently expanding *ADAM*s Model Repository. Funding for this work was provided through U.S. Army Research Office Grant Nr. W911NF-09-1-0538, National Science Foundation Grant Nr. CMMI-0908201, and National Science Foundation Grant Nr. 0755322.

Chapter 5

Fast Gröbner basis computation for Boolean polynomials

This manuscript is currently under revision for the *Journal of Software for Algebra and Geometry: Macaulay2* [43]. The work is authored jointly with Elizabeth Arnold. FH's contribution are the implementation and application.

Abstract

We introduce the Macaulay2 package *BooleanGB*, which computes a Gröbner basis for Boolean polynomials using a binary representation rather than symbolic. We compare the runtime of several Boolean models from systems in biology and give an application to Sudoku.

5.1 Introduction

Buchberger's algorithm provides a theoretical basis for computing a Gröbner basis for any ideal of a multivariate polynomial ring with rational coefficients. However, due to memory constraints, many examples still cannot be computed in real time. Two difficulties occur in the symbolic Gröbner basis computations over the rational numbers. During the computation of Buchberger's algorithm, the size of the coefficients of the intermediate polynomials can grow to be very large, even if the size of the coefficients of the input polynomials and the reduced Gröbner basis are relatively small. The degrees of the leading terms of the intermediate polynomials can also grow very large relative to the input and output polynomials. These high degree leading terms allow large numbers of monomials in the intermediate polynomials. Boolean polynomials in the quotient ring, $QR = \mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2+x_1, x_2^2+x_2, \ldots, x_n^2+x_n\rangle$, where coefficients are restricted to 0 and 1, and polynomials are linear in each variable, avoid

both of these difficulties. We implement a modified Buchberger algorithm in Macaulay2 that takes advantage of this situation. Similar implementations in other systems appear in [8, 10, 87].

This paper focusses on a fast Gröbner basis implementation for Boolean polynomials in *C++* designed for Macaulay2 and included with version 1.4 [33]. In Section 5.2 we discuss the M2 algorithm, *gbBoolean*, the binary representation for polynomials and the corresponding modifications made to Buchberger's algorithm. In Section 5.3 we discuss the applications to systems biology and the game Sudoku. In section 5.4 we look at timings for *gbBoolean* versus two different methods for computing Boolean Gröbner bases in Macaulay2.

5.2 Algorithm

5.2.1 Binary representation

We define a Boolean polynomial to be the coset representative in the quotient ring $QR = \mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 + x_1, x_2^2 + x_2, \ldots x_n^2 + x_n\rangle$ that is multilinear. In other words, a polynomial in $\mathbb{F}_2[x_1, \ldots, x_n]$ that is linear in each variable. Because of this linearity, we can represent a Boolean monomial in n variables as a binary string of length n where a 1 in the ith position represents that the variable x_i is present in the monomial. A polynomial can be represented as a list of these binary strings of length n. We call these lists Binary Representations of Polynomials (BRPs). For example, $xy + z$ in the ring $R = \mathbb{F}_2[x, y, z]/\langle x^2 + x, y^2 + y, z^2 + z\rangle$ as a BRP is $\{\{110\}, \{001\}\}$. Since we do not need to store coefficients or degrees, arithmetic on BRPs is bitwise rather than symbolically. Since we are computing over $\mathbb{F}_2$, two identical monomials sum to zero. Addition of two polynomials is concatenation of the lists, removing duplicate pairs of strings. For example, in the ring R above $xy + yz + 1$ added to $xz + yz + 1$ is $xy + xz$. As BRPs this becomes $\{\{101\}, \{011\}, \{000\}\} + \{\{110\}, \{101\}, \{000\}\} = \{\{101\}, \{011\}, \{000\}, \{110\}, \{101\}, \{000\}\}$. The duplicate pairs are removed leaving $\{\{110\}, \{101\}\}$ as the sum.

To multiply two monomial BRPs, we use the binary operation OR. Since $x_i^2 = x_i$ for every i, if x_i appears in either monomial, then x_i will appear in the product. In the ring R above, the product of xy and xz is xyz. As BRPs, $\{110\}\text{OR}\{101\} = \{111\}$. To multiply a monomial by a polynomial, we multiply the monomial by each monomial in the polynomial, and then we add the monomials by concatenating the binary strings, removing duplicate pairs of monomials.

In order to compute the S-polynomial of two polynomials, we need to find the least common multiple of two monomials. The least common multiple of two monomial BRPs is again bitwise OR on the BRPs. In fact, the operation of least common multiple in this Boolean ring is multiplication.

Next we need to divide one monomial by another. First, we check divisibility. If a monomial m_1 is divisible by a monomial m_2, then any variable appearing in m_2 must also appear in m_1. Therefore m_1 is divisible by m_2 if and only if componentwise $m_1 > m_2$. Once divisibility has been established, if m_1 is divisible by m_2, then $m_1 \div m_2 = m_1$ XOR m_2, where XOR is the binary exclusive OR. For example, let $m_1 = \{1, 0, 1\}$ and $m_2 = \{1, 1, 0\}$. Then m_1 is not divisible by m_2 since the second component of m_1 is not greater than the second component of m_2. Thus m_1 is divisible by $m_3 = \{1, 0, 0\}$. Since divisibility has been established, we can perform $m_1 \div m_3 = m_1$ XOR $m_3 = \{0, 0, 1\}$.

We can use Buchberger's first criterion [11] to eliminate S-pairs whose leading terms are relatively prime. To determine if two BRPs are relatively prime, we compare the minimum value for each variable in the monomial. If the minimum for each variable is 0, then the monomials are relatively prime. Now we have enough operations to execute our modified Buchberger algorithm for BRPs.

5.2.2 Buchberger's algorithm

Given a set of Boolean polynomials, F, in $QR = \mathbb{F}_2[x_1, \ldots, x_n]/I$, where $I = \langle x_1^2 + x_1, x_2^2 + x_2, \ldots x_n^2 + x_n \rangle$, we want to compute a Gröbner basis for the ideal generated by F using a Buchberger algorithm modified for BRPs. When computing the S-polynomials, it is necessary to compute S-polynomials for the generating polynomials in F as well as the quadratic polynomials in I. A difficulty is immediately encountered since the quadratic polynomials are not BRPs. However, the difficulty is bypassed using the following result.

Theorem 5.2.1. *The S-polynomial of any multilinear Boolean polynomial and a quadratic field polynomial is always multilinear.*

Proof. Let f be a Boolean polynomial with leading term $\mathrm{lt}(f)$. Consider f in the ring $\mathbb{F}_2[x_1, \ldots, x_n]$. If the variable x_i is not contained in $\mathrm{lt}(f)$, then the leading terms of the two polynomials are relatively prime, and the S-polynomial of f and $x_i^2 + x_i$ will reduce to 0 by Buchberger's first criterion [11]. Therefore, we only consider quadratic polynomials in I whose variable in the leading term is contained in $\mathrm{lt}(f)$. Now suppose x_i divides $\mathrm{lt}(f)$. We compute the S-polynomial in the ring $\mathbb{F}_2[x_1, \ldots, x_n]$ and then reduce modulo I. Let $f = \mathrm{lt}(f) + \mathrm{tail}(f)$. Then $S(f, x_i^2 + x_i) = x_i f - (\frac{\mathrm{lt}(f)}{x_i})(x_i^2 + x_i) = x_i \mathrm{lt}(f) + x_i \mathrm{tail}(f) - x_i \mathrm{lt}(f) - \mathrm{lt}(f) = \mathrm{lt}(f) + x_i \mathrm{tail}(f)$. which can be written as the sum and product of BRPs. □

Therefore, no special encoding for quadratic polynomials in I is necessary. We can carry out all the necessary operations for the modified Buchberger algorithm on BRPs.

5.2.3 Implementation

The package *BooleanGB* contains the algorithm *gbBoolean*, which is an implementation of Buchberger's algorithm for binary representations of Boolean polynomials over the quotient ring, $QR = \mathbb{F}_2[x_1, x_2, \ldots, x_n]/\langle x_1^2 + x_1, x_2^2 + x_2, x_n^2 + x_n\rangle$. The algorithm is implemented in *C++* and is part of the Macaulay2 engine. *gbBoolean* computes a reduced Gröbner basis in lexicographic order for an ideal of Boolean polynomials in QR. All computations are done bitwise instead of symbolically, by representing a monomial as bits in an integer and a polynomials as a list of monomials. On a 64-bit machine, the algorithm works in the ring with up to 64 indeterminates. The algorithm uses Buchberger's first and second criteria [11] as described in [16, p. 109]. Here is an example. 5.2.2.

```
i1 : loadPackage "BooleanGB";

i2 : QR = ZZ/2[x,y,z]/ideal(x^2-x, y^2-y, z^2-z);

i3 : gbBoolean ideal(x*y+z);

o3 = ideal (x*y + z, y*z + z, x*z + z)

o3 : Ideal of QR
```

5.3 Applications

Problems involving Boolean polynomials include exact cover problems, satisfiability problems and problems in systems biology. In some cases, it may be possible to turn a system of non-Boolean polynomials into a Boolean system. This involves dramatically increasing the number of variables. But in the example of Sudoku, described below, the time saved by the bitwise computations outweighs the increase in variables.

