

FREIE UNIVERSITÄT BERLIN
FACHBEREICH MATHEMATIK UND INFORMATIK

DISSERTATION

Contributions to the Analysis of Qualitative Models of Regulatory Networks

Hannes KLARNER

November 2014

1. Gutachter:

Prof. Dr. Alexander BOCKMAYR

2. Gutachter:

Prof. Dr. Denis THIEFFRY

3. Gutachter:

Prof. Dr. Heike SIEBERT

Tag der Disputation:

10. Dezember 2014

Contents

1	Introduction	9
1.1	Positioning and Motivation	9
1.2	Summary	14
2	Qualitative Models of Regulatory Networks	17
2.1	The Qualitative Modeling Framework	17
2.1.1	The Components	17
2.1.2	The Update Functions	22
2.2	The Interaction Graph	23
2.2.1	Condensation and SCCs	24
2.3	The State Transition Graph	26
2.3.1	Transition Rules	26
2.3.2	Trajectories	28
2.4	Temporal Logic	29
2.4.1	Linear Time Logic	30
2.4.2	Computation Tree Logic	32
2.4.3	NuSMV Model Checking	33
2.5	Case Study: The Galactose Switch in Yeast	38
3	Reachability Queries	43
3.1	Introduction: Time Series Data	43
3.2	Compatibility	45
3.2.1	Temporal Logic Encoding	47
3.3	Monotonic Compatibility	48
3.3.1	Temporal Logic Encoding	49
3.3.2	Computational Complexity	53
3.4	Assessment of the Sampling Rate	55
3.5	Variations of Compatibility	57
3.5.1	Robust Compatibility	57
3.5.2	Attracting Pathways	59
3.5.3	Stability and Partial Stability	60
3.6	Software	60
3.7	Case Study: The Galactose Switch in Yeast	61
3.7.1	Simulations	62
3.7.2	Data Discretization	64
3.7.3	Compatibility	67
3.7.4	Assessment of Sampling Rate	69
3.8	Discussion	71

4	Asymptotics of a Model	77
4.1	Attractors	77
4.2	Attractor Queries	78
4.3	Seeds and Trap Sets	81
4.4	Seeds and Cyclic Attractors	84
4.5	The Prime Implicant Graph	87
4.6	Computing Seeds	91
4.6.1	ILP Encoding	92
4.6.2	ASP Encoding	93
4.7	Seeds and Positive Feedback	94
4.8	Completeness and Univocality	97
4.8.1	Model Reduction 1: Substitution	99
4.8.2	Model Reduction 2: Input Percolations	99
4.9	Case Study: A MAPK Signaling Network	100
4.10	Discussion	106
5	Construction and Analysis of a Model Pool	109
5.1	Introduction: Reverse Engineering	109
5.2	The Specification Language	110
5.3	Description of a Prototype Software	114
6	Conclusion	121
	Bibliography	132
	Declaration	133

Extended Abstract

This thesis addresses three challenges in modeling regulatory and signal transduction networks. Starting point is the generalized logical formalism as introduced by R. Thomas and further developed by D. Thieffry, E. H. Snoussi and M. Kaufman. We introduce the fundamental concepts that make up such models, the interaction graph and the state transition graph, as well as model checking, a computer science technique for deciding whether a finite transition system satisfies a given temporal specification.

The first problem we turn to is that of whether a given model is consistent with time series data. To do so we introduce query patterns that can be automatically derived from discretized data. Time series data, being such an abundant source of information for reverse engineering, has previously been used in the context of logical models but only under the synchronous, transition-based notion of consistency. The arguably more realistic asynchronous transition relation has so far been excluded from such data driven reverse engineering, probably because the corresponding non-determinism in the transition system introduces additional obstacles to the already hard problem. Our contribution here is a path-based notion of consistency between model and data that works for any transition relation. In particular, we discuss linear time properties like monotony and branching time properties like robustness. The result are several query patterns, similar to but more complex than the ones proposed by P. T. Monteiro et al. A toolbox, called `TEMPORALLOGICTIMESERIES` for the automated construction of queries from data is also presented.

The second problem we turn to concerns the two types of long-term behaviors that logical models are capable of producing: steady states, in which the activity levels of all network components are kept at a fixed value, and cyclic attractors in which some components are unsteady and produce sustained oscillations. We attempt to understand the emergence of these behaviors by searching for symbolic steady states as defined by H. Siebert. Our main contribution is the introduction of the prime implicant graph, which describes all minimal conditions under which components may change their activities, and an optimization-based algorithm for the enumeration of all maximal and minimal symbolic steady states. Essentially, we generalize the canalizing effects and forcing structure that were first introduced and studied by S. Kauffman and F. Fogelman in the context of random Boolean networks. The chapter includes a theorem that relates symbolic steady states to the existence of positive feedback circuits in the interaction graph. A toolbox, called `BOOLNETFIXPOINTS` that implements our algorithm is also described.

The theme of the last chapter is how to deal with uncertainties that inevitably appear during the modeling of biological systems. One is often forced to resolve them since most types of analysis require a single, fully specified model. The knowledge gap is usually filled by making simplifications or by introducing additional assumptions that are hard to justify and therefore somewhat arbitrary. The alternative is to work with and analyze sets of alternative models, rather than single models. This idea entails additional theoretical and practical challenges: With which language should we describe our partial knowledge about a system? How can predictions be made given that each model in the set may behave differently? How can hypotheses and additional data be added to the current knowledge in a systematic manner?

It seems that there are in principle two different approaches. The first one is constraint-based and studied by F. Corblin et al. It translates the partial knowledge and modeling formalism into facts and rules of a logic program. Common solvers can then deduce additional properties or test the validity of given queries across all models. In contrast, we propose to study the pros and cons of an explicit approach that enumerates all models that agree with a given partial specification. During the first step, models are enumerated and stored in a database. During a second step, models are annotated with additional information that is obtained from custom algorithms. The relationships between the annotations are then analyzed in a third step. The chapter is based on the prototype implementation `LOGICMODELCLASSIFIER` that performs the discussed steps.

Throughout, we apply our results to two previously published models of biological systems. The first one is a small model of the galactose switch which regulates the transcription of genes that are involved in the metabolism of yeast. We address questions that arise during the construction of the model, for example the number of involved components and their interactions, as well as issues related to model validation and model revision with time series data. The case study also discusses different approaches to data discretization.

The second one is a medium size model of the MAPK network studied by D. Thieffry et al. that is used to predict the cell fate response to different stimuli involving the growth factors *EGF*, *TGFB*, *FGF* and DNA damage. With the methods developed in this thesis we can prove that the model is capable of 18 different asymptotic behaviors, 12 of them steady states and 6 cyclic attractors. The question of which attractor is reached from which initial state is answered and we can show that the response in terms of proliferation or growth arrest and apoptosis is fully determined by the input stimulus.

Ausführliche Kurzfassung

Diese Arbeit beschäftigt sich mit drei Herausforderungen, die beim Modellieren von regulatorischen Netzwerken und der Signaltransduktion auftreten. Zunächst beschreiben wir den logischen Formalismus, der von R. Thomas eingeführt und D. Thieffry, E. H. Snoussi und M. Kaufman weiterentwickelt wurde. Er zeichnet sich dadurch aus, dass die Komponenten des Modells nur Werte aus einem endlichen Bereich annehmen. Wir stellen die grundlegenden Objekte eines logischen Modells, den Zustandsübergangsgraphen und den Interaktionsgraphen, vor und besprechen das Model Checking, eine Methode zur automatischen Prüfung von Ausdrücken temporaler Logiken in gegebenen Modellen.

Der erste Teil der Arbeit beschäftigt sich damit, wie wir entscheiden können, ob ein gegebenes Modell mit Zeitreihendaten konsistent ist. Dazu konstruieren wir verschiedene Anfragen nach denen Daten in temporale Logiken übersetzt werden können. Zeitreihendaten spielen eine wichtige Rolle beim Reverse Engineering von logischen Modellen nach Daten, aber bisher nur unter der Annahme, dass die Übergänge des dem Modell zugrundeliegenden Übergangssystems synchron sind. Die realistischere Annahme, nämlich dass sich die Aktivitäten der Komponenten asynchron ändern, wurde bisher in diesem Zusammenhang nicht untersucht. Das liegt wahrscheinlich daran, dass die dadurch entstehenden nicht-deterministischen Übergangssysteme ein ohnehin schon schwieriges Problem noch weiter verkomplizieren. Unser Beitrag in diesem Zusammenhang sind verschiedene pfadbasierte Definitionen von Konsistenz, die unabhängig von der gewählten Übergangsrelation prüfbar sind. Wir diskutieren die Möglichkeit Monotonie- und Robustheits-Annahmen mithilfe von Linear Time Logic und Computational Tree Logic zu kodieren. Außerdem wird die Toolbox `TEMPORALLOGICTIMESERIES` zur automatischen Generierung der besprochenen Anfragen vorgestellt.

Im zweiten Teil wenden wir uns dem Langzeitverhalten und den Attraktoren von logischen Modellen zu. Wir versuchen die Existenz von stabilen Zuständen, in denen die Aktivitäten aller Komponenten konstant bleiben, und auch von zyklischen Attraktoren, in denen einige Komponenten dauerhaft instabil sind, mithilfe der sogenannten symbolischen Fixpunkte zu erklären. Die Ergebnisse beziehen sich dabei auf die Definitionen von H. Siebert. Es werden die Prim-Implikanten, als minimale Bedingungen unter denen diskrete Funktionen ihren Wert ändern können, eingeführt und der Prim-Implikanten-Graph vorgestellt. Das zentrale Ergebnis ist, dass symbolische Fixpunkte durch bestimmte Kantemengen in diesem Graphen repräsentiert werden. Diese können durch 0-1 Optimierungsprobleme beschrieben und mithilfe von üblichen Constraint-Solvern gefunden werden. Ein Skript, das alle beschriebenen Schritte durchführt, ist unter dem Namen `BOOLNETFIXPOINTS` verfügbar.

Im letzten Teil der Arbeit beschäftigen wir uns mit Ungewissheiten, die während des Modellierens biologischer Systeme zwangsläufig auftreten. Oft ist man gezwungen diese auszuräumen, da die meisten Analysemethoden vollständig spezifizierte Modelle benötigen. Das geschieht oft dadurch, dass starke Vereinfachungen gemacht oder schwer zu begründende, und damit willkürliche, Annahmen getroffen werden müssen. Die Alternative dazu besteht darin gleichzeitig mit allen Modellen zu arbeiten, die dem aktuellen Stand des Wissens entsprechen. Dadurch entstehen zusätzliche theoretische und praktische Herausforderungen: Mit welcher Sprache können Modelle teilweise spez-

ifiziert werden? Wie lassen sich Vorhersagen treffen, wenn sich jedes Modell potenziell anders Verhalten kann? Wie können zusätzliche Annahmen und Daten möglichst systematisch hinzugefügt werden?