5.3.1 Boolean models in systems biology

Logical models are widely used in systems biology. In general such models correspond to polynomial dynamical systems, and in particular, logical models with binary variables result in Boolean polynomials [68, 98]. Key dynamical features of logical models such as fixed points correspond to the points in algebraic varieties generated by the polynomials describing the model. To assure that *gbBoolean* is efficient on "practical" ideals, we translate all binary logical models in the GINsim repository to Boolean polynomial systems using ADAM, and compute the Gröbner basis of the ideal describing the fixed points [45, 69, 44].

GINsim (Gene Interaction Network simulation) is a tool for the modeling and simulation of genetic regulatory networks. Models in this repository are logical models of previously published regulatory networks. e.g., the network of the ERBB receptor-regulated G1/S transition or the network controlling Th1/Th2 cell differentiation. For all logical models in the repository, *gbBoolean* is faster than current Macaulay2 implementations. Run-times for five such networks, labeled TCR signaling pathway, Th1/Th2 cell differentiation, Mammalian cell cycle, G1/S transition and Yeast cell cycle are depicted in Table 5.1 in Section 5.4.

5.3.2 Sudoku

Sudoku is a popular game played on a 9×9 grid where the numbers 1-9 are filled in so that each number appears exactly once in each row column and 3×3 block. Assigning one variable to each cell in the grid, we get a polynomial representation of the constraints (See [6] and [29] for more details). For ease of demonstration, we use Shidoku as an example. Shidoku is played with the same rules as Sudoku, but on a 4×4 grid with the numbers 1-4.

For Shidoku, 16 variables are used that each take only the values 1-4. We represent this fact, together with the constraints of the game, in a total of 40 polynomials. If we consider the ideal generated by these polynomials in $\mathbb{Q}[x_1, \ldots, x_{16}]$ with the lexicographical ordering, we cannot compute the Gröbner basis for the ideal in Macaulay2 in a reasonable amount of time. We can, however, convert the problem to a system of Boolean polynomials and use *gbBoolean* to compute the Gröbner basis.

The Boolean system has 64 variables, representing each of the possible values 1-4 for each of the 16 cells. We consider the ideal of the polynomials in $QR = \mathbb{F}_2[x_1, x_2, \ldots, x_n]/\langle x_1^2 + x_1, x_2^2 + x_2, x_n^2 + x_n \rangle$. The Boolean system gives us a total of 256 polynomials to represent the Shidoku constraints. As seen in Section 5.4 below, *gbBoolean* computes the Gröbner basis for the Boolean ideal in just 4.1 seconds.

5.4 Results

We test *gbBoolean* on several random ideals and compare the run times with the standard Gröbner basis implementation in Macaulay2. In addition, we also test published Boolean logical models stemming from biological systems and compute the basis of the ideals corresponding to key dynamic features. Finally, we test the algorithm on Sudoku, a problem that can be described by a system of Boolean polynomials.

We compare *gbBoolean* to the symbolic computation in lexicographic order. Since the graded reverse lexicographic (gRL) monomial order usually yields the fastest calculation, we also compare *gbBoolean* to the combined times of computing a basis in gRL and lifting it to the quotient ring with lexicographic order.

Table 5.1 shows the run-times on a 3.06 GHz Intel Core 2 Duo MacBook Pro. In almost all examples, *gbBoolean* is faster than current implementations in Macaulay2. Cumulatively, *gbBoolean* is about four times faster (0.27614741%) than the Gröbner basis calculation in the ring with gRL order, and 50 times faster (0.017562079%) than the calculation in the ring with lexicographic order.

	Variables	QR Lex	QR Lift	gbBoolean
random 1	64	220.681	19.0562	4.4871
random 2	20	0.002174	0.003237	0.00001
random 3	20	0.0032	0.004926	0.000009
random 4	30	0.003438	0.005541	0.00001
TCR signaling pathway [57]	40	0.007104	0.012106	0.000208
Th1/Th2 cell differentiation [79]	12	0.001946	0.003759	0.000009
Mammalian cell cycle [27]	10	0.002007	0.003859	0.000012
G1/S transition [85]	20	0.002609	0.003017	0.00001
Yeast cell cycle [61]	11	0.002593	0.002013	0.00009
Shidoku	64	470.703	19.6797	4.11442

Table 5.1: Run times in seconds for random, biological, and Shidoku examples

5.5 Conclusion

Computing Gröbner bases for Boolean polynomials bitwise rather than using standard implementations can speed up computation time considerably. Even in cases such as Shidoku, while converting the polynomials from coefficients in $\mathbb{Q}$ to a new set of polynomials with coefficients in $\mathbb{F}_2$ increases the number of variables from 16 to 64, the time to compute the Gröbner basis decreases. The package *BooleanGB* in Macaulay2 is generally faster for computing a Gröbner basis for Boolean polynomials than the standard algorithm in Macaulay2. Further improvements might be achieved by implementing the sugar strategy [32] or other monomial orders.

Acknowledgments

The authors thank Mike Stillman for adding the *C++* implementation to the M2 engine and for his improvements of the implementation. Furthermore, the authors thank Samuel Lundqvist, Amelia Taylor, Dan Grayson, and Reinhard Laubenbacher for stimulating discussions, insightful comments, and organizing several Macaulay2 workshops.

Chapter 6

Boolean models of bistable biological systems

This manuscript is published in the *Journal of Discrete and Continuous Dynamical Systems - Series S*. It is joint work with Reinhard Laubenbacher. FH contributed the algorithm, examples, and applications.

Abstract

This paper presents an algorithm for approximating certain types of dynamical systems given by a system of ordinary delay differential equations by a Boolean network model. Often Boolean models are much simpler to understand than complex differential equations models. The motivation for this work comes from mathematical systems biology. While Boolean mechanisms do not provide information about exact concentration rates or time scales, they are often sufficient to capture steady states and other key dynamics. Due to their intuitive nature, such models are very appealing to researchers in the life sciences. This paper is focused on dynamical systems that exhibit bistability and are described by delay equations. It is shown that if a certain motif including a feedback loop is present in the wiring diagram of the system, the Boolean model captures the bistability of molecular switches. The method is applied to two examples from biology, the lac operon and the phage λ lysis/lysogeny switch.

6.1 Introduction

Since the discovery of the first gene regulatory network, the lactose metabolism network in the bacterium *E. Coli* by Jacob Monod, [49], such networks have been modeled extensively, traditionally with differential equations. But other modeling techniques like Boolean networks [4], stochastic models [96], Petri nets, or Bayesian networks [28] have also been used successfully. Work by Kauffman in 1969 [56] suggested that gene networks behave like Boolean switching nets and therefore

Boolean networks are suitable to model them. Using a Boolean network model, it has been shown, for instance, that in some cases the network topology and the logic of the interactions among the different molecular species is sufficient to determine the qualitative dynamics of the network, without taking into account the detailed kinetics [4]. In cases where not enough information about the kinetic parameters is available to build a detailed continuous model, one often still has enough information to build a Boolean network model which can provide important information.

The purpose of this paper is to show for a particular family of continuous models that they can be approximated by a Boolean network model that retains the key information about model dynamics. Our intent is to demonstrate that Boolean models can be used to study a variety of structural and dynamic features of biological and other systems that have traditionally been modeled using continuous models. They have the added advantage that they are intuitive and do not require much mathematical background. This makes them particularly suitable for use in the life sciences. The focus of this paper is on two features: bistability and time delays. We give a general algorithm how to approximate a dynamical system given by a system of ordinary delay differential equations by a Boolean network model and validate it with two well-known examples from systems biology. The relation between discrete and continuous models has been examined before, e.g., [23] describes how a logical network can be used to create the analogous differential equations. Other methods allow to systematically create a continuous model from a discrete dynamical system [64], or to reduce the model space by using a finite state machine [15]. We briefly review the main concepts related to Boolean network models.

6.1.1 Introduction to Boolean models

We use a time discrete deterministic Boolean network with synchronous update. In a Boolean model, a variable can only be in the "on" (1) or "off" (0) state. In systems biology applications, each variable typically corresponds to a molecular entity, e.g., the concentration of a gene product such as mRNA or proteins. A Boolean network of n variables consists of an update function $f = (f_1, \ldots, f_n) : \mathbb{F}_2^n \to \mathbb{F}_2^n$, also called a transition function. The dynamics of the system are generated deterministically by iteration of the transition function f. It can be visualized by its phase space, which shows all possible states and their transitions. A cycle in the phase space is called a limit cycle; if it has length one, a fixed point. Fixed points of a discrete system are the equivalent to steady states of a continuous system.

In the dependency graph, also referred to as wiring diagram, each variable is a vertex and an edge from variable x_i to x_j is drawn if x_i shows up in the local transition function f_j. A directed cycle in the dependency graph is called a feedback loop.

Example

Consider the following Boolean network with three variables:

$$f_1 = ((\neg x_1) \wedge x_3)$$
$$f_2 = (x_1 \wedge x_3)$$
$$f_3 = ((x_1 \wedge x_2) \vee x_3).$$

The phase space of this function is depicted in Fig 6.1. The network has a limit cycle of length two because the state (1 0 1) transitions into (0 1 1), which turns back to (1 0 1). It also includes the fixed point (0 0 0). Discrete Visualizer of Dynamics (DVD tool) [58] was used to create the

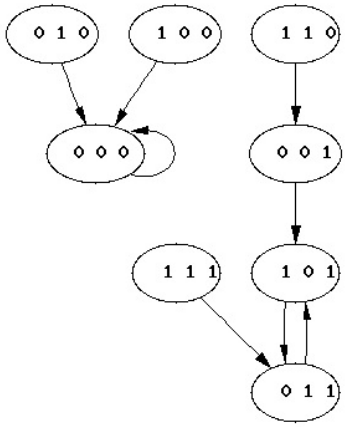


Figure 6.1: Phase space of $f = (\neg x_1 \wedge x_3, x_1 \wedge x_3, (x_1 \wedge x_2) \vee x_3)$

phase space. It calculates the number and length of limit cycles and fixed points, as well as the trajectories from any initial state. It also generates a graph of the phase space along with the dependency graph.

6.2 Approximation by discrete models

We begin by describing several relevant features of biochemical networks that a model needs to capture and describe in each case how this is done in the Boolean network case. We use lower case letters for continuous variables and upper case letters for discrete ones.

Dilution and degradation

It is common in biochemical networks that the concentration of a substance X decreases with time because of dilution and degradation. In a differential equations model this is usually accounted for

with a negative degradation term:

$$\frac{dx}{dt} = \cdots - \gamma x.$$

In the discrete model, for simplicity it is assumed that the degradation rate for a substance is either vanishingly small and can therefore be ignored, or the substance is degraded below the threshold for discretization after two discrete time steps.

To model the decrease of concentration of a substance by dilution and degradation, which is modeled in the continuous case by a negative degradation term for x, a new variable X_{old} is introduced. If X_{old} is "on," then any quantity of X that is available has already been reduced once by dilution and degradation, and if no new substrate is produced, X will be "off" in the next time step.

Time delay

In gene regulatory networks, time delays are often caused by transcription and translation. Dependence of a variable x on the concentration of a substance y time τ ago can be described with a delay differential equations model

$$\frac{d}{dt}x(t) = \ldots\ y(t-\tau)\ \ldots$$

In a time discrete model, further variables are needed to model this delay. One has to choose the length $\bar{t}$ of a discrete time step. If the delay $\tau = \bar{t}$, then X is chosen to depend on Y_1 and Y_1 depends on Y.

$$\begin{aligned} f_X &= \ldots Y_1 \ldots \\ f_Y &= \ldots \\ f_{Y1} &= Y \end{aligned}$$

If $\tau > \bar{t}$, additional variables have to be introduced.

Distinguishing between low, medium, and high concentrations

To differentiate between three states for a variable X in a Boolean system, a second variable X_{high} is introduced. By doing so one can make a distinction between low, medium, and high concentration of X: if X is on, the concentration is a least medium, whereas X_{high} is only on, if the concentration is high.

Bistability

If previous experiments or computations have shown that the system exhibits bistable behavior depending on the concentration of a stimulus X, it is necessary to distinguish in the model between different concentrations of X: the concentrations for which there is a unique steady state for the system and for which there are multiple possible steady states. Without this distinction, a

Boolean model could not capture bistability. In the lac operon, a medium allolactose concentration leads to bistability, therefore the discrete model must differentiate between low, medium, and high concentration of allolactose. The stimulus X is the parameter that drives a system in and out of

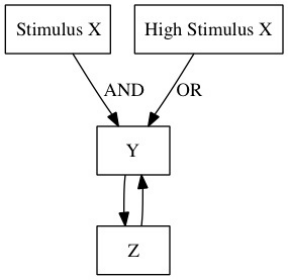


Figure 6.2: Necessary motif in dependency graph for bistability

the bistable region; within the bistable region the previous state determines the resulting steady state. The stimulus acts on one or more variables. Usually X_{high} is part of an OR statement in the transition function (if the stimulus is present in high concentration, it should overrule everything and turn the system on), whereas X is part of an AND statement (under its presence other factors might prevent the system from turning on). The state of Z (this could be a set of variables or a single variable) indicates whether the system was in a preinduced state before the stimulus concentration was changed or not, depending on the input from Z:

$$Y = (Z \wedge X) \vee X_{\text{high}}.$$

To assure that Z reflects whether the system was preinduced, Z needs to receive input from Y. This leads to a system that can be reduced to a system as the one shown in Fig. 6.2. As long as the dependency graph of a Boolean model can be reduced to Fig. 6.2, it is guaranteed, that the discrete network captures the bistability of the system correctly.

6.2.1 Example of building Boolean model

We demonstrate the above method on the following generic delay differential equations model:

$$\begin{aligned} \frac{dx}{dt} &= y(t-\tau) - 1 \\ \frac{dy}{dt} &= x(t-\tau) - 1. \end{aligned} \tag{6.1}$$

The steady state of this system is $(x, y) = (1, 1)$. When translating (6.1) into a discrete model, the first step is to generate Boolean equations for x and y,

$$\begin{aligned} f_x &= y \\ f_y &= x. \end{aligned}$$

To account for the degradation term -1 in the equation for y, a new variable y_{old} is introduced as outlined in section 6.2, and, similarly, x_{old}. We use x_1 and y_1 for the delay τ , as described in section 6.2. This leads to the following system

$$\begin{aligned} f_x &= y_1 \vee (x \wedge \neg x_{\text{old}}) \\ f_{x_1} &= x \\ f_{x_{\text{old}}} &= \neg y_1. \\ f_y &= x_1 \vee (y \wedge \neg y_{\text{old}}) \\ f_{y_1} &= y \\ f_{y_{\text{old}}} &= \neg x_1. \end{aligned} \tag{6.2}$$

Part of the phase space of the network is depicted in Fig. 6.3. It clearly shows $(x, y) = (1, 1) = (1, \_, 1, \_, \_)$ as its fixed point, just as expected from the solution of the continuous model. In

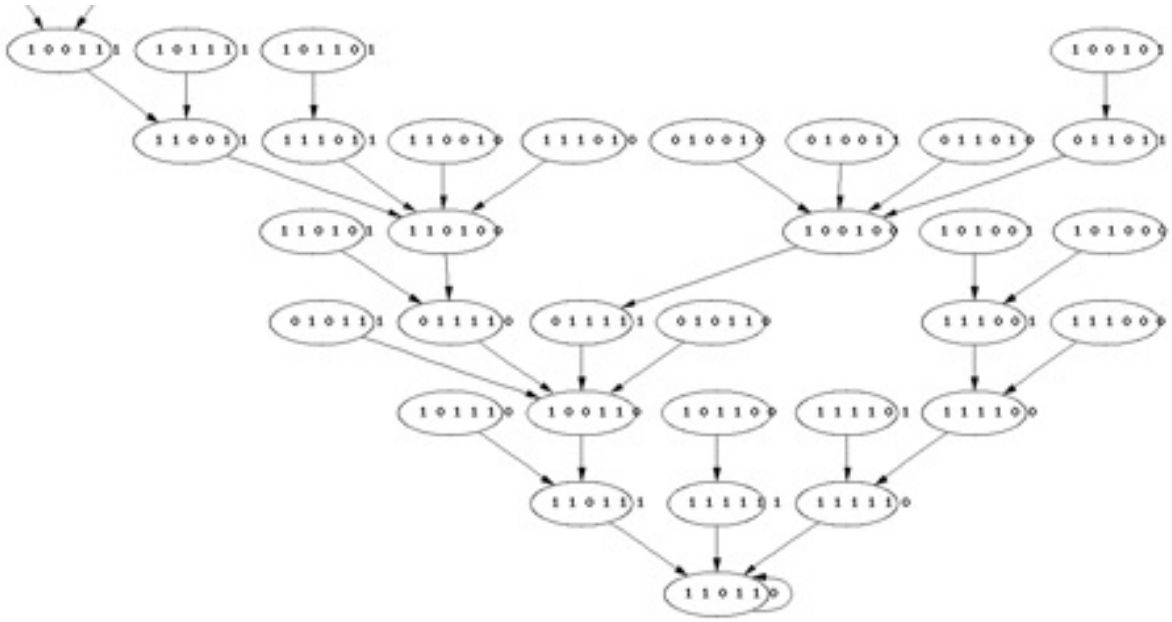


Figure 6.3: Phase space of network (6.2)

the continuous system values for x and y could be negative, this is represented in the fixed point $(x, y) = (0, 0) = (0, \_, 0, \_, \_)$ of the discrete system.

In the next two sections we apply the above methods to two well-studied examples of gene regulatory networks, for which there exist delay differential equations models.

6.3 The *Lac* operon

The lac operon is required for the transport and metabolism of lactose in *Escherichia coli*. It has been studied extensively, and Novick [72] showed in the 1950s that bistability is observed with artificial inducers. He observed that "preinduced" bacteria, that is bacteria grown in a high concentration of inducer, are able to maintain a high internal inducer concentration if subsequently grown in a low external inducer concentration. Novick, [72] calls this the "preinduction effect". He observed, however, that if preinduced bacteria are transferred to a medium with no inducer, enzyme synthesis ceases immediately. The minimum concentration in which the high rate of synthesis of a preinduced culture can be maintained is called the *maintenance concentration*.

Novick's experiments have led to various mathematical models of the lac operon whose steady state solutions show bistability, for example Yildirim [104], Wong[101], Boer [17].

We first give a brief overview of the functionality of the lac operon depicted in Fig. 6.4. In the absence of glucose for cellular metabolism, extracellular lactose is transported into the cell, either actively by permease or passively through diffusion. Inside the cell, β-galactosidase breaks up lactose into glucose, galactose, and allolactose. Allolactose binds to the repressor, which is usually bound to the operator region where it inhibits the transcription process, therefore the transcription process can proceed. RNA polymerase initiates transcription of the structural genes to produce mRNA, which is then translated into proteins, permease, and β-galactosidase.