Im Prinzip gibt es zwei Herangehensweisen. Der Constraint-Programming Ansatz, umgesetzt von F. Corblin et al., übersetzt das vorhandene, partielle Modell sowie den Modell-Formalismus in Fakten und Regeln eines logischen Programms. Übliche Logic Programming Solver können dann prüfen ob sich eine Eigenschaft aus diesem Programm herleiten läßt, oder nicht. Im Gegensatz dazu untersuchen wir die Vor- und Nachteile eines expliziten Ansatzes. Dabei werden alle Modelle, die mit einer gegebenen Spezifikation konsistent sind, aufgezählt und in einer Datenbank gespeichert. In einem zweiten Schritt können die Modelle mit zusätzlichen Informationen versehen werden, deren Beziehungen zueinander dann in einem dritten Schritt ausgewertet werden. Das Kapitel orientiert sich an der prototypischen Umsetzung LOGICMODELCLASSIFIER mit der die besprochenen Schritte ausgeführt werden können.

Die entwickelten Methoden und Ideen werden an zwei Modellen illustriert. Das erste ist ein kleines Modell des Galaktose-Genschalters in Hefe welcher am Stoffwechsel beteiligt ist. Es werden Fragen behandelt die sich beim Aufstellen des Modells stellen, zum Beispiel wieviele Komponenten gebraucht werden und wie diese interagieren sollen. Des Weiteren wird die Modell-Validierung und Revision mit Hilfe von Expressionsdaten angesprochen. Verschiedene Herangehensweisen zur Diskretisierung der Daten werden miteinander verglichen.

Das zweite ist ein größeres Modell des MAPK Systems, welches das Schicksal von Krebszellen in Abhängigkeit von verschiedenen Umwelteinflüssen beschreibt. Zu den Einflüssen zählen die Wachstumsfaktoren *EGF*, *TGFB* und *FGF* sowie DNS-Schäden. Mit den in der Dissertation erarbeiteten Methoden und Ideen können wir zeigen, dass das Model in der Lage ist 18 verschiedene Reaktionen zu zeigen. 12 davon sind stabile Zustände und 6 sind zyklische Attraktoren. Die Frage welcher Attraktor von welchem Anfangszustand erreicht werden kann wird beantwortet und wir können zeigen, dass das asymptotische Verhalten des Modells, in Bezug auf die Entscheidung Zellwachstum oder Zelltod, vollständig durch die Anfangsbedingungen bestimmt ist.

Chapter 1

Introduction

”We are interested in studying the dynamics of arbitrarily cross coupled biochemical networks in which the rates of synthesis of key metabolites are regulated by the concentrations of control molecules in the medium.”

–L. Glass and S. Kauffman in [1]

1.1 Positioning and Motivation

This thesis belongs to the field of life sciences that attempts to understand and simulate biological processes with mathematical models and computer programs. More specifically, the biological context is the study of protein interaction networks and in particular gene regulatory and signal transduction networks. Gene regulation is the process by which living cells control the transcription and translation of their genes and involves various mechanisms. There is, for example, the regulation by DNA binding proteins, the transcription factors, whose transcription is in turn regulated. The patterns of gene expression change over time and are characteristic for specific cell types as well as diseases. They can be monitored quantitatively by various techniques such as blotting mRNA concentrations or fluorescent microscopy in which the intensity of co-expressed fluorescent proteins is measured. Signal transduction is the mechanism by which cells sense stimuli in their environment via receptors in their membranes. A stimulus propagates via kinases along numerous pathways and results in a response that ranges from the production of specific enzymes that metabolize available nutrients to programmed cell death. Within a single cell there may be hundreds of genes, transcriptions factors, receptors and signaling proteins. To predict the dynamics of such networks therefore requires more than understanding how each part works.

Mathematics offers a large variety of frameworks and rules that define what a state of the system might be and how states may change over time. The concentrations of a set of proteins can, for example, be modeled by a system of differential equations. Each equation consists of synthesis and degradation terms that are constructed according to given rate laws. These are often highly non-linear and the equations may involve tens of variables and kinetic parameters. Even if knowledge regarding the reaction laws is available, ODE models still

require much quantitative information regarding the kinetic parameters. Such information is often difficult or impossible to measure. It is frequently remarked that the high accuracy of ODE trajectories may then be misleading, given that the parameters and rate laws behind them are only known to a much lesser degree.

To challenge these obstacles and potentially study larger networks, qualitative approaches have been suggested. One recurring technique is to introduce a mapping from the continuous space to a discrete space that, hopefully, preserves key properties like the number of steady states and their stabilities while being invariant to the precise values of the kinetic parameters. Examples are the work of L. Glass and S. Kauffman [1] and E. H. Snoussi [2].

Logical models were also suggested by M. Sugita [3] in a series of papers that introduced the idea that reaction networks may be understood as coupled molecular automata that behave much like the switching circuits of electrical engineering, each having binary inputs, outputs, and certain time delays. R. Thomas took a similar viewpoint and developed, together with D. Thieffry and M. Kaufman, a methodology that deals explicitly with time delays, asynchronous updates and more than two activity levels [4–6]. In [4] he also remarked that biological expertise is often expressed in qualitative, natural language statements like

"in the absence of immunity, this, in the presence of immunity, that;
at low temperature, this, at high temperature, that ..."

that is particularly predisposed to a logical formalization. Our work is based on this generalized logical formalism. We introduce it in Chap. 2.

A lot of effort goes into the construction and validation of individual models of specific biological systems and processes. Published models exist, for example, for the immunity control in bacteriophage lambda [6], the flower morphogenesis in *A. thaliana* [7], the cell cycle control in eukaryotes [8], the embryonic development of *D. melanogaster* [9,10] and processes involved in cell fate decisions [11].

Early on, the computer played an important role in studying logical networks. Originally it was required to simulate the properties of randomly generated Boolean networks. In [12] S. Kauffman used a computer to generate networks and record information about their attractors, sets of states that the system will eventually be trapped in. He suggested that the number of limit cycles and their lengths relates well to known characteristics regarding cell differentiation and robustness. Since then formal methods from computer science, notably logic programming and model checking, but also constraint programming have been introduced to the realm of logical networks. In particular, G. Bernot et al. [13] employed computation tree logic (CTL) for the specification and model checking for the verification of properties that can be used to validate logical models. Their ideas were implemented in the tool SMBIONET [14], which automates the selection of models that satisfy topological constraints as well as CTL specifications. At the same time F. Fages et al. introduced a rule-based modeling language and tool, called BIOCHAM [15], which is also capable of answering CTL queries.

Temporal logics, for which CTL is an example, have originally been introduced for software and hardware design of safety-critical systems for which merely testing some executions is not an option. Model checking provides the algorithms that can prove whether a design is reliable or not, see for example [16].

Transition systems, essentially directed graphs with nodes that represent states and arcs that indicate possible transitions, are used to model the designs. In addition, the states are labeled with information about the system at a certain moment of its behavior. For models of biological systems this may, for example, be whether a kinase is phosphorylated, whether a protein concentration has crossed a certain threshold or whether a gene is active. Besides the formal language, the power of model checking lies in it accepting a symbolic description of the transition system. Rather than requiring lists of states and transitions, symbolic model checking operates on transition rules. Depending on the specification, transition systems may therefore be checked for which graph search algorithms would fail.

The question of how to formulate appropriate queries and in which temporal logic is not straightforward, the semantic difference between queries in even the most common logics, linear time and branching time, can be very subtle, see for example [17]. P. T. Monteiro et al. [18] address this issue by suggesting four query patterns that capture recurring themes like *consequence* or *invariance*. Chap. 3 belongs to this domain. Here, we propose patterns that encode time series data as nested reachability queries. The motivation was to automatically check whether a model is capable of reproducing given sequential observations. We discuss several properties of paths, including monotony, path length, stability and robustness, that seem to us to be relevant to model validation. Naturally, one potential application of our encodings is to reverse engineering. But, we have also asked if a high confidence in the model can be used to assess a particular aspect of the quality of the data, the sampling rate.

A second family of questions that are usually asked when studying dynamical systems concerns the possible long-term behaviors. The logical networks that are considered in this thesis are capable of producing two types of long-term behaviors: sustained oscillations and steady states. Many algorithms for the detection of the corresponding attractors in the state space of a given system exist, see for example [19–29]. But, these algorithms can not be used interchangeably. Some work only for synchronous transition systems while others assume the asynchronous update. Some are specifically designed to detect steady states, others can detect sustained oscillations but only of a certain length. Furthermore, some are deterministic and exhaustive while others are stochastic and incomplete: they rely on sampling and partial state space exploration. Also, some algorithms depend on established problem-solving machinery and respective solvers, while others are stand-alone implementations. In addition, most algorithms include a reduction step during which the original problem is transformed into an equivalent smaller problem, for example by removing all so-called output cascades which can be shown to be irrelevant for the detection step. Since model reduction techniques potentially affect the efficiency of any analysis algorithm they have been addressed explicitly, for example by A. Naldi et al. [30] and A. Saadatpour et al. [31].

It turns out that finding steady states is more straightforward than finding sustained oscillations. For one, steady states are invariant of the assumed update strategy. Second, the number of states involved in sustained oscillations may grow exponentially with the size of the network. Hence, even if an algorithm can detect oscillations it may not be able to return the answer without finding a good, short description of the involved states, a problem that is itself, in general, NP-hard.

More importantly, the mere enumeration of all attractors of a logical network offers only very little insight into questions regarding decision making, robustness and control. That such insight is possible was first suggested by R. Thomas [32] who conjectured a connection between feedback circuits and the existence of certain long-term behaviors. More specifically, Thomas conjectured that positive and negative feedback circuits in the interaction graph are necessary conditions for the existence of multi-stationarity and sustained oscillations, respectively. Later, A. Richard et al. proved these conjectures to be true [33, 34]. Unfortunately, Thomas' theorems are rather weak in practice since all interesting models contain positive as well as negative feedback. Several results which aim at sharpening his theorems have since been obtained by the systematic study of elementary circuits by E. Remy and P. Ruet [35], of intersections of circuits by J. Demongeot et al. [36], and of the notion of circuit functionality by J. P. Comet et al. [37].

Starting point for our investigations regarding this topic is the work of S. Kauffman and F. Fogelman on canalizing effects and self-freezing circuits, see [38]. Kauffman observed that in random Boolean networks with an average connectivity of k between 2 and 3, a large number of components are asymptotically stable, leading to few and short limit cycles. He argued that this observation can be explained by the large proportion of canalizing functions among the functions of that connectivity. The key observation he made is that canalizing effects may form pathways along which activities propagate from one component to the next. If the network of all canalizing pathways, which he called the forcing structure, contains a cycle then the involved components can be shown to be fixed at the circuit values in some limit cycles. In [39] F. Fogelman developed these ideas further and proposed an algorithm for the computation of the forcing structure of a given network. Curiously, when the interest in logical models extended from random networks to specific networks, which are carefully assembled to model specific biological systems, the worth of the forcing graph was forgotten.

It was rediscovered by H. Siebert in [40] in the context of computing the so-called symbolic steady states, which generalize the notion of steady states. Rather than representing a single state, a symbolic state refers to a subspace of states, that is, a set of states with a particularly simple geometry. Under the appropriate definition of steadiness, symbolic states become useful in predicting the asymptotic behaviors of a network. They contain information about the location of attractors as well as their basins of attraction. Siebert showed that cycles in the forcing graph and symbolic steady states are related concepts: instances of the latter induce instances of the former, but she also remarked that not every symbolic steady state is induced by a corresponding circuit. The forcing structure in its original definition can therefore not detect all symbolic steady states. Our research regarding the long-term behaviors was hence motivated by the problem of how to compute all symbolic steady states of a given model.

The third topic that is addressed belongs to the domain of reverse engineering. Usually, the biological knowledge or data that is available at any given time can not be expected to define a model unambiguously. Even the simplest models may require an exponential number of experimental observations to be uniquely determined, see for example the problem of identifying a Boolean network by Knock-Out and Knock-In experiments, as described by T. Akutsu et

al. [41].

It seems that systematic investigations of the space of all models that are feasible with a given set of assumptions are often neglected in favor of ad hoc definitions or some form of model selection. Model selection is usually guided by a notion of optimality, for example by introducing a cost that is proportional to the number of components or the number of interactions between components and seeking the cheapest model that is consistent with all observations. In general, there may of course be more than one optimal model in which case one is again faced with the question of how to deal with sets of models, unless one chooses to randomly select one of them. The preference of single, fully specified models over sets of models has at least three aspects. First, most types of analysis require a single, fully specified model, see for example [42–45]. Second, working with sets of models entails additional theoretical and practical challenges: With what language should we describe our partial knowledge about a system? How can predictions be made given that each model in the set may behave differently? How can hypotheses and additional data be added to the current knowledge in a systematic manner? Third, many problems, for example the aforementioned time-series decision and attractor detection problems, are NP-hard. To be interested in all models that satisfy a specification based on these problems may then easily lead to an exponential number of NP-hard problems, because the search space may grow exponentially in the model size (e.g. number of network components). Our own experience in working with sets of models is that one may easily be interested in questions that are computationally infeasible.

To partly overcome the computational difficulties, frameworks that deal with model pools are therefore often restricted to specific reasoning engines. An example is the work by F. Corblin et al. [46–48]. Their approach is constraint-based and translates the partial knowledge and modeling formalism into facts and rules of a logic program. Common solvers, specifically the prolog engine SICSTUS [49] and the answer set solving collection POTASSCO [50], can then deduce additional properties or test the validity of given queries across all models.

An example of such constraint-based reasoning is the inference of a model of the regulatory network that controls the first steps of *D. Melanogaster* embryo segmentation as published in [48]. The prior knowledge consists of two parts, structural knowledge and expression profiles. The structural knowledge is represented by a generic interaction map that includes well-established as well as potential interactions. The expression profiles are translated into steady states assumptions that are loosely defined as successions of local peaks of expression of the gap genes. The answer set programming (ASP) based framework SYSBIOX then performs two tasks. It checks whether the prior knowledge is consistent and, second, it infers properties in the form of parameter inequalities, that are common to all consistent models.

The computational performance is impressive, given that the space of feasible models, those that are consistent with the structural information, exceeds 2^{100} . On the other hand, the employed steady state assumptions can arguably be seen as strong constraints. It is not clear how SYSBIOX performs on properties other than steady states, for example the aforementioned reachability properties.

Related to the inference problem and its constraint-based solution is the so-called training of models to perturbation data advocated by J. S. Rodriguez et al. [51]. The task here is to select from the set of models that are consistent

with a prior-knowledge interaction map, those that minimize the difference between predictions and observations, in this case normalized phospho-proteomics data. Whereas [51] relied on model selection and a genetic algorithm, i.e., a local search, A. Siegel et al. [52] have recently proposed to employ ASP and a complete model pool analysis. They have also pointed out benefits of a complete analysis, which they variability analysis, to problems regarding robustness and experimental design. An example is the extension of the combinatorial problem of finding intervention strategies, as defined by S. Klamt et al. [53], from single models to the complete set of feasible models, an extension that may intuitively be interpreted as "intervention under uncertainty".

A typical question regarding the design of experiments is to find an experiment, for example a perturbation, that guarantees a maximally divergent behavior between the currently feasible models. The reasoning is that such an experiment, since it discriminates so well between the models, maximizes the gained information and leads to the smallest subsequent model pool.

In contrast to the constraint-based and ASP-based frameworks we propose to study the pros and cons of an explicit approach that enumerates all models that agree with a given partial specification. The idea is to create a universal framework for the analysis of sets of models that is not restricted to specific types of prior knowledge or data.

1.2 Summary

Chapter 2

This chapter discusses the mathematical framework for logical models. It sets out by introducing components as finite domain variables that represent the activities of the substances of the modeled system. The fundamental notions of states and the state space are discussed and a language for referencing subsets and subspaces of states is suggested. The second ingredient of logical models, the update functions, which determine the trajectories in state space, are subsequently introduced. We briefly describe how update functions may be represented by rules rather than tables of input-output values and emphasize a special class logical models, the Boolean networks. The next section defines the interaction graph which is derived from the update functions and contains all information regarding interaction pathways and feedback. The common additional interaction attributes, their sign and threshold, are also mentioned. In addition, the condensation graph, which is closely related to the SCC graph, is defined. It is a layered, acyclic graph in which each node represents a SCC of the interaction graph. It has applications in making analysis algorithms more efficient by dividing a given problem into smaller sub-problems. The next section deals with the state transition graph which relates states by directed transitions. They are in turn derived from the so-called update strategy, an assumption regarding the possibility that two components change activity during a single transition. Two widely used strategies, the fully synchronous and fully asynchronous update, are discussed and the respective transition relations are defined.

The last sections recapitulate model checking and in particular the syntax and semantics of two frequently used temporal logics. An instruction for trans-

lating logical models into the model checking format SMV is also given. The chapter concludes with a model checking benchmark and a case study that illustrates how logical models are constructed.

Chapter 3

This chapter is dedicated to temporal logic specifications in the context of model validation with time series data. Our contribution here is a path-based notion of compatibility. Time series data had previously been used for the reverse engineering of logical networks, but only under the transition-based notion of compatibility, that is, the assumption that successive measurements are represented by synchronous transitions in the model. We discuss various linear time and branching time query patterns and the tool `TEMPORALLOGICTIMESERIES` [54] for the automatic generation of queries from data.

The chapter begins with an introduction to time series data and data discretization. We introduce the notion of compatibility and give a temporal logic encoding. The notion of monotonic compatibility and the nested conditional reachability query are introduced. A proof that deciding monotonic compatibility is at least NP-hard is given. We propose a method for the assessment of the sampling rate of time series data. Variations of reachability queries are discussed, notably branching time compatibility, which is related to robustness. The chapter is concluded by a case study.

Chapter 4

This chapter addresses the asymptotics of logical networks. Our main contribution is an algorithm for the computation of all minimal and maximal symbolic steady states of logical networks in terms of sets of prime implicants. Our definitions generalize the self-freezing circuits of S. Kauffman and F. Fogelman and we establish a connection to positive circuits in the interaction graph.

The chapter begins with the definition of attractors, trap sets and related concepts like basin of attraction and stable and oscillating components. The attractor decision problem and a model checking solution are introduced. Symbolic steady states and seeds are recapitulated and their counterparts in state space are discussed. A natural model reduction technique and a lower bound for the number of cyclic attractors are introduced. The prime implicant graph (PIG) is introduced as a directed hypergraph. A theorem is proved that relates seeds to stable and consistent arc sets in the PIG. A connection to positive circuits in the interaction graph and Thomas' Theorems is explained. The structure of symbolic steady states is discussed. An optimization-based method for computing all maximal seeds is proposed. An ILP and ASP encoding are given. Preliminary results combining model checking with symbolic steady states is presented. The chapter is concluded by a case study of a MAPK signaling network.

Chapter 5

This chapter is divided into three sections that present results regarding uncertainties in the construction of models. Our contribution is the software toolbox

LOGICMODELCLASSIFIER [55] that combines constraint satisfaction for the construction, with databases for the storage and model checking for the selection of logical models.

The chapter begins with a discussion of the iterative process of reverse engineering. Next we introduce a predicate language for the specification of logical models. The section is followed by a description of our software. The section is divided into model specification, model annotation and model analysis.

Chapter 2

Qualitative Models of Regulatory Networks

”We distinguish between two fundamentally different types of mathematical models: causal, theory-like models and non-causal, correlational models. By making causal claims about how certain aspects of a real system function, causal models constitute theories about that system. Therefore, such models can be used for both prediction and explanation. Non-causal mathematical models, on the other hand, simply express observed statistical correlations among various elements of a real system. These models should be used only for prediction purposes, on the assumption that they work only within a certain range of values of variables.”

– Y. Barlas and S. Carpenter in [56]

2.1 The Qualitative Modeling Framework

A qualitative, or logical model of a regulatory network is a tuple (V, F) which describes the components V and the update functions F of the network. In the following two subsections we introduce both objects in terms of finite-valued variables and functions.

2.1.1 The Components

V is a set that consists of n variables $V = \{v_1, \dots, v_n\}$ that represent the network components under consideration of the model. Each variable $v \in V$ has a finite domain $Dom(v) = [0..Max(v)]$ where $Max(v) \in \mathbb{N}_0$ is a constant and

$$[a..b] := \{k \in \mathbb{Z} \mid a \leq k \leq b\}$$

is our notation for a discrete interval. The convention that $Dom(v)$ is a discrete interval with lower bound 0 is not essential to the qualitative framework, whereas the fact that the domains are finite and ordered, is.

A variable is usually identified with a single gene or a single type of protein of the network but it may also represent a family of proteins, a multi-protein

complex or even a complete cellular process like *Apoptosis* or *Proliferation*.

A value $a \in \text{Dom}(v)$ of a variable v represents the activity of the corresponding network component. The use of the term *activity* is deliberately vague as it may represent various different situations. For example:

- at what *qualitative* rate a gene is being transcribed,
- the *qualitative* concentration of a protein,
- the *qualitative* ratio of phosphorylated to un-phosphorylated proteins,
- the *qualitative* temperature, acidity or saltiness of the environment,

If a variable $v \in V$ satisfies $\text{Dom}(v) = \mathbb{B} = \{0, 1\}$, it is understood to be a Boolean variable and we identify the value 0 with the Boolean value *false* and 1 with *true*.

A *state* of the network is an assignment of values to variables, i.e., an unambiguous determination of the activity of every network component. We formalize states by functions: A state is a function x that satisfies

$$x : V \rightarrow \mathbb{N}_0, \forall v \in V : x(v) \in \text{Dom}(v).$$

We specify states by a sequence of $n = |V|$ values that correspond to the components in the order given in V . For example, $x = 1102$ should be read as $x(v_1) = 1, x(v_2) = 1, x(v_3) = 0$ and $x(v_4) = 2$.