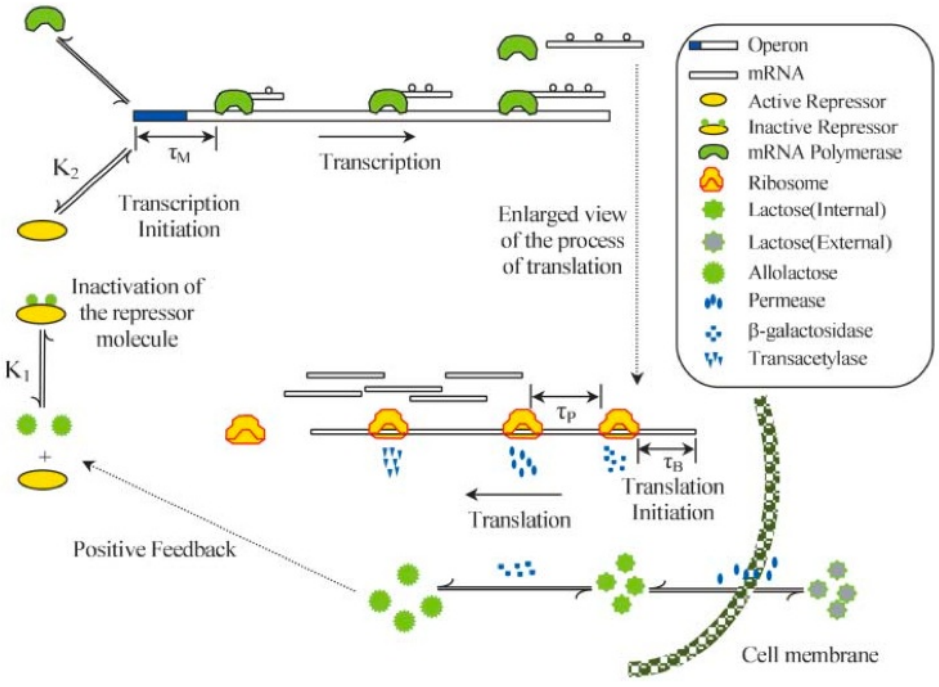


Figure 6.4: Schematic representation of the lactose operon regulatory system from [1]

The system contains a positive feedback loop, with an increase in the concentration of allolactose, mRNA concentration increases, and therefore more β-galactosidase and permease are produced, which in turn leads to more internal lactose and allolactose. Positive (or negative) feedback loops are very common in biological systems because they allow for a rapid increase in concentration. In the case of the lac operon, the enzyme induction is an "all-or-none phenomenon", with the first permease molecule extracellular lactose is transported into the cell, increasing the inducer concentration, and therefore increasing the probability of the appearance of another permease molecule, [72].

6.3.1 Boolean model of Lac operon

We derive a Boolean network for the gene regulatory network from the continuous model developed in [104] consisting of 5 equations. All its parameters were estimated from the biological literature and the equations were numerically solved for their steady states. The delay differential equations model predicts a bistable region for a medium external inducer concentration, which is in accordance to what has been observed experimentally by [72] and [14]. As in the continuous model, we take the following 5 variables for the Boolean model into account: messenger RNA M, β-galactosidase B, allolactose A, permease P, and internal lactose L.

We will show how to derive the Boolean equation for messenger RNA, the process for the other variables is similar and can be found in the appendix. In [104], the rate of change of mRNA is given by the following equation:

$$\begin{aligned}\frac{d}{dt}M(t) =&\alpha_M \frac{1+K_1(e^{-\mu\tau_M}A_{\tau_M})^n}{K+K_1(e^{-\mu\tau_M}A_{\tau_M})^n}+\Gamma_0-\tilde{\gamma}_M M\\ =&\alpha_M \frac{1+K_1(e^{-\mu\tau_M}A(t-\tau_M))^n}{K+K_1(e^{-\mu\tau_M}A(t-\tau_M))^n}+\Gamma_0-(\gamma_M+\mu)M,\end{aligned}$$

where n is the number of molecules of allolactose required to inactivate the repressor. $\frac{d}{dt}M(t)$ depends on A at time $(t-\tau_M)$ and $-(\gamma_M+\mu)M$ models the loss caused by dilution and degradation.

Under the presence of allolactose A, $\frac{d}{dt}M$ is non-negative, so the Boolean equation for mRNA is

$$f_M = x_{A_\tau},$$

where A_τ describes the allolactose concentration A time τ_M ago.

If mRNA is present and no new mRNA is produced in the next time step, the concentration will decrease according to the degradation rate. To capture this in the Boolean model, we introduce the artificial variable M_{old} as described in section 6.2. M_{old} is "on", if allolactose is absent, because then no new mRNA was produced. If no mRNA was produced for 2 time steps, its concentration is too low and we consider M to be "off". This results in the following equations:

$$\begin{aligned}f_M &= x_{A_\tau} \vee (x_M \wedge \neg x_{M_{\text{old}}})\\ f_{M_{\text{old}}} &= \neg x_{A_\tau}.\end{aligned}$$

mRNA depends on A time τ ago, so a new variable $A1$ is introduced, as described in 6.2 and we set

$$\begin{aligned} f_M &= x_{A1} \vee (x_M \wedge \neg x_{M_{\text{old}}}) \\ f_{M_{\text{old}}} &= \neg x_{A1} \\ f_A &= \dots \\ f_{A1} &= x_A. \end{aligned}$$

We discretize the external inducer concentration L_e to be "on", if it is above the minimal maintenance concentration. The high external inducer concentration $L_{e_{\text{high}}}$ is "on", if the concentration is at least (*), as shown in Fig 6.5.

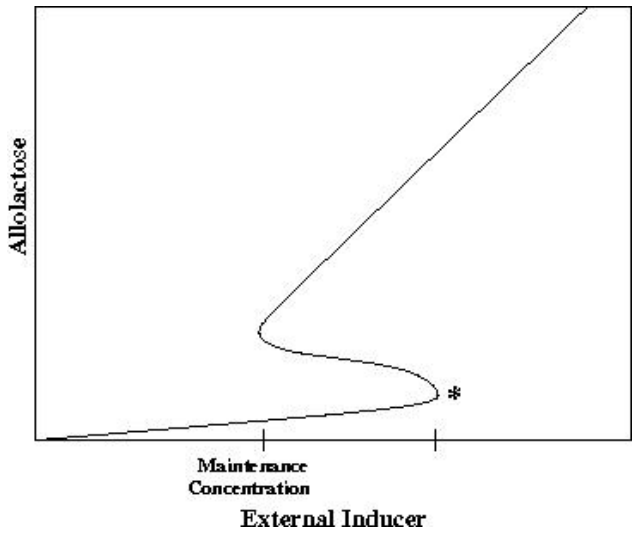


Figure 6.5: Discretization of External Inducer

Using the method described above, accounting for degradation and delays, we derive the following Boolean model:

$$\begin{aligned} f_M &= x_{A1} \vee (x_M \wedge \neg x_{M_{\text{old}}}) \\ f_{M1} &= x_M \\ f_{M2} &= x_{M1} \\ f_{M3} &= x_{M2} \\ f_{M_{\text{old}}} &= \neg x_{A1} \\ f_B &= x_{M2} \vee x_B \\ f_A &= (x_B \wedge x_L) \vee (x_L \wedge x_{L_{e_{\text{high}}}}) \\ &\quad \vee (x_A \wedge \neg x_{A_{\text{old}}}) \\ f_{A1} &= x_A \end{aligned}$$

$$\begin{aligned} f_{A_{\text{old}}} &= \neg x_B \vee \neg x_L \\ f_L &= (x_P \wedge x_{L_e}) \vee x_{L_{e_{\text{high}}}} \\ &\quad \vee [x_L \wedge \neg (x_{L_{\text{old}}} \wedge (x_P \vee x_B))] \\ f_{L_{\text{old}}} &= \neg x_P \vee \neg x_{L_e} \\ f_P &= x_{M3} \vee (x_P \wedge \neg x_{P_{\text{old}}}) \\ f_{P_{\text{old}}} &= \neg x_{M3} \\ f_{L_e} &= x_{L_e} \vee x_{L_{e_{\text{high}}}} \\ f_{L_{e_{\text{high}}}} &= x_{L_{e_{\text{high}}}}. \end{aligned}$$

Table 6.1: Fixed Points of Lac Operon

	M	M_1	M_2	M_3	M_{old}	B	A	A_1	A_{old}	L	L_{old}	P	P_{old}	L_e	$L_{e_{\text{high}}}$
1	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0
2	0	0	0	0	1	0	0	0	1	1	1	0	1	0	0
3	0	0	0	0	1	1	0	0	1	0	1	0	1	0	0
4	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0
5	0	0	0	0	1	1	0	0	1	0	1	0	1	1	0
6	0	0	0	0	1	0	0	0	1	0	1	0	1	1	0
7	1	1	1	1	0	1	1	1	0	1	1	1	0	1	0
8	1	1	1	1	0	1	1	1	0	1	1	1	0	1	1

We use the DVD simulation software, see section 6.1.1, to calculate that the system has 8 fixed points and no limit cycles.

As expected, a low external inducer concentration (last two variables are set to 0) drives the system to fixed points corresponding to the operon in the "off" state, points $1-3$ in table 6.1. If the concentration of the artificial external inducer is medium, but not high, the system results in the fixed points $4-7$ in table 6.1. With high external inducer concentration, the system settles in the remaining fixed point which corresponds to an induced operon.