The set of all states, called *state space*, of the variables V is denoted by $S(V)$ or simply S if it is clear which variables are meant. Since the activities of components in a state are chosen independently, S can naturally be identified with the Cartesian product

$$\prod_{v \in V} \text{Dom}(v)$$

by specifying some ordering of V . A useful metric for the state space is the Manhattan distance.

Definition 1. The distance $\text{dist}_v(x, y)$ between two states $x, y \in S$ in the direction of $v \in V$ is defined by $\text{dist}_v(x, y) := |x(v) - y(v)|$ and the Manhattan distance $\text{Dist}(x, y)$ by

$$\text{Dist}(x, y) := \sum_{v \in V} \text{dist}_v(x, y).$$

A principle challenge of working with qualitative models is the state explosion problem which is entailed by the above definition of S . It is caused by the fact that the number of states $|S|$ of a network grows exponentially in the number of variables $n = |V|$ and the maximal activities $\text{Max}(v)$ for $v \in V$:

$$|S| = \prod_{v \in V} (\text{Max}(v) + 1).$$

Many interesting model properties are specified on the basis of the large set S . So we need a descriptive language for larger objects than individual states and algorithms that use that language.

A popular approach is to consider sets of states rather than individual states. Apart from algorithmic advantages, a concise language for sets of states makes many statements and descriptions clearer and easier to read. In the following we discuss the view point we adopt in this thesis. The objects that reference the sets of states are called *state descriptions*. As the language of state descriptions we use propositional formulae over atomic propositions that compare a component with an activity. An atomic proposition a is hence a statement

$$a ::= v \diamond k$$

where $v \in V$ is a component, $\diamond \in \Omega := \{=, \neq, <, \leq, >, \geq\}$ a comparison operator and $k \in \mathbb{N}_0$ an activity. For every comparison operator $\diamond \in \Omega$, we denote by $AP_\diamond = AP_\diamond(V)$ the set of all atomic propositions that are restricted to the fixed comparison operator \diamond . The set of all atomic propositions of all types is then denoted by

$$AP = AP(V) := \bigcup_{\diamond \in \Omega} AP_\diamond.$$

Whether an atomic proposition is true or false in a given state is determined by a labeling function $L : S \rightarrow 2^{AP}$ that assigns to every state $x \in S$ a set of propositions $L(x) \subseteq AP$ that will be understood to be true in x while all others are false. Naturally, we define L to agree with the semantics of a state and a proposition:

$$v \diamond k \in L(x) :\Leftrightarrow x(v) \diamond k.$$

Definition 2. A state description d over AP is a formula constructed by the grammar

$$d ::= (v \diamond k) \mid \bar{d} \mid d_1 \cdot d_2 \mid d_1 + d_2$$

where $v \diamond k \in AP$ or the special formula ϵ which we call the empty description. The set of all state descriptions is denoted by $StateDesc = StateDesc(V)$.

Definition 3. The satisfaction relation \models between states and state descriptions is defined by the usual rules of logic:

$$\begin{array}{ll} x \models \epsilon & \text{is true} \\ x \models (v \diamond k) & \text{iff } (v \diamond k) \in L(x) \\ x \models d_1 \cdot d_2 & \text{iff } x \models d_1 \text{ and } x \models d_2 \\ x \models d_1 + d_2 & \text{iff } x \models d_1 \text{ inclusive or } x \models d_2 \\ x \models \bar{d} & \text{iff not } x \models d \end{array}$$

A description d defines a subset $S[d] \subseteq S$ of states by selecting those states that satisfy d :

$$S[d] := \{x \in S \mid x \models d\}.$$

The states $S[d]$ are said to be *referenced* by d , see for example Fig. 2.1.

Observation 1. The states that are referenced by a state description may be given in terms of sub-formulae by the following observation:

$$\begin{aligned} S[d_1 \cdot d_2] &= S[d_1] \cap S[d_2] \\ S[d_1 + d_2] &= S[d_1] \cup S[d_2]. \end{aligned}$$

A useful sub-family of descriptions is the one obtained by allowing only conjunctions of equalities.

Definition 4. A symbolic state of V is a state description that is constructed by the grammar

$$p ::= (v = k) \mid p_1 \cdot p_2$$

where $v = k \in AP_{=}$ or the empty description ϵ . The set of all symbolic states is denoted by $Sym = Sym(V)$.

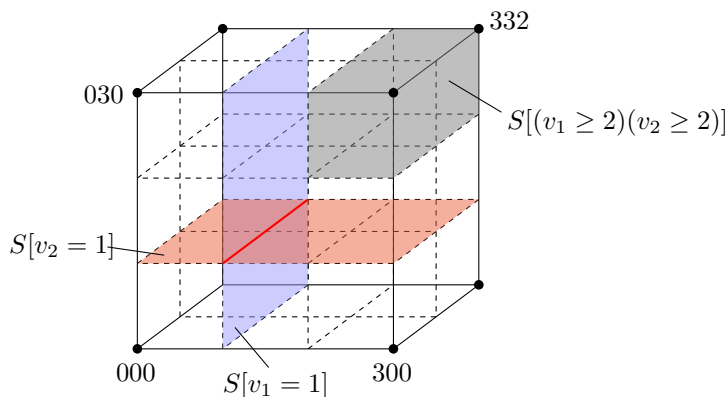


Figure 2.1: The grid structure of the state space of the components $V = \{v_1, v_2, v_3\}$ with maximal activities $Max(v_1) = Max(v_2) = 3$ and $Max(v_3) = 2$. Every intersection of three lines represents a state whose activities are given as a sequence of 3 integers that represent the activities of v_1, v_2 and v_3 respectively. Lines connect states with Manhattan distance equal to 1. The distinction between solid and dashed lines is made only for aesthetic reasons. States in the blue plane are referenced by the symbolic state $(v_1 = 1)$, in the red plane by $(v_2 = 1)$ and in their intersection by $(v_1 = 1)(v_2 = 1)$. The states in the gray box, although rectangular, are not a subspace and not referenced by a symbolic state.

In this setting, we call ϵ the *empty symbolic state*. A symbolic state p is called *consistent* if it references at least one state, i.e., if $S[p] \neq \emptyset$. Two symbolic states p, q are said to be *consistent* if $p \cdot q$ is consistent. A symbolic state may reference no states, for example $(v = 0) \cdot (v = 1)$, or all states because $\epsilon \in Sym$ and $S[\epsilon] = S$. Symbolic states are usually assumed to be consistent unless explicitly stated otherwise.

With the intention of simplifying statements about symbolic states we introduce two additional representations. Let p be the symbolic state $(v_1 = k_1) \cdot \dots \cdot (v_m = k_m)$. First, the *set representation* Set_p of p is defined by

$$Set_p = \{(v_i, k_i) \mid 1 \leq i \leq m\}.$$

The *domain* $V[p]$ of p is defined to consist of the components that appear in the conjunction:

$$V[p] = \{v_i \mid 1 \leq i \leq m\}.$$

Second, the *mapping representation* Map_p of p is a function with range $\{k_1, \dots, k_m\}$ that is defined by

$$Map_p : V[p] \rightarrow \{k_1, \dots, k_m\}, \quad Map_p(v_i) = k_i.$$

The notion of states being symbolic has been applied to questions regarding circuit functionality and the long-term behaviors of a network. They appear, for example, in [57, 58] under the term *singular states*. We have adopted the term *symbolic state* from [40] and will come back to the implications regarding the long-term behaviors in Chap. 4.

Note that symbolic states could also be seen as *partial states* in the sense that $p \in Sym$ is a state of a subset of the variables $V[p] \subseteq V$.

If all variables are Boolean then the state space is also called the *Boolean hypercube* and the symbolic states Sym then correspond to the faces of the Boolean hypercube, in particular $|Sym| = 3^n$. An example is given in Fig. 2.2.

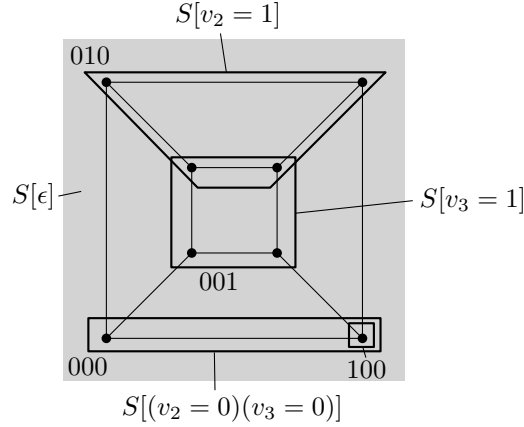


Figure 2.2: The Boolean hypercube for $V = \{v_1, v_2, v_3\}$ and 5 of its $3^3 = 27$ subspaces (faces) and corresponding symbolic states. States with Manhattan distance equal to 1 are connected by lines.

The *size* of $p \in Sym$ is defined to be $|p| := |V[p]|$. For the empty symbolic state we define $V[\epsilon] = \emptyset$ and hence $|\epsilon| = 0$. The notion of certain maximal and minimal symbolic states with respect to the following partial order will be essential to Chap. 4.

Definition 5. We define the following partial order for symbolic states $p, q \in Sym$:

$$p \leq q :\Leftrightarrow Set_p \subseteq Set_q.$$

Note that p and q are therefore comparable if they are consistent and that the notion of size corresponds, intuitively, to the "dimension" of the subspace and not the referenced states. As a corollary we get

$$p \leq q \Leftrightarrow S[p] \supseteq S[q]$$

which may seem counter-intuitive at first.

We specify symbolic states by a sequence of $|p|$ values whose subscript corresponds to the index of the component as given in V . For example, $p = 1_1 0_3$

means $V[p] = \{v_1, v_3\}$ with $\text{Map}_p(v_1) = 1, \text{Map}_p(v_3) = 0$ in mapping representation and $\text{Set}_p = \{(v_1, 1), (v_3, 0)\}$ in set representation.

Observation 2. *Since for any state $x \in S$ there is a unique symbolic state p_x such that $S[p_x] = \{x\}$ there is also, for any subset $S' \subseteq S$, a state description d such that $S[d] = S'$.*

Finally, we identify a state $x \in S$ and a symbolic state $p \in \text{Sym}$ if $S[p] = \{x\}$.

2.1.2 The Update Functions

The second object of a model (V, F) is a set of functions F , called the *update functions* of V , that determine the activities of the components over time. For every $v \in V$ there is exactly one function $f_v \in F$ with $f_v : S \rightarrow \text{Dom}(v)$ that maps a state $x \in S$ into the domain $\text{Dom}(v)$ of v . Note that sometimes we will use the indexed notation $v_i \in V$ and $f_i \in F$ to indicate that f_i is the update function of v_i .

Since S is a Cartesian product of finite sets, the functions $f \in F$ belong to the family of n -variable discrete functions with $n = |V|$.

The image $f_v(x) \in \text{Dom}(v)$ is called the *target activity* of v in x . It determines the response of v to the state x , i.e., the activity that v tends towards when the network is in state x .

The combined response of the network to a state is called *target state* and denoted by $F(x)$ where the set F is interpreted as a function $F : S \rightarrow S$ that assigns to each component $v \in V$ its target activity $f_v(x)$, i.e., for all $v \in V : F(x)(v) := f_v(x)$. The rules by which the target activities unfold the global dynamics of a network are discussed below in Sec. 2.3.

A qualitative model (V, F) of a regulatory network therefore demands a full specification of F by, for example, input-output tuples $(x, f_v(x))$ for every state $x \in S$ and every component $v \in V$. Functions that show up in applications have, however, often a relatively simple structure that allows for a more condensed representation of the update function. A typical case where this is possible are *Boolean networks* for which we can define F by Boolean expressions. A *Boolean expression* is a formula constructed by the grammar

$$f ::= 0 \mid 1 \mid v \mid \bar{f} \mid f_1 \cdot f_2 \mid f_1 + f_2$$

where $v \in V$ signifies a variable, \bar{f} the negation, $f_1 \cdot f_2$ the conjunction and $f_1 + f_2$ the (inclusive) disjunction of the expressions f, f_1 and f_2 . Since Boolean networks are predominant among publications about qualitative regulatory networks and since they often serve as good illustrations for otherwise technical definitions, we take a paragraph to define them.

Definition 6. *A network (V, F) is Boolean iff all its components are Boolean. In Boolean networks, every update function $f \in F$ is therefore a n -variable Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ which we can represent by a Boolean expression over the n input variables V .*

Given a state $x \in S$ of a Boolean network, each expression $f \in F$ can be evaluated by substituting the value $x(v)$ for each occurrence of the variable v in f and applying the rules of logic.

The idea of defining a function by a formula extends to general discrete functions. But, since the input variables V may not be Boolean, we must resort to state descriptions instead of Boolean expressions. By the same reason, since the target value of $f(x)$ may not be Boolean, we must specify the implied value explicitly.

Hence, whereas the representation of a Boolean function consists of a single Boolean expression, the formula-based representation of a general discrete function f requires a set of so-called rules, where each rule specifies all states for which f_i is constant at a certain value.

Definition 7. A rule is a tuple $(r, (v = k))$ where r is a state formula and $(v = k) \in AP_{=}$ an atomic equality. We denote a rule $(r, (v = k))$ by $r \Rightarrow (v = k)$. A rule $r \Rightarrow (v = k)$ is valid for a given f_v if

$$\forall x \in S[r] : f_v(x) = k.$$

Definition 8. A set of rules $\{r_i \Rightarrow (v = k_i) \mid 1 \leq i \leq m\}$ represents a function $f_v \in F$, iff each rule is valid and every state $x \in S$ is referenced by some rule:

$$S[r_1 + \dots + r_m] = S.$$

Observation 3. For any function $f_v \in F$ there is a set of rules that represents it. The natural construction is for every $k \in \text{Dom}(v)$ to determine the subset $S_k := \{x \in S \mid f_v(x) = k\}$ of k pre-images under f_v and then to find, by Observation 2, a state description r_k that references S_k . Clearly, the rules $\{(r_k \Rightarrow (v = k)) \mid k \in \text{Dom}(v)\}$ then represent f_v .

(A)	<table> <tr> <th>$x = (u, v, w)$</th> <th>$f_v(x)$</th> </tr> <tr><td>(0, 0, 0)</td><td>1</td></tr> <tr><td>(0, 0, 1)</td><td>0</td></tr> <tr><td>(1, 0, 0)</td><td>1</td></tr> <tr><td>(1, 0, 1)</td><td>1</td></tr> <tr><td>(0, 1, 0)</td><td>1</td></tr> <tr><td>(0, 1, 1)</td><td>0</td></tr> <tr><td>(1, 1, 0)</td><td>1</td></tr> <tr><td>(1, 1, 1)</td><td>1</td></tr> </table>	$x = (u, v, w)$	$f_v(x)$	(0, 0, 0)	1	(0, 0, 1)	0	(1, 0, 0)	1	(1, 0, 1)	1	(0, 1, 0)	1	(0, 1, 1)	0	(1, 1, 0)	1	(1, 1, 1)	1	(B)	$f_v = u + \overline{w}$
$x = (u, v, w)$	$f_v(x)$																				
(0, 0, 0)	1																				
(0, 0, 1)	0																				
(1, 0, 0)	1																				
(1, 0, 1)	1																				
(0, 1, 0)	1																				
(0, 1, 1)	0																				
(1, 1, 0)	1																				
(1, 1, 1)	1																				
		(C)	$(u = 1) + (w = 0) \Rightarrow (v = 1)$ $(u = 0) \cdot (w = 1) \Rightarrow (v = 0)$																		
		(D)	$(u = 1) \Rightarrow (v = 1)$ $(w = 0) \Rightarrow (v = 1)$ $(u = 0) \cdot (w = 1) \Rightarrow (v = 0)$																		

Figure 2.3: The four different representations of an update function $f_v \in F$ of a Boolean network with $V = \{u, v, w\}$. (A) input-output representation, (B) a Boolean expression representation, (C) a rule based representation, and (D) another rule-based representation.

2.2 The Interaction Graph

The interaction graph can be seen as an abstraction of a model, it captures only the dependencies between the variables as inherent in the update functions

but discards the details. As a consequence two models may have the same interaction graph. The structure of the interaction graph, and in particular the presence or absence of positive and negative feedback, is useful when predicting the dynamics of the respective models.

For the definitions in this section it is convenient to introduce the unit activity $e_u \in S$ for $u \in V$ by $e_u(v) := 1$ if $u = v$ and 0 otherwise, and a shorthand for the addition of states $x, y \in S$ by $(x \oplus y) \in S$ and $(x \oplus y)(v) := x(v) + y(v)$. As in [59] we say that $f_v \in F$ depends on $u \in V$ if there are $1 \leq t \in \text{Dom}(u)$ and $x \in S[u = t - 1]$ such that

$$f_v(x) \neq f_v(x \oplus e_u). \quad (2.1)$$

The dependencies are usually visualized by the following graph-based representation.

Definition 9. *The interaction graph of (V, F) is the directed graph (V, \rightarrow) that consists of the node set V and the arc set $\rightarrow_F \subseteq V \times V$ with $(u, v) \in \rightarrow$ iff f_v depends on u .*

For convenience we write $u \rightarrow v$ and $u \not\rightarrow v$ instead of $(u, v) \in \rightarrow$ and $(u, v) \notin \rightarrow$. If $u \rightarrow v$ then we say that u is a *regulator* of v and that the interaction (u, v) is *present* or *observable* in (V, F) .

The interaction graph is usually labeled by additional information. The value t in the above definition of dependency is, for example, often interpreted as a threshold value of concentration-dependent interactions. We define the set of *thresholds* $T(u, v) \subset \text{Dom}(v)$ of an interaction $u \rightarrow v$ to consist of all $1 \leq t \in \text{Dom}(u)$ that satisfy Eq. 2.1. We may use $u \xrightarrow{t_1, \dots, t_i} v$ to indicate that $\{t_1, \dots, t_i\} \subseteq T(u, v)$ is a subset of the thresholds of an interaction. In Boolean models the set of thresholds of any interaction is always $T(u, v) = \{1\}$.

Another characteristic of interactions is whether they are activating or inhibiting or both. We define the *sign* of an interaction $\text{Sign}(u \xrightarrow{t} v) \subseteq \{+, -\}$ and one of its thresholds $t \in T(u, v)$ according to the ways in which Eq. 2.1 can be satisfied.

$$\begin{aligned} + \in \text{Sign}(u \xrightarrow{t} v) &\Leftrightarrow \exists x \in S[u = t - 1] : f_v(x) < f_v(x \oplus e_u) \\ - \in \text{Sign}(u \xrightarrow{t} v) &\Leftrightarrow \exists x \in S[u = t - 1] : f_v(x) > f_v(x \oplus e_u) \end{aligned}$$

Note that $\text{Sign}(u \xrightarrow{t} v) = \{+, -\}$ is therefore allowed in which case the interaction $u \xrightarrow{t} v$ is activating as well as inhibiting. We may combine the threshold and sign and write, for example, $u \xrightarrow{+t} v$ to indicate that $+ \in \text{Sign}(u \xrightarrow{t} v)$. If u is a Boolean variable then the thresholds must be equal to $\{1\}$. We may omit them and write $u \xrightarrow{+} v$ instead of $u \xrightarrow{+1} v$.

2.2.1 Condensation and SCCs

The interaction graph is already an abstraction, but further abstractions, which are called the *condensation graph* and the *SCC graph*, are also useful. In particular, many computational problems can be split into sub-problems that are defined by the nodes of the condensation graph. In this section we introduce

basic concepts that will be useful in Chap. 4, for extensions of these concepts see e.g. [60]. The definitions rely on the following standard terminology for graphs and directed graphs.

The *predecessors* and *successors* of $v \in V$ in a digraph (V, \rightarrow) are

$$\begin{aligned} \text{Pred}_{\rightarrow}(v) &= \text{Pred}(v) := \{u \in V \mid u \rightarrow v\}, \text{ and} \\ \text{Succ}_{\rightarrow}(v) &= \text{Succ}(v) := \{w \in V \mid v \rightarrow w\}. \end{aligned}$$

A *finite path* in (V, \rightarrow) is a sequence (v_0, v_1, \dots, v_k) of $v_i \in V$ such that $v_i \rightarrow v_{i+1}$ for all $0 \leq i < k$. Note that $k \in \mathbb{N}_0$ indicates the length of the path in terms of the number of interactions rather than the number of components. A path of length 0 consists therefore of a single node and is permissable. The existence of a path from u to v is indicated by $u \rightsquigarrow v$.

Definition 10. A strongly connected component (*SCC*) of a digraph (V, \rightarrow) is an inclusion-wise maximal subset $U \subseteq V$ that satisfies $u \rightsquigarrow v$ for all $u, v \in U$. We denote the set of SCCs by $\text{SCC}(V, \rightarrow)$.

Note that it is possible that $v \in V$ does not belong to any SCC, namely if there is no path $v \rightsquigarrow v$. We call $v \in V$ that do not belong to a SCC *cascade components* and denote them by $\text{Cascade}(V, \rightarrow)$. Similarly, *undirected paths* and *(weakly) connected components* are defined by considering the undirected interaction graph that is obtained by ignoring the orientation of the edges.

A component that has no predecessors in the interaction graph is called a *constant* of the model. A constant $v \in V$ hence satisfies $f_v(x) = k$ for all $x \in S$ and some fixed $k \in \text{Dom}(v)$ and we may write $f_v = k$ to indicate that f_v is constant. A component that has only itself as a predecessor is called an *input* of the model. There are two types of inputs. A *stable input* $v \in V$ satisfies $f_v(x) = x(v)$ for all $x \in S$. An input that is not stable is also called an *unstable input*. Note that constants are cascade components while inputs are SCCs of size 1.