For a medium inducer concentration, fixed point 4 and 5 are biologically not meaningful, because they correspond to states in which only the internal lactose or β-galactosidase concentration is present but no other substance. Fixed point 6 corresponds to the "off" operon, fixed point 7 to the "on" operon.

To show bistability, we analyze the behavior of the fixed points as we change the concentration rates of the external inducer. If we start in a state corresponding to fixed point 1 and increase the inducer concentration to a medium concentration, the system settles down in fixed point 6, corresponding to the "off" operon. If we start with state 8, the system settles down in fixed point 7, corresponding to the "on" operon. This is exactly what we expect from the solution of the delay differential equation: under a medium concentration of external inducer, the steady state depends on whether the cell was preinduced or not.

6.4 Lambda phage

The virus lambda phage (phage λ) is a bacteriophage that infects *Escherichia coli*. After injecting its DNA into the host, the phage can enter the lytic pathway where it alters the host DNA to produce phage particles and then lyses the host cell, or it can enter the lysogenic pathway, where it is duplicated with every cell division and harmless until the cell is under stress, then it enters its lytic pathway. A schematic representation of the phage λ switch is shown in Fig. 6.6. Interestingly, the lysogenic state is more stable than the genome itself [7]. A comprehensive explanation of the lambda phage switch can be found in [77].

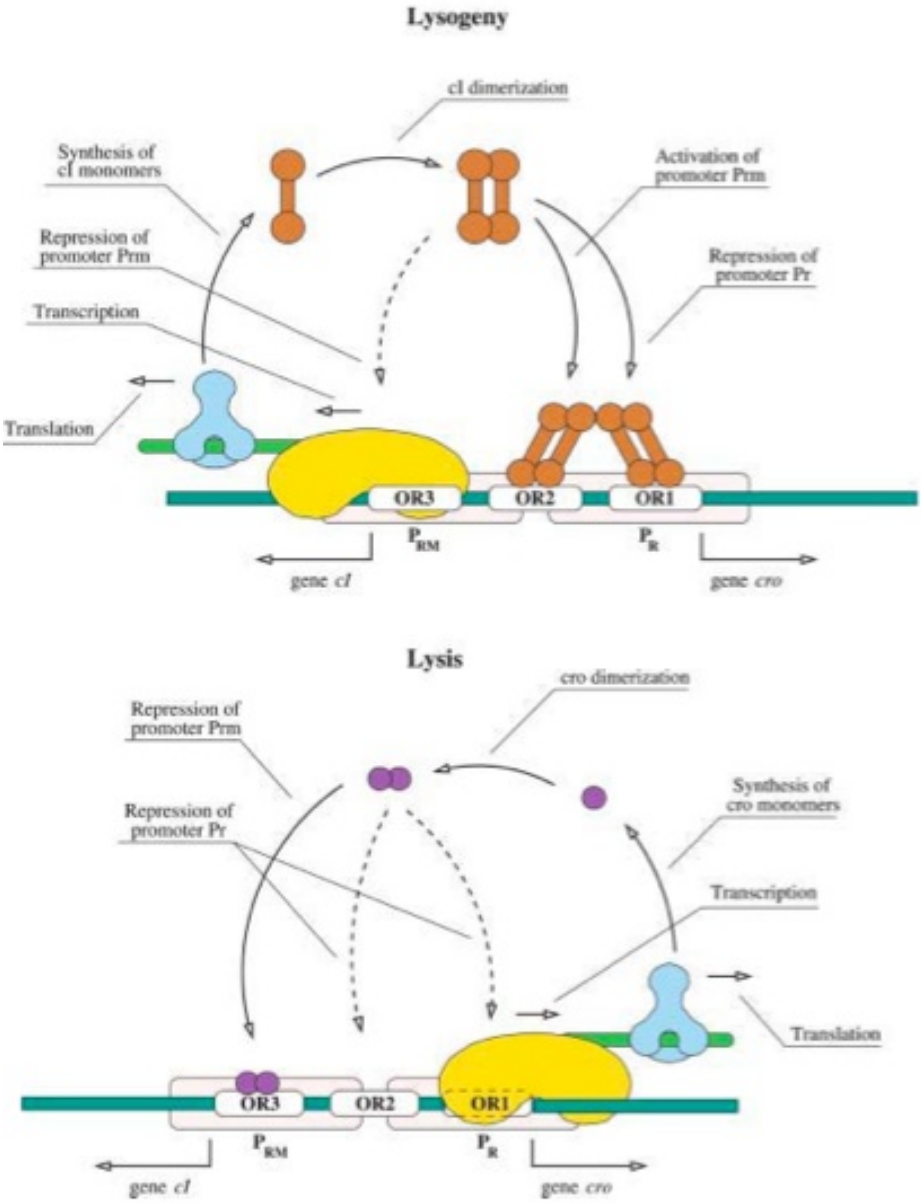


Figure 6.6: The Life Cycle of Lambda Phages from [2]

In [86] a delay differential equations model for the switch between lysogenic and lytic state of phage lambda is presented. This continuous model consists of four equations for the rate of change of the concentrations of cI and cro messenger RNA molecules, cI and Cro monomers and of two equations for the dimer concentrations of $[CI_2]$ and $[Cro_2]$.

Bistable behavior depends on the degradation rate of cI, γ_{cI}. Numerical calculations show that the range for γ_{cI} in which the systems has two fixed points, one corresponding to lysogeny, the other to lysis, is $0.0\text{min}^{-1} < \gamma_{cI} < 0.35\text{min}^{-1}$. If the degradation rate of cI is zero, the phage enters the lysogenic pathway, if it is above 0.35min^{-1}, the lytic pathway. For a medium degradation rate of cI, phage lambda can enter the lytic or lysogenic pathway, so one needs to distinguish between three different degradation rates of γ_{cI}. Note that for the lambda phage a degradation rate drives the system in the bistable region, whereas for the lac operon an external inducer has that role. To distinguish between three states of γ_{cI} in the Boolean model, the variable $\gamma_{\text{cI}_{high}}$ is introduced. To avoid wrong input, e.g., $\gamma_{\text{cI}} = 0$ and $\gamma_{\text{cI}_{high}} = 1$, γ_{cI} is turned on, whenever $\gamma_{\text{cI}_{high}} = 1$.

The delay caused by transcription is only 0.005 min^{-1}, whereas the delays caused by the translation of the monomers cI and Cro are 0.06 and 0.24 min^{-1}. Therefore delay caused by transcription is neglected in the model.

For a high degradation rate, no cI monomers are left after one time step. We assume, that a small concentration of monomers is generated also if the Boolean expression that generates cI is zero. As a consequence, if the degradation rate for cI is zero, we assume that sufficient cI monomers are produced to set $x_{CI} = 1$ after one time step.

This results in the following Boolean model.

$$
\begin{aligned}
f_{M_{\text{cI}}} &= x_{CI_T} \vee \overline{x_{Cro_T}} \vee (x_{M_{\text{cI}}} \wedge \overline{x_{M_{\text{cI}_{\text{old}}}}}) \\
f_{M_{\text{cI}_{\text{old}}}} &= \overline{x_{CI_T}} \wedge x_{Cro_T} \\
x_{M_{\text{cI}_1}} &= x_{M_{\text{cI}}} \\
f_{M_{\text{cro}}} &= \overline{x_{CI_T}} \vee (x_{M_{\text{cro}}} \wedge \overline{x_{M_{\text{cro}_{\text{old}}}}}) \\
f_{M_{\text{cro}_{\text{old}}}} &= x_{CI_T} \\
x_{M_{\text{cro}_1}} &= x_{M_{\text{cro}}} \\
x_{M_{\text{cro}_2}} &= x_{M_{\text{cro}_1}} \\
f_{CI_T} &= \overline{\gamma_{\text{cI}_1}} \vee (\overline{\gamma_{\text{cI}_{\text{high}}}} \wedge (x_{M_{\text{cI}_1}} \vee (x_{CI_T} \wedge (\overline{x_{CI_{T_{\text{old}}}}} \vee \overline{\gamma_{\text{cI}}})))) \\
f_{CI_{T_{\text{old}}}} &= \overline{x_{M_{\text{cI}_1}}} \vee \gamma_{\text{cI}_{\text{high}}} \\
f_{Cro_T} &= x_{M_{\text{cro}_2}} \vee (x_{Cro_T} \wedge \overline{x_{Cro_{T_{\text{old}}}}}) \\
f_{Cro_{T_{\text{old}}}} &= \overline{x_{M_{\text{cro}_2}}} \\
f_{\gamma_{\text{cI}}} &= \gamma_{\text{cI}} \vee \gamma_{\text{cI}_{\text{high}}} \\
f_{\gamma_{\text{cI}_1}} &= \gamma_{\text{cI}} \\
f_{\gamma_{\text{cI}_{\text{high}}}} &= \gamma_{\text{cI}_{\text{high}}}.
\end{aligned}
$$

DVD simulation software [58] is used to calculate the fixed points and generate the dynamics in the bistable region. If the degradation rate is 0.0min^{-1}, in the model the last three variables are off, the

system results in the fixed point [10101001001000] which corresponds to $[M_{cI}CI]$, a high cI and low Cro concentration which means that the system is in the lysogenic state. If the degradation rate is medium, in the model the third last variable is on and the last two variables are off, there are two fixed points, [01010110110110] and [10101001001110], corresponding to $[M_{cro}Cro]$ and $[M_{cI}CI]$, respectively, lysogenic and lytic pathway. With a high degradation rate, in the model only the last variable of the last three variables is on, the system settles in the fixed point [01010110110111], representing $[M_{cro}Cro]$. This means that the system is in the lytic state. All four fixed points are in accordance with the results found numerically and the switching behavior observed experimentally.