With these definitions in place we are now interested in information regarding the hierarchy of the interactions between components. The first observation is that the interaction graph may not be (weakly) connected. In such a case the system consists of several independent sub-models that can, for most questions, be analyzed separately. Each connected component is in turn partitioned into SCC components and cascade components. The *condensation graph* captures the interactions between SCCs and cascade components.

Definition 11. The condensation graph of a digraph (V, \rightarrow) is the digraph (C, \dashrightarrow) where $C := \text{Cascade}(V, \rightarrow) \cup \text{SCC}(V, \rightarrow)$ and the presence of an arc is determined for $c_1, c_2 \in \text{Cascade}(V, \rightarrow)$ and $U_1, U_2 \in \text{SCC}(V, \rightarrow)$ by:

$$\begin{aligned} c_1 \dashrightarrow c_2 &: \Leftrightarrow c_1 \rightarrow c_2 \\ c_1 \dashrightarrow U_1 &: \Leftrightarrow \exists u \in U_1 : c_1 \rightarrow u \\ U_1 \dashrightarrow c_1 &: \Leftrightarrow \exists u \in U_1 : u \rightarrow c_1 \\ U_1 \dashrightarrow U_2 &: \Leftrightarrow U_1 \neq U_2, \exists u_1 \in U_1, u_2 \in U_2 : u_1 \rightarrow u_2 \end{aligned}$$

The *SCC graph* is similar to the condensation graph, but its node set consists only of SCCs and discards the cascade components. It is useful for defining the hierarchies between components as inherent in the interaction graph.

Definition 12. The SCC graph of a digraph (V, \rightarrow) is the digraph (D, \succ) where $D := SCC(V, \rightarrow)$ and $U_1 \succ U_2$ iff $U_1 \dashrightarrow U_2$ in the condensation graph (C, \dashrightarrow) or there is a path $U_1 \dashrightarrow c_1 \dots c_k \dashrightarrow U_2$ in (C, \dashrightarrow) such that $c_1, \dots, c_k \in Cascade(V, \rightarrow)$.

Note that the SCC graph (D, \succ) is acyclic. We now assign a number $0 \leq layer(U) \in \mathbb{N}$ to each $U \in SCC(V, \rightarrow)$ such that $layer(U_i) = k$ implies that there is no path from $U_j \succ \dots \succ U_i$ for all U_j with $layer(U_j) \leq k$. The motivation for dividing components into layers is that since the first k layers (for every $0 \leq k$) are upstream of all components in the remaining layers, its dynamics is effectively independent of the components downstream. This observation can be exploited for questions regarding, for example, the asymptotics of a model.

We define the layers in terms of the longest paths leading into a SCC. We set $layer(U) := 0$ for all $U \in D$ that have no predecessors in (D, \succ) . Since (D, \succ) is acyclic there is at least one such U . For $U \in SCC(V, \rightarrow)$ with predecessors we define

$$layer(U) := \max(\{m \mid U_0 \succ \dots \succ U_m = U \text{ is a path in } (D, \succ)\}).$$

Since (D, \succ) is acyclic, $layer(U)$ is well defined. All input components belong to layer 0. Note that the layers are similar to a topological sort but that they are also unique. An example is given in Fig. 2.4.

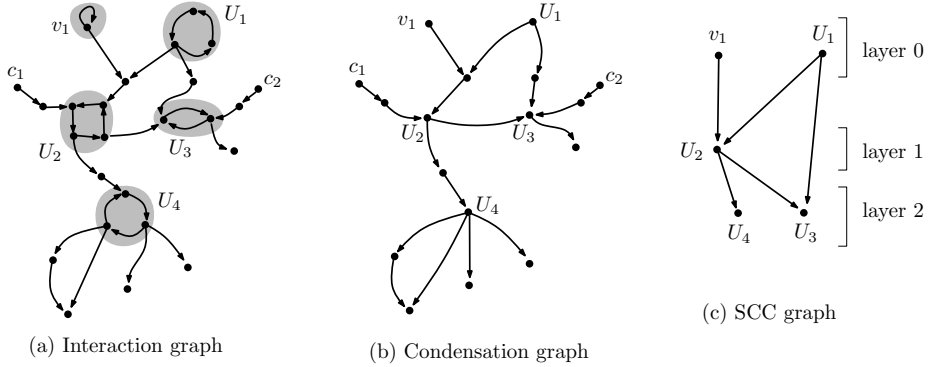


Figure 2.4: (a) The interaction graph (V, \rightarrow) with $U_i \in SCC(V, \rightarrow)$, $c_1, c_2 \in V$ constants, $v_1 \in V$ an input and the remaining components are cascade components. Note that $\{v_1\} \in SCC(V, \rightarrow)$. (b) The condensation graph (C, \dashrightarrow) is obtained by replacing the SCCs with single nodes, see Def. 11. (c) The SCC graph (D, \succ) consists only of nodes that represent SCCs in the interaction graph. There is an edge from U_i to U_j if there is a path of cascade components from U_i to U_j . This graph has 3 layers.

2.3 The State Transition Graph

2.3.1 Transition Rules

A transition rule specifies how F determines for every $x \in S$ a set of states that are reachable by a single transition. The resulting relation between the

states of a model is usually thought of as a directed graph (S, \rightarrow) called the *state transition graph*, or STG for short. The STG is the basis for the analysis and for simulations of the dynamics of a qualitative model.

There are many different definitions for transition rules. For an overview see for example [61–64]. The methods developed in this thesis are not restricted to a particular transition rule. As representative examples we define the *synchronous update*, as introduced by S. Kauffman in [12] to study Boolean models of gene regulatory networks, the *asynchronous update* that was introduced by R. Thomas in [65] with the intention of including varying time scales in models of gene regulation, and the *mixed update* that contains both synchronous and asynchronous transitions but also partially synchronous transitions.

The essential principle behind the existence of a transition is that each state is attracted to its target state. Formally, a transition $x \rightarrow y$ must reduce the distance to the target state $F(x)$. That is, a transition relation satisfies

$$x \rightarrow y \Rightarrow \text{Dist}(x, F(x)) \geq \text{Dist}(y, F(x)). \quad (\text{P1})$$

Second, a transition $x \rightarrow y$ is often required to be *quasi-continuous* in every $v \in V$:

$$x \rightarrow y \Rightarrow \forall v \in V : \text{dist}_v(x, y) \leq 1 \quad (\text{P2})$$

There is some freedom of choice regarding whether every state is required to have a successor, i.e., whether transition relations are required to be *total*, and the use of self-loops, i.e., for which states $x \rightarrow x$ holds. For our purposes it turned out to be convenient to require that transition relations are total

$$\forall x \in S : |\text{Succ}(x)| \geq 1 \quad (\text{P3})$$

and that self-loops are characteristic of fixpoints of F

$$x \rightarrow x \Leftrightarrow F(x) = x. \quad (\text{P4})$$

Transition relations are conveniently defined by the *tendencies* of components.

Definition 13. The tendency $\text{tend}_v(x) \in \{-1, 0, 1\}$ of a component $v \in V$ in a state $x \in S$ is defined by

$$\text{tend}_v(x) := \begin{cases} 1 & : f_v(x) > x(v) \\ 0 & : f_v(x) = x(v) \\ -1 & : f_v(x) < x(v) \end{cases}.$$

The Synchronous Update

The synchronous transition relation is denoted by \rightarrow and demands that every component's tendency is realized in a single transition:

$$x \rightarrow y \Leftrightarrow \forall v \in V : y(v) = x(v) + \text{tend}_v(x)$$

It is not hard to see that the synchronous transition relation respects the requirements (P1) – (P4). The resulting STG is deterministic in the sense that $|\text{Succ}(x)| = 1$ for all $x \in S$. Deterministic updates are popular in modeling network dynamics because they are easier to handle, conceptually as well as computationally, than non-deterministic updates. For Boolean networks an equivalent definition is $x \rightarrow y \Leftrightarrow y = F(x)$.

The Asynchronous Update

One motivation for considering asynchronous updates is that the modeled processes may take place on different time scales. A synthesis rate may, for example, be much higher than the degradation rate or vice versa, for a given protein. To incorporate this, target activities are usually interpreted as calling activities to change rather than stating that they have changed. If no counter-call is received then the activities will eventually change after some specific delay. In general, the delays of different processes are not equal and hence a race to change decides which activity is updated first.

A second motivation for studying the asynchronous update is that connections have been established between the dynamics of a logical models and a similar system of piecewise linear differential equations, see for example [2, 66, 67]. In that setting, the asynchronous STG can be seen as an approximation to the phase space and trajectories of the differential equations.

The asynchronous transition relation \hookrightarrow introduced here assumes that the delays are not known and that any one process may finish first.

$$x \hookrightarrow y :\Leftrightarrow \begin{cases} \text{either} & F(x) = y \wedge x = y \\ \text{or} & \text{Dist}(x, y) = 1 \wedge \exists v \in V : y(v) - x(v) = \text{tend}_v(x) \end{cases}$$

Again, it is not hard to see that this relation respects the principles (P1) – (P4). It results in a STG that is non-deterministic in the sense that $|Succ(x)| \geq 1$ for all $x \in S$. Non-deterministic STGs are usually seen as over-approximations of the modeled system in the sense that not every path in the STG is predictive for a behavior of the biological system.

The Mixed Update

The mixed transition relation is the most general transition relation. It is denoted by \rightsquigarrow and defined by permitting a change in an arbitrary number of components. That is, $x \rightsquigarrow y$ if either $F(x) = y$ and $x = y$, i.e., if $x = y$ is a steady state, or if there is a non-empty $U \subseteq V$ such that $y(v) \neq x(v)$ and $y(v) - x(v) = \text{tend}_v(x)$ for all $v \in U$.

We use \rightarrow to denote any transition relation that satisfies (P1) – (P4), and \rightarrow , \hookrightarrow or \rightsquigarrow if we want to specifically denote synchronous, asynchronous or mixed transitions.

2.3.2 Trajectories

We denote the set of all *infinite paths* (x_0, x_1, \dots) in (S, \rightarrow) by $Paths_\infty = Paths_\infty(S, \rightarrow)$ and the set of all paths, finite or infinite, by $Paths$. As for the interaction graph, the existence of a path from x to y is denoted by $x \rightsquigarrow y$. As shortcuts for denoting a state, prefix, suffix or fragment of a path we use the square bracket notation as in [16]: The i^{th} state of an infinite path $\pi = (x_0, x_1, \dots)$ is denoted by $\pi[i] := x_i$, a prefix is denoted by $\pi[..k] := (x_0, \dots, x_k)$, a suffix by $\pi[k..] := (x_k, x_{k+1}, \dots)$ and a fragment by $\pi[k..l] := (x_k, \dots, x_l)$. The length of a finite path $\pi = (x_0, \dots, x_r)$ is defined to be the number of occurring transitions rather than states and is denoted by $len(\pi) = r$.

Furthermore we use the following notation for paths with a given initial state $x \in S$ or set of initial states $I \subseteq S$:

$$\begin{aligned} Paths(x) &:= \{\pi \in Paths \mid \pi[0] = x\} \\ Paths(I) &:= \{\pi \in Paths \mid \pi[0] \in I\} \end{aligned}$$

So far we have described how (S, \rightarrow) is obtained from (V, F) . One may also ask the reverse question: Given a digraph (S, \rightarrow) , is there is a model (V, F) such that its STG is equal to (S, \rightarrow) ? The question is, for example, relevant for the reverse engineering of models from data and addressed in e.g. [68, 69]. For our purposes this question is relevant because in Chaps. 3 and 4 we will frequently want to point out the existence of a Boolean network with a given asynchronous STG. Rather than defining F we will draw (S, \leftrightarrow) and make sure that a corresponding F exists. The rules that ensure the existence of F are the following.

Observation 4. *Let S be the state space some Boolean variables V . For every subset $E \subseteq S \times S$ that satisfies $(x, y) \in E \Rightarrow Dist(x, y) \leq 1$ and $(x, x) \in E \Leftrightarrow Succ_E(x) = \{x\}$ there is a Boolean network (V, F) such that $(S, \leftrightarrow) = (S, E)$.*

In practice we can therefore choose for each pair of neighbouring states one of the four orientations in Fig. 2.5 (b). If we then add self-loops to all states with out-degree 0 the resulting digraph will be equal to (S, \leftrightarrow) of some Boolean network (V, F) .

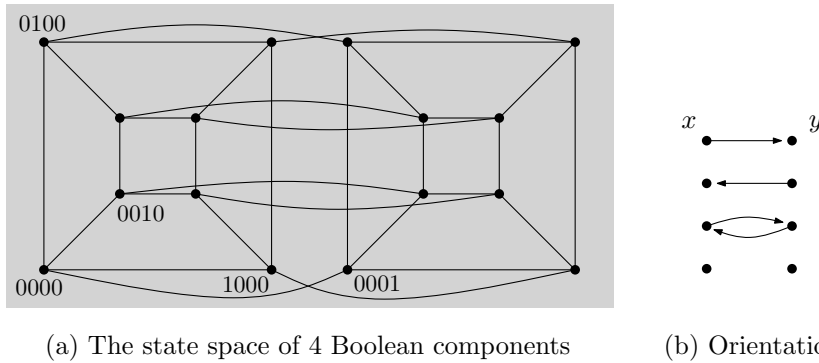


Figure 2.5: (a) The state space of 4 Boolean components, states with Manhattan distance equal to 1 are connected by an undirected edge. If each edge in (a) is replaced by one of the four orientations in (b), and if self-loops are added to all states with out-degree 0 then the resulting directed graph will correspond to the asynchronous STG (S, \leftrightarrow) of some Boolean network (V, F) , see Obs. 4.

2.4 Temporal Logic

Model checking is a formal method from computer science that solves the problem of deciding whether a temporal logic specification is satisfied by a given transition system. Its main area of application are safety critical, complex systems and in particular the design of reliable electronic circuits. E. M. Clarke,

E. A. Emerson and J. Sifakis have together been awarded the 2007 Turing Award for their contribution in developing model checking into an effective technology that is widely used in the industries [70].

The idea to apply model checking to systems biology was proposed around 2003 in [13, 71] and has since been extended by various groups, see for example [18, 72–74] and [75] for a review. As a result, there are now a number of different languages and corresponding model checking algorithms and tools.

The benefit of the model checking machinery for the analysis of qualitative models is the efficiency of available algorithms, in particular the symbolic representation of states, and the expressiveness of the specification languages.

In this section we briefly recapitulate the basic definitions and the two fundamental specification languages, *linear time logic* (LTL) and *computation tree logic* (CTL). The intention is to prepare the theoretical background for Chap. 3 and 4. The definitions and notation are based on [16].

Transition Systems

For our purposes, a *transition system* is a state transition graph together with state-specific information regarding the activities and target states. Formally 5-tuple

$$TS = (S, \rightarrow, AP, L, I)$$

where (S, \rightarrow) is a state transition graph, AP is the set of atomic propositions, $L : S \rightarrow 2^{AP}$ is the labeling function for state descriptions, and $I \subseteq S$ a set of initial states. For an increased expressiveness we add new symbols to the atomic state descriptions AP . The first one is δ_v with $v \in V$ and used in propositions of the form $\delta_v \diamond k$ where $\diamond \in \Omega, k \in \{-1, 0, 1\}$. It relates directly to the tendency of a component, see Def. 13. The labeling function L is extended by:

$$\delta_v \diamond k :\Leftrightarrow tend_v(x) \diamond k.$$

The second one is Δ and used in the form $\Delta \diamond k$ with $\diamond \in \Omega, k \in [0..n], n = |V|$. It refers to the number of components whose activities are equal to their target activities:

$$\Delta \diamond k \in L(x) :\Leftrightarrow |\{v \in V \mid tend_v(x) \neq 0\}| \diamond k.$$

With these new propositions we may now reference states by the component's activities, as in $S[(u = 0)(v > 2)]$, or by the component's tendencies, for example $S[(\delta_v = 0)(\Delta \leq 2)]$, or by a combination of both. For example, $(\Delta = 0)(v = 1)$ references all fixpoints that lie in the $(v = 1)$ subspace:

$$S[(\Delta = 0)(v = 1)] = \{x \in S \mid F(x) = x \wedge x(v) = 1\}.$$

The relevance of fixpoints and other asymptotic behaviors of models is discussed in Chap.4.

Since throughout this thesis the atomic propositions AP and the labeling function L do not change anymore, we will specify transition systems as 3-tuples $TS = (S, \rightarrow, I)$.

2.4.1 Linear Time Logic

Linear time logic (LTL) is a specification language that concerns individual, infinite paths in the transition system. The language consists, in addition to

the Boolean operators, of linear time operators that permit the specification that given sub-formulae should hold for certain fragments of the path. The syntax of LTL is defined as follows.

Definition 14. An LTL formula ϕ over the set of atomic propositions AP is formed according to the following grammar:

$$\phi ::= \text{true} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \mathbf{X}\phi \mid \mathbf{F}\phi \mid \mathbf{G}\phi \mid \phi_1 \mathbf{U}\phi_2$$

where $a \in AP$.

The satisfaction relation $\pi \models \phi$ that defines whether a path $\pi \in \text{Paths}_\infty(S)$ satisfies an LTL formula ϕ is given in Table 2.1.

$\pi \models \text{true}$	
$\pi \models a$	iff $a \in L(\pi[0])$
$\pi \models \phi_1 \wedge \phi_2$	iff $\pi \models \phi_1$ and $\pi \models \phi_2$
$\pi \models \neg\phi$	iff $\neg(\pi \models \phi)$
$\pi \models \mathbf{X}\phi$	iff $\pi[1 \dots] \models \phi$
$\pi \models \mathbf{F}\phi$	iff $\exists 0 \leq j : \pi[j \dots] \models \phi$
$\pi \models \mathbf{G}\phi$	iff $\forall 0 \leq j : \pi[j \dots] \models \phi$
$\pi \models \phi_1 \mathbf{U}\phi_2$	iff $\exists 0 \leq j : \pi[j \dots] \models \phi_2$ and $\forall 0 \leq i < j : \pi[i \dots] \models \phi_1$

Table 2.1: The definition of the satisfaction relation \models for an individual, infinite path $\pi \in \text{Paths}_\infty(S)$ and a LTL formula ϕ .

The operators are usually called "next" for \mathbf{X} , "finally" for \mathbf{F} , "globally" for \mathbf{G} and "until" for \mathbf{U} . The logical disjunction $\phi_1 \vee \phi_2$ and implication $\phi_1 \Rightarrow \phi_2$ are defined by their equivalences involving \neg and \wedge . Note that these linear time operators allow syntactically different formulae that are semantically identical, for example $\mathbf{F}\phi \equiv \text{true} \mathbf{U}\phi$. In fact, the full expressiveness of the above operators is already achieved with formulae involving only \mathbf{U} and \mathbf{X} , but for our purposes it is convenient to also include \mathbf{F} and \mathbf{G} . Also note that the until operator may be true even though ϕ_1 is always false: if $\pi \models \phi_2$ then $\pi \models (\phi_1 \mathbf{U}\phi_2)$ for any ϕ_1 .

The satisfaction relation extends from paths to transition systems by requiring that every path $\pi \in \text{Paths}_\infty(I)$ of the set $I \subseteq S$ of initial states satisfies the formula. For our purposes, however, it is convenient to introduce the quantified satisfaction relations \models^\exists and \models^\forall . They are defined for a $TS = (S, \rightarrow, I)$ and a LTL formula ϕ by

$$\begin{aligned} TS \models^\exists \phi & \text{ iff } \exists x \in I : \exists \pi \in \text{Paths}_\infty(x) : \pi \models \phi \\ TS \models^\forall \phi & \text{ iff } \forall x \in I : \forall \pi \in \text{Paths}_\infty(x) : \pi \models \phi. \end{aligned} \quad (2.2)$$

Again, these are just convenient shortcuts and the following equivalence holds:

$$TS \models^\exists \phi \Leftrightarrow \neg \left(TS \models^\forall \neg\phi \right).$$

Note that Eq. 2.2 involves two quantifiers, one for the initial states and one for the paths that are rooted in the initial states. So far we have considered \models^\exists

which uses \exists for both and \models^{\forall} which uses \forall twice. The reason is that these are decidable with standard model checking algorithms. Two more satisfaction relations are theoretically possible, namely

$$\begin{aligned} TS \models^{\exists\forall} \phi & \text{ iff } \exists x \in I : \forall \pi \in Paths_{\infty}(x) : \pi \models \phi \\ TS \models^{\forall\exists} \phi & \text{ iff } \forall x \in I : \exists \pi \in Paths_{\infty}(x) : \pi \models \phi. \end{aligned} \quad (2.3)$$

but neither are decidable with standard LTL model checking algorithms.

2.4.2 Computation Tree Logic

Whereas LTL is a linear time logic whose satisfaction relation is based on individual paths, *computation tree logic* (CTL) is a branching time logic. Branching time specifications depend on trees and sub-trees of alternative paths which are rooted in states $x \in S$. The universal "for all paths" **A** and the existential "there is a path" **E** specify the dependence while the linear time operators **F**, **X**, **G**, **U** specify the paths. Hence, the syntax of CTL is divided into state formulae and path formulae where the former quantifies and logically connects the later.

Definition 15. A CTL state formula ψ over the set of atomic propositions AP is formed according to the following grammar:

$$\psi ::= true \mid a \mid \psi_1 \wedge \psi_2 \mid \neg\psi \mid \mathbf{E} \varphi \mid \mathbf{A} \varphi$$

where $a \in AP$ and φ is a CTL path formula formed according to the grammar:

$$\varphi ::= \mathbf{X} \psi \mid \mathbf{F} \psi \mid \mathbf{G} \psi \mid \psi_1 \mathbf{U} \psi_2$$

where ψ, ψ_1 and ψ_2 are CTL state formulae.