To investigate the bistable region, we start the model with an initialization that represents the lysogenic state. Increasing the degradation rate to a medium level results in the fixed point [10101001001110], which is still the lysogenic state. Starting the model from a lytic state and decreasing the rate results in the fixed point [01010110110110], representing the lytic state. Decreasing the rate even further finally results in the fixed point corresponding to the lysogenic state.

6.5 Discussion

The results presented in this paper show that biochemical networks that exhibit bistability can be modeled successfully using a Boolean network model, incorporating delays for variables, as needed. This was done by showing that continuous delay-differential equations models can be approximated by Boolean networks. The method presented here is quite general and could be applied to other types of biological networks. The examples show that simple Boolean models are able to capture steady states and complicated dynamics like hysteresis.

Boolean network models have the drawback that they do not give rise to exact concentration rates of the steady states because their discretization is too coarse grained. Discrete models with more than just two states, so called multi state models, might be suitable to give enough quantitative information about concentration rates while they are intuitive enough and easy to use for a wide range of scientists.

Acknowledgments

FH would like to thank Terry Herdman for his invaluable support and encouragement without which this research could not have been possible.

6.6 Appendix

6.6.1 *Lac* operon

Dilution and degradation

Since the degradation rates γ_M, γ_P, and γ_A are close to 0.5min^{-1}, we assume that mRNA, permease, and allolactose are degraded after 2 times steps. γ_L and γ_B are very small and will be neglected in our model.

β-galactosidase

For the β-galactosidase enzyme the equation in the continuous model is

$$\begin{aligned}\frac{dB}{dt} &= \alpha_B e^{-\mu\tau_B} M_{\tau_B} - \tilde{\gamma}_B B \\ &= \alpha_B e^{-\mu\tau_B} M_{\tau_B} - (\gamma_B + \mu)B.\end{aligned}$$

Messenger RNA is translated into β-galactosidase which takes time τ_B, so f_B depends on M_τ, the mRNA concentration time τ_B ago.

Since the degradation rate γ_B is low, the Boolean model neglects the decrease due to dilution and degradation and we model β-galactosidase with the single equation

$$f_B = x_{M_\tau} \vee x_B.$$

Allolactose A

$$\frac{dA}{dt} = \alpha_A B \frac{L}{K_L + L} - \beta_A B \frac{A}{K_A + A} - \tilde{\gamma}_A A$$

Allolactose is gained by conversion of lactose and reduced by the loss via conversion to glucose and galactose, both mediated by β-galactosidase. Like for mRNA, we use an extra variable A_{old} to capture the loss of allolactose due to dilution and degradation.

$$\begin{aligned}f_A &= (x_B \wedge x_L) \vee (x_A \wedge \neg x_{A_{\text{old}}}) \\ f_{A_{\text{old}}} &= \neg(x_B \wedge x_L) \\ &= \neg x_B \vee \neg x_L\end{aligned} \tag{6.3}$$

Notice that (6.3) does not depend on any delayed variables.

Internal lactose L

For internal lactose, Yildirim's model suggests the following equation

$$\begin{aligned}\frac{dL}{dt} =&\alpha_L P\frac{L_e}{K_{L_e}+L_e} - \beta_L P\frac{L}{K_{L_1}+L}\\ &- \beta_{L_2} B\frac{L}{K_{L_2}+L} - \tilde{\gamma}_L L.\end{aligned}$$

The degradation term γ_L is low enough to be ignored in the discrete model. Lactose is broken down into glucose, galactose and allolactose by β-galactosidase, and permease actively transports lactose in and out of the cell. Again, since permease and β-galactosidase reduce the internal lactose concentration, we introduce the extra variable L_{old} to turn L off, if it has not been produced and if permease or β-galactosidase are present to reduce it.

$$\begin{aligned}f_L &= (x_P \wedge x_{L_e}) \vee [x_L \wedge \neg(x_{L_{\text{old}}} \wedge (x_P \vee x_B))]\\ f_{L_{\text{old}}} &= \neg(x_P \wedge x_{L_e})\\ &= \neg x_P \vee \neg x_{L_e}\end{aligned}$$

Permease P

$$\frac{dP}{dt} = \alpha_P e^{-\mu(\tau_P+\tau_B)} M_{\tau_P+\tau_B} - \tilde{\gamma}_p P$$

Messenger RNA is translated into permease, so the permease concentration P is directly proportional to the mRNA concentration M at time $(\tau_P + \tau_B)$ ago and dilution and degradation reduce permease concentration, which is why P_{old} is used in the Boolean model.

$$\begin{aligned}f_P &= x_{M_\tau} \vee (x_P \wedge \neg x_{P_{\text{old}}})\\ f_{P_{\text{old}}} &= \neg x_{M_\tau}.\end{aligned}$$

6.6.2 Lambda phage

Re-numerating the equations results in

$$
\begin{aligned}
f_1 &= x_8 \vee \overline{x_{10}} \vee (x_1 \wedge \overline{x_2}) \\
f_2 &= \overline{x_8} \wedge x_{10} \\
f_3 &= x_1 \\
f_4 &= \overline{x_8} \vee (x_4 \wedge \overline{x_5}) \\
f_5 &= x_8 \\
f_6 &= x_4 \\
f_7 &= x_6 \\
f_8 &= \overline{x_{13}} \vee (\overline{x_{14}} \wedge (x_3 \vee (x_8 \wedge (\overline{x_9} \vee \overline{x_{12}})))) \\
f_9 &= \overline{x_3} \vee x_{14} \\
f_{10} &= x_7 \vee (x_{10} \wedge \overline{x_{11}}) \\
f_{11} &= \overline{x_7} \\
f_{12} &= x_{12} \vee x_{14} \\
f_{13} &= x_{12} \\
f_{14} &= x_{14}.
\end{aligned}
$$

The Boolean model has the dependency graph shown in Fig. 6.7.

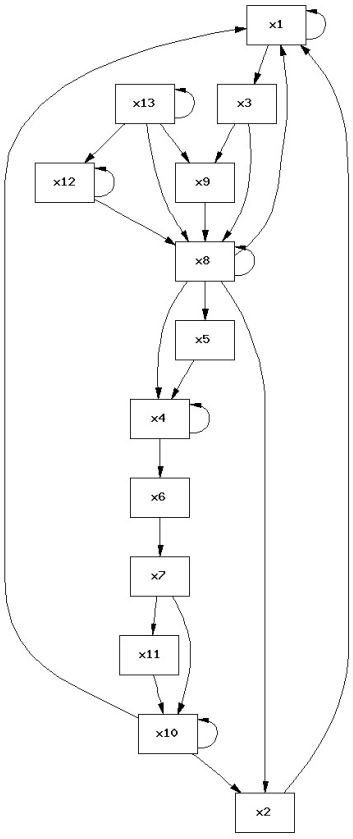


Figure 6.7: Dependency graph of the Boolean model for Lambda Phage

Chapter 7

Future directions

Adding discrete modeling to the standard toolset of biologists requires computationally powerful software that is simple to learn and easy to use. We envision the mathematical theory developed in this dissertation to be incorporated into software tools that do not require an understanding of the underlying mathematical methods. Rather, the software interface must be intuitive to biologists, not modelers or mathematicians. The usefulness of such software would be immensely increased by incorporating efficient algorithms for the following, which constitute interesting mathematical problems by themselves.

- Optimal control: deterministic and heuristic optimal control algorithms that benefit from the algebraic structure of polynomial dynamical systems;
- parameter estimation;
- stochastic models;
- automated conversion of agent based models into PDS;
- analysis of siphons and traps in Petri nets;
- bifurcation theory.

We hope that such software would introduce computational thinking on a broad scale to biologists, and ultimately lead to a better understanding of all living organisms.

Bibliography

[1] DNA info. Available at http://dnainfo.wikispaces.com.

[2] NBK institute of mining engineering. Available at http://www.mining.ubc.ca.

[3] W. Adams and P. Loustaunau. *An introduction to Gröbner bases*, volume 3 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 1994.

[4] R. Albert and H. G. Othmer. The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in *Drosophila melanogaster*. *Journal of Theoretical Biology*, 223:1–18, 2003.

[5] G. An. In silico experiments of existing and hypothetical cytokine-directed clinical trials using agent-based modeling. *Crit Care Med*, 32(10):2050–2060, Oct. 2004.

[6] E. Arnold, S. Lucas, and L. Taalman. Gröbner basis representations of sudoku. *College Mathematics Journal*, 41(2):101–111, 2010.

[7] E. Aurell and K. Sneppen. Epigenetics as a first exit problem. *Phys. Rev. Lett.*, 88(4):048101, Jan 2002.

[8] A. Bernasconi, B. Codenotti, V. Crespi, and G. Resta. Computing gröbner bases in the boolean setting with applications to counting, 1997.

[9] D. Bonchev, S. Thomas, A. Apte, and L. B. Kier. Cellular automata modelling of biomolecular networks dynamics. *SAR and QSAR in Environmental Research*, 21(1):77–102, 2010.

[10] M. Brickenstein and A. Dreyer. Polybori: A framework for gröbner-basis computations with boolean polynomials. *Journal of Symbolic Computation*, 44(9):1326 – 1345, 2009. Effective Methods in Algebraic Geometry.

[11] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, Inst. University of Innsbruck, 1965.

[12] L. Calzone, L. Tournier, S. Fourquet, D. Thieffry, B. Zhivotovsky, E. Barillot, and A. Zinovyev. Mathematical modelling of cell-fate decision in response to death receptor engagement. *PLoS Comput Biol*, 6(3):e1000702, 03 2010.

[13] F. Castiglione, K. Duca, A. S. Jarrah, R. Laubenbacher, D. Hochberg, and D. Thorley-Lawson. Simulating epstein-barr virus infection with c-immsim. *Bioinformatics*, 23:1371–1377, 2007.

[14] M. Cohn and K. Horibata. Inhibition by glucose of the induced synthesis of the β-galactoside-enzyme system of escherichia coli. analysis of maintenance. *J Bacteriol.*, 78:601–612, 1959.

[15] C. Conradi, J. Stelling, and J. Raisch. Structure discrimination of continuous models for biochemical reaction networks via finite state machines. In *Proc. IEEE Int. Symposium on Intelligent Control*, pages 138–143, Mexico City, Mexico, 2001.

[16] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer, 2 edition, 1997.

[17] R. J. de Boer. *Theoretical Biology*. Utrecht University, 2011.

[18] E. Dimitrova, L. D. Garcia-Puente, F. Hinkelmann, A. S. Jarrah, R. Laubenbacher, B. Stigler, M. Stillman, and P. Vera-Licona. Polynome. Available at http://polymath.vbi.vt.edu/polynome, 2010.

[19] E. Dimitrova, L. D. Garcìa-Puente, F. Hinkelmann, A. S. Jarrah, R. Laubenbacher, B. Stigler, M. Stillman, and P. Vera-Licona. Parameter estimation for boolean models of biological networks. *Theoretical Computer Science*, 412(26):2816 – 2826, 2011.

[20] E. Dimitrova, A. Jarrah, R. Laubenbacher, and B. Stigler. A Gröbner fan method for biochemical network modeling. In *ISSAC 2007*, pages 122–126. ACM, New York, 2007.

[21] E. S. Dimitrova, M. P. V. Licona, J. McGee, and R. Laubenbacher. Discretization of time series data. *Journal of Computational Biology*, 17(6):853–868, 2010.

[22] E. Dubrova and M. Teslenko. A sat-based algorithm for finding attractors in synchronous boolean networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8:1393–1399, 2011.

[23] R. Edwards, H. T. Siegelmann, K. Aziza, and L. Glass. Symbolic dynamics and computation in model gene networks. *Chaos*, 11:160–169, Mar. 2001.

[24] B. Elspas. The theory of autonomous linear sequential networks. In *Linear Sequential Switching Circuits*, pages 21–61. Holden-Day, San Francisco, Calif., 1965.

[25] H. Enderling, M. Chaplain, and P. Hahnfeldt. Quantitative modeling of tumor dynamics and radiotherapy. *Acta Biotheoretica*, 58:341–353, 2010.

[26] S. Eubank, H. Guclu, V. S. A. Kumar, M. V. Marathe, Z. T. Aravind Srinivasan, and N. Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429(6988):180–184, 2004.

[27] A. Faure, A. Naldi, C. Chaouiya, and D. Thieffry. Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle. *Bioinformatics*, 22(14):e124–131, July 2006.

[28] N. Friedman, M. Linial, and I. Nachman. Using bayesian networks to analyze expression data. *Journal of Computational Biology*, 7:601–620, 2000.

[29] J. Gago-Vargas, M. I. Hartillo-Hermoso, J. Martín-Morales, and J. M. Ucha-Enríquez. Sudokus and gröbner bases: Not only a *ivertimento*. In V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors, *CASC*, volume 4194 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 2006.

[30] M. Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123, October 1970.

[31] I. Gat-Viks and R. Shamir. Chain functions and scoring functions in genetic networks. *Bioinformatics*, 19:108–117, 2003.

[32] A. Giovini, T. Mora, G. Niesi, L. Robbiano, and C. Traverso. "One sugar cube, please" or selection strategies in the buchberger algorithm. In *ISSAC'91*, pages 49–54, 1991.

[33] D. R. Grayson and M. E. Stillman. Macaulay2, a software system for research in algebraic geometry. Available at http://www.math.uiuc.edu/Macaulay2/, 1992.

[34] G.-M. Greuel, G. Pfister, and H. Schönemann. Singular computer algebra system. Available at http://www.singular.uni-kl.de/.

[35] V. Grimm, U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S. K. Heinz, G. Huse, A. Huth, J. U. Jepsen, C. Jørgensen, W. M. Mooij, B. Müller, G. Pe'er, C. Piou, S. F. Railsback, A. M. Robbins, M. M. Robbins, E. Rossmanith, N. Rüger, E. Strand, S. Souissi, R. A. Stillman, R. Vabø, U. Visser, and D. L. DeAngelis. A standard protocol for describing individual-based and agent-based models. *Ecological Modelling*, 198(1-2):115 – 126, 2006.

[36] V. Grimm and S. F. Railsback. *A Course in Individual-based and Agent-based Modeling. Unpublished manuscript*. Princeton Univ. Press, 2010.

[37] S. Hardy and P. Robillard. Modeling and simulation of molecular biology systems using petri nets: modeling goals of various approaches. *J Bioinform. Comput. Biol.*, 2(4):595–613, 2004.

[38] S. Harris, B. Sawhill, A. Wuensche, and S. Kauffman. A model of transcriptional regulatory networks based on biases in the observed regulation rules. *Complex.*, 7(4):23–40, 2002.

[39] L. Heiser, N. Wang, C. Talcott, K. Laderoute, M. Knapp, Y. Guan, Z. Hu, S. Ziyad, B. Weber, S. Laquerre, J. Jackson, R. Wooster, L. Kuo, J. Gray, and P. Spellman. Integrated analysis of breast cancer cell lines reveals unique signaling pathways. *Genome Biology*, 10:1–17, 2009. 10.1186/gb-2009-10-3-r31.

[40] F. Hinkelmann. Model repository of discrete biological systems. Available at http://dvd.vbi.vt.edu/repository.html.

[41] F. Hinkelmann. Interpolation parser for truth tables. Available at http://admg.vbi.vt.edu/software/interpolation-tool-for-truth-tables, 2010.

[42] F. Hinkelmann. Singular implementation of NCF inferring algorithm. Available at http://www.math.vt.edu/people/fhinkel/ncf.lib, 2010.

[43] F. Hinkelmann and E. Arnold. Fast Gröbner basis computation for Boolean polynomials. *Journal of Software for Algebra and Geometry: Macaulay2, under revision*, 2011.

[44] F. Hinkelmann, M. Brandon, B. Guang, R. McNeill, G. Blekherman, A. Veliz-Cuba, and R. Laubenbacher. ADAM: Analysis of discrete models of biological systems using computer algebra. *BMC Bioinformatics*, 12(1):295, 2011.

[45] F. Hinkelmann, M. Brandon, B. Guang, R. McNeill, A. Veliz-Cuba, G. Blekherman, and R. Laubenbacher. ADAM: Analysis of analysis of dynamic algebraic models. Available at http://adam.vbi.vt.edu/, 2010.

[46] F. Hinkelmann and A. S. Jarrah. Inferring biologically relevant models: Nested canalyzing functions. *under review*, 2010.

[47] F. Hinkelmann and R. Laubenbacher. Boolean models of bistable biological systems. *Discrete and Continuous Dynamical Systems. Series S*, 4(6):1443–1456, 2011.

[48] F. Hinkelmann, D. Murrugarra, A. Jarrah, and R. Laubenbacher. A mathematical framework for agent based models of complex biological networks. *Bulletin of Mathematical Biology*, 73:1583–1602, 2011. 10.1007/s11538-010-9582-8.

[49] F. Jacob and J. Monod. Genetic regulatory mechanisms in the synthesis of proteins. *J Mol Biol*, 3:318–356, Jun 1961.

[50] A. Jarrah and R. Laubenbacher. Discrete models of biochemical networks: The toric variety of nested canalyzing functions. In H. Anai, K. Horimoto, and T. Kutsia, editors, *Algebraic Biology*, number 4545 in LNCS, pages 15–22. Springer, 2007.

[51] A. Jarrah, R. Laubenbacher, and A. Veliz-Cuba. The dynamics of conjunctive and disjunctive boolean network models. *Bulletin of Mathematical Biology*, 72:1425–1447, 2010.

[52] A. Jarrah, B. Raposa, and R. Laubenbacher. Nested canalyzing, unate cascade, and polynomial functions. *Physica D*, 233:167–174, 2007.

[53] A. S. Jarrah, R. Laubenbacher, B. Stigler, and M. Stillman. Reverse-engineering of polynomial dynamical systems. *Advances in Applied Mathematics*, 39(4):477 – 489, 2007.

[54] S. Kauffman, C. Peterson, B. Samuelsson, and C. Troein. Random boolean network models and the yeast transcriptional network. *PNAS*, 100(25):14796–14799, 2003.

[55] S. Kauffman, C. Peterson, B. Samuelsson, and C. Troein. Genetic networks with canalyzing Boolean rules are always stable. *PNAS*, 101(49):17102–17107, 2004.

[56] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, March 1969.

[57] S. Klamt, J. Saez-Rodriguez, J. A. Lindquist, L. Simeoni, and E. D. Gilles. A methodology for the structural and functional analysis of signaling and regulatory networks. *BMC Bioinformatics*, 7(56), 2006.

[58] R. Laubenbacher and A. Jarrah. DVD - Discrete Visualizer of Dynamics. Available at http://dvd.vbi.vt.edu/.

[59] R. Laubenbacher, A. Jarrah, H. Mortveit, and S. Ravi. *Encyclopedia of Complexity and System Science*, chapter The mathematics of agent-based modeling formalisms. Springer Verlag, New York, 2009.

[60] R. Laubenbacher and B. Stigler. A computational algebra approach to the reverse engineering of gene regulatory networks. *Journal of Theoretical Biology*, 229:523–537, 2004.

[61] F. Li, T. Long, Y. Lu, Q. Ouyang, and C. Tang. The yeast cell-cycle network is robustly designed. *PNAS*, 101:4781, 2004.

[62] R. Lidl and H. Niederreiter. *Finite Fields*. Cambridge University Press, New York, 1997.

[63] Y. Mansury, M. Kimura, J. Lobo, and T. S. Deisboeck. Emerging patterns in tumor systems: Simulating the dynamics of multicellular clusters with an agent-based spatial agglomeration model. *Journal of Theoretical Biology*, 219(3):343 – 370, 2002.

[64] L. Mendoza and I. Xenarios. A method for the generation of standardized qualitative dynamical systems of regulatory networks. *Theoretical Biology and Medical Modelling*, 3(1):13, March 2006.

[65] G. Mullen and D. Panario, editors. *Handbook on Finite Fields*. CRC Press, 2011.

[66] D. Murrugarra and R. Laubenbacher. Multi-states nested canlyzing functions. *preprint*, 2011.

[67] C. Müssel, M. Hopfensitz, and H. A. Kestler. BoolNet—an R package for generation, reconstruction and analysis of Boolean networks. *Bioinformatics*, 26(10):1378–1380, 05 2010.

[68] A. Naldi, D. Berenguier, A. Fauré, F. Lopez, D. Thieffry, and C. Chaouiya. Logical modelling of regulatory networks with GINsim 2.3. *Biosystems*, 97(2):134–139, 2009.

[69] A. Naldi, D. Thieffry, and C. Chaouiya. GINsim - model repository. Available at http://gin.univ-mrs.fr/GINsim/model_repository.html.

[70] A. Naldi, D. Thieffry, and C. Chaouiya. Decision diagrams for the representation and analysis of logical models of genetic networks. In *CMSB'07: Proceedings of the 2007 international conference on Computational methods in systems biology*, pages 233–247, Berlin, Heidelberg, 2007. Springer-Verlag.

[71] S. Nikolajewaa, M. Friedela, and T. Wilhelm. Boolean networks with biologically relevant rules show ordered behavior. *Biosystems*, 90(1):40–47, 2007.

[72] A. Novick and M. Weiner. Enzyme induction as an all-or-none phenomenon. *Proc Natl Acad Sci U S A*, 43(7):553–66, Jul 1957.

[73] M. Pedicini, F. Barrenäs, T. Clancy, F. Castiglione, E. Hovig, K. Kanduri, D. Santoni, and M. Benson. Combining network modeling and gene expression microarray analysis to explore the dynamics of Th1 and th2 cell regulation. *PLoS computational biology*, 6(12), 2010.

[74] G. Pe'er, D. Saltz, and K. Frank. Virtual corridors for conservation management. *Conservation Biology*, 19(6):1997–2003, December 2005.

[75] M. Pogson, R. Smallwood, E. Qwarnstrom, and M. Holcombe. Formal agent-based modelling of intracellular chemical interactions. *Biosystems*, 85(1):37–45, 2006.

[76] R. Porreca, E. Cinquemani, J. Lygeros, and G. Ferrari-Trecate. Identification of genetic network dynamics with unate structure. *Bioinformatics*, 26(9):1239–1245, 2010.

[77] M. Ptashne. *A Genetic Switch Phage Lambda Revisited*. Cold Spring Harbor Laboritory Press, 3rd edition, 2004.

[78] L. Raeymaekers. Dynamics of boolean networks controlled by biologically meaningful functions. *Journal of Theoretical Biology*, 218(3):331 – 341, 2002.

[79] E. Remy, P. Ruet, L. Mendoza, D. Thieffry, and C. Chaouiya. From logical regulatory graphs to standard petri nets: Dynamical roles and functionality of feedback circuits. In C. Priami, A. Ingólfsdóttir, B. Mishra, and H. Riis Nielson, editors, *Transactions on Computational Systems Biology VII*, volume 4230 of *Lecture Notes in Computer Science*, pages 56–72. Springer Berlin / Heidelberg, 2006.

[80] Robert D. Leclerc. Survival of the sparsest: robust gene networks are parsimonious. *Mol Syst Biol*, 4, 2008.

[81] C. Rohr, W. Marwan, and M. Heiner. Snoopy—a unifying petri net framework to investigate biomolecular networks. *Bioinformatics*, 26(7):974–975, 04 2010.

[82] D. Ruths, M. Muller, J.-T. T. Tseng, L. Nakhleh, and P. T. Ram. The signaling petri net-based simulator: a non-parametric strategy for characterizing the dynamics of cell-specific signaling networks. *PLoS computational biology*, 4(2):e1000005+, Feb. 2008.

[83] A. Sackmann, M. Heiner, and I. Koch. Application of petri net based analysis techniques to signal transduction pathways. *BMC Bioinformatics*, 7(1):482, 2006.

[84] J. Saez-Rodriguez, L. G. Alexopoulos, J. Epperlein, R. Samaga, D. A. Lauffenburger, S. Klamt, and P. K. Sorger. Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. *Molecular Systems Biology*, 5, Dec. 2009.

[85] O. Sahin, H. Frohlich, C. Lobke, U. Korf, S. Burmester, M. Majety, J. Mattern, I. Schupp, C. Chaouiya, D. Thieffry, A. Poustka, S. Wiemann, T. Beissbarth, and D. Arlt. Modeling erbb receptor-regulated g1/s transition to find novel targets for de novo trastuzumab resistance. *BMC Systems Biology*, 3(1):1, 2009.

[86] M. Santillán and M. C. Mackey. Why the lysogenic state of phage lambda is so stable: a mathematical modeling approach. *Biophys J*, 86:75–84, Jan 2004.

[87] Y. Sato, A. Nagai, and S. Inoue. Computer mathematics. In D. Kapur, editor, *Lecture Notes In Artificial Intelligence*, chapter On the Computation of Elimination Ideals of Boolean Polynomial Rings, pages 334–348. Springer-Verlag, Berlin, Heidelberg, 2008.

[88] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang. Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, February 2002.

[89] I. Shmulevich and H. Lähdesmäki. BN/PBN MATLAB toolbox. Available at http://personal.systemsbiology.net/ilya/PBN/PBN.htm, 2005.

[90] I. Shmulevich, H. Lähdesmäki, E. R. Dougherty, J. Astola, and W. Zhang. The role of certain Post classes in Boolean network models of genetic networks. *PNAS*, 100(19):10734–10739, 2003.

[91] C. Sima, J. Hua, and S. Jung. Inference of gene regulatory networks using time-series data: A survey. *Current Genomics*, 10(14):416–429, 2009.

[92] L. J. Steggles, R. Banks, O. Shaw, and A. Wipat. Qualitatively modelling and analysing genetic regulatory networks: a Petri net approach. *Bioinformatics*, 23:336–343, 2007.

[93] B. Stigler, A. Jarrah, M. Stillman, and R. Laubenbacher. Reverse-engineering of dynamic networks. *Annals of the New York Academy of Sciences*, 1115:168–177, 2007.

[94] R. Thomas. Boolean formalisation of genetic control circuits. *Journal of Theoretical Biology*, 42:565–583, 1973.

[95] R. Thomas and R. D'Ari. *Biological Feedback*. CRC Press, 1998.

[96] T. Tian and K. Burrage. Stochastic models for regulatory networks of the genetic toggle switch. *Proc Natl Acad Sci U S A*, 103(22):8372–8377, May 2006.

[97] R. A. H. Toledo. Linear finite dynamical systems. *Communications in Algebra*, 33:2977–2989, 2005.

[98] A. Veliz-Cuba, A. S. Jarrah, and R. Laubenbacher. Polynomial algebra of discrete models in systems biology. *Bioinformatics*, 26(13):1637–1643, July 2010.

[99] C. H. Waddington. Canalisation of development and the inheritance of acquired characters. *Nature*, 150:563–564, 1942.

[100] Z. Wang, C. M. Birch, J. Sagotsky, and T. S. Deisboeck. Cross-scale, cross-pathway evaluation using an agent-based non-small cell lung cancer model. *Bioinformatics*, 25(18):2389–2396, 2009 Sep 15.

[101] P. Wong, S. Gladney, and J. D. Keasling. Mathematical model of the lac operon: inducer exclusion, catabolite repression, and diauxic growth on glucose and lactose. *Biotechnology progress*, 13(2):132–143, 1997.

[102] M. Wu, X. Yang, and C. Chan. A Dynamic Analysis of IRS-PKR Signaling in Liver Cells: A Discrete Modeling Approach. *PLoS ONE*, 4(12):e8040, 12 2009.

[103] A. Wuensche. DDLab - Discrete Dynamics Lab. Available at http://www.ddlab.com/, 2010.

[104] N. Yildirim and M. C. Mackey. Feedback regulation in the lactose operon: a mathematical modeling study and comparison with experimental data. *Biophys J*, 84(5):2841–2851, May 2003.

Index