Path formulae are therefore defined in terms of state formulae and vice versa but CTL model checking requires a CTL state formulae which, for brevity, we simply call *CTL formulae*. Path formulae are identical to LTL formulae but defined over an extended set of propositions involving the state formulae. The satisfaction relation that defines whether a state satisfies a CTL formula is defined in Fig. 2.6.

As for LTL, we derive the quantified satisfaction relations \models^{\exists} and \models^{\forall} for CTL formulae ψ and transition systems $TS = (S, \rightarrow, I)$ by

$$\begin{aligned} TS \models^{\exists} \psi & \text{ iff } \exists x \in I : x \models \psi \\ TS \models^{\forall} \psi & \text{ iff } \forall x \in I : x \models \psi. \end{aligned}$$

Note that, as opposed to LTL, there are no other quantified satisfaction relations because the previously second path quantifier is now part of the path formulae. But, the relations of Eq. 2.3 are not equivalent to CTL. They are neither expressible in pure LTL nor CTL. Finally it should be remarked that LTL and CTL are not comparable in terms of their expressiveness, i.e., depending on the kind of property we wish to check we may not have a choice as to formulating it in either LTL or CTL. Also, the available model checking algorithms for the two languages differ in their time and space complexities, see [16] for model checking algorithms, complexities and a comparison regarding the expressive powers of LTL and CTL.

$x \models \text{true}$	
$x \models a$	iff $a \in L(x)$
$x \models \neg\psi$	iff not $x \models \psi$
$x \models \psi_1 \wedge \psi_2$	iff $x \models \psi_1$ and $x \models \psi_2$
$x \models \mathbf{E} \varphi$	iff $\exists \pi \in \text{Paths}_\infty(x) : \pi \models \varphi$
$x \models \mathbf{A} \varphi$	iff $\forall \pi \in \text{Paths}_\infty(x) : \pi \models \varphi$
$\pi \models \mathbf{X} \psi$	iff $\pi[1] \models \psi$
$\pi \models \mathbf{F} \psi$	iff $\exists 0 \leq i : \pi[i] \models \psi$
$\pi \models \mathbf{G} \psi$	iff $\forall 0 \leq i : \pi[i] \models \psi$
$\pi \models \psi_1 \mathbf{U} \psi_2$	iff $\exists 0 \leq j : \sigma[j] \models \psi_2$ and $\forall 0 \leq i < j : \sigma[i] \models \psi_1$

Figure 2.6: (top) The satisfaction relation $x \models \psi$ for states $x \in S$ and CTL state formulae ψ . (bottom) The satisfaction relation for $\pi \models \phi$ for paths $\pi \in \text{Paths}_\infty(x)$ and CTL path formulae.

2.4.3 NuSMV Model Checking

NuSMV is a model checking software [76, 77] which is based on SMV, the original symbolic model verifier that was developed by E. M. Clarke et al. [78]. It features BDD-based and SAT-based algorithms for the automatic verification of LTL and CTL specifications for finite transition systems.

The first aim of this section is to illustrate how a model (V, F) may be translated into the NuSMV language. In practice, the description is then passed to the software together with a LTL or CTL specification for verification. Although encodings are frequently used [14, 43] they are rarely discussed, for an exception see [79]. The second part consists of a benchmark for logical networks that attempts to determine the size of the largest networks that are still tractable with NuSMV.

Encodings

For our purposes, the system description consists of declaring the variables, the transition system and the temporal specifications. The transition system is essentially defined by a set of equations whose solutions determine the next state for a given current state, just as we used $F(x)$ to define the successors of x in Sec. 2.3. NuSMV allows the use of non-deterministic equations, which are treated by considering all possible solutions, and hence the definition of several successors of a state.

NuSMV's system description is organized into a hierarchy of communicating modules, each behaving like a sub-system with their own inputs and variables. But, compared with industrial systems, the STGs of qualitative models are rather simple and we can define them within a single module, which by convention must be called `main`. For a working example of a NuSMV system description see Fig. 2.7.

Next, we discuss how to translate the synchronous and asynchronous STGs of a model into the NuSMV language. Starting point is the previous example of Fig. 2.7 and a rule-based representation for every $f \in F$. The missing ingredients are the requirements that transitions are quasi-continuous and, in the case of

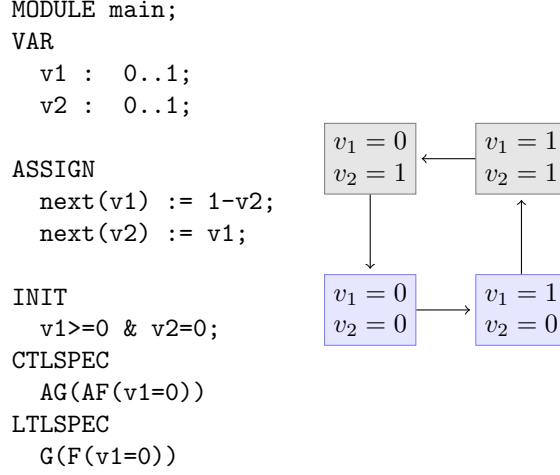


Figure 2.7: (a) An example of a complete system description in the NuSMV language. It consists of several blocks of declarations that are identified by capital letters. The first of them, **MODULE**, sets up the environment in which we define the transition system, **VAR** defines the state variables, **ASSIGN** defines the system of equations that in turn define the transition relation by the **next** expression, **INIT** is an expression that defines the initial states, and **CTLSPEC** and **LTLSPEC** are temporal logic specifications. (b) The resulting transition system with the initial states in blue.

the asynchronous update, the non-determinism. Furthermore, we would like to include information about the tendency and stability, i.e., atomic propositions of the form $\delta_v \diamond k$ and $\Delta \diamond k$, in the system description so that temporal specifications may refer to them.

Let us begin with the latter. The NuSMV language permits the use of auxiliary variables that do not belong to the state space and therefore do not induce transitions. The sole purpose of those variables is to make the system description more concise, for example by replacing an often recurring expression by such a variable. They are defined within their own declaration block which is identified by the **DEFINE** keyword. An example is given in Fig. 2.8.

The encoding of the tendency propositions $\delta_i \diamond k$ uses the if-then-else construct `cond ? expr1 : expr2` which evaluates `expr1` if the Boolean condition `cond` is true and `expr2` otherwise. Since $tend_v(x) \in \{-1, 0, 1\}$ for all $v \in V$ and $x \in S$ we use two if statements to decide whether the difference is greater, less or equal to 0. The encoding of the stability propositions $\Delta \diamond k$ uses the count operator `count(list)` which counts the number of expression that are true in `list`.

With the tendency variables, the synchronous transition relation of a network is readily defined by using the **ASSIGN** construction of Fig. 2.9 (a). The asynchronous transition relation requires NuSMV's capability of non-determinism. That is, instead of single solutions only, NuSMV permits a set of successor values for each state variable, and every possible assignment of state variables to values results in a corresponding transition. For the asynchronous transition relation we exploit this by giving each variable the choice to remain at the current value or to change towards its target state. But, the resulting transi-

```

DEFINE
  f1      := 1-v2;
  f2      := v1;
  diff1   := f1-v1;
  diff2   := f2-v2;
  delta1  := dif1>0?1: dif1<0?-1:0;
  delta2  := dif2>0?1: dif2<0?-1:0;
  Delta   := count( (delta1!=0), (delta2!=0) );

```

Figure 2.8: We introduce auxillary variables for the target activities $f_i(x)$ by **f1** and **f2**, for the difference between the target state and the current state $f_i(x) - x(v_i)$ by **diff1** and **diff2**, for the tendencies δ_i by **delta1** and **delta2** and for the number of unstable components Δ by **Delta**. The variable for δ_i is defined by a if-then-else construct and the one for Δ by a count expression, see main text.

tion system would have a self-loop on every state because in every state the case that no variable changes is not yet forbidden. We therefore introduce a so-called *transition constraint*. A transition constraint is a Boolean expression that may involve the current state and next state values. When the transition system is built, all transitions for which the constraint evaluates to *false* are removed. An example of the construction of an asynchronous transition relation is given in Fig. 2.9 (b) and for the mixed update, which is very similar, in Fig. 2.9 (c).

```

ASSIGN
  next(v1) := v1+delta1;
  next(v2) := v2+delta2;

```

(a)

```

ASSIGN
  next(v1) := {v1,v1+delta1};
  next(v2) := {v2,v2+delta2};
TRANS
  Delta=0 | count( (v1!=next(v1)), (v2!=next(v2)) )=1;

```

(b)

```

TRANS
  Delta=0 | count( (v1!=next(v1)), (v2!=next(v2)) )>=1;

```

(c)

Figure 2.9: The constructions of the deterministic, synchronous transition relation in (a) and the non-deterministic, asynchronous transition relation in (b) and the mixed transition relation, which is identical to the asynchronous one except for the transition constraint, in (c).

Benchmark Setup

We close the section on NUSMV with a benchmark for model checking logical networks. The question we attempt to answer is

How large can a logical network be before basic biologically meaningful queries become computationally infeasible?”

To our knowledge this question has not been addressed before.

The question has three aspects that need to be discussed: (1) What type of networks do we consider?, (2) What is a basic query?, and (3) What is meant by computationally infeasible?

We decided to consider randomly generated Boolean networks because they are the simplest networks and because published network generators are available. In the most basic setup, first studied in [12] and also called random N, K networks, a random network for the parameters $(n, k) \in \mathbb{N} \times \mathbb{N}$ is generated by creating n components and selecting for each $v \in V$ exactly k predecessors at random from V . The predecessors are not required to be observable regulators, that is, the update function for each $v \in V$ is chosen at random from all 2^{2^k} possibilities including $f = 0$ and $f = 1$. Intuitively, the number of observable interactions of (V, F) has a considerable effect on the complexity of the state transition graph and hence on whether a query is computationally feasible or not. Hence, we consider a slight variation of the original setup in which the predecessors are required to be observable regulators.

For the benchmark to be reproducible, we used the R package `BOOLNET` [44] and its function `generateRandomNKNetwork` to generate the networks. In addition to the fixed in-degree topology, as defined above, we tested the homogeneous in-degree topology in which the in-degree of each component is chosen according to a Poisson distribution with mean $\lambda = k$. For both topologies we tested the values $k = 2$ and $k = 3$. The function `generateRandomNKNetwork` has more parameters but we have kept them at their default values, see [44] for the details. In particular, as mentioned above, the default parameter for choosing the update functions is `noIrrelevantGenes=True` which requires that the predecessors of each $v \in V$ are observable regulators.

Whether a query is feasible for a network (V, F) also depends on the update strategy. Intuitively, the more transitions there are the more paths there are and the more difficult it is to decide whether a query holds or not. We decided to test all three update strategies of Sec. 2.3, the synchronous, asynchronous and the mixed update.

The second aspect concerns the temporal logic specifications that are tested for each generated network. It seems that there are in principle two approaches. Either the queries are also randomly generated or the same, fixed set of queries is tested each time. Since random queries introduce additional difficulties regarding the parameters of their generation we opted for a fixed set of queries. Our query patterns range from ones that are trivial in the sense that we can a priori say that they are true, to basic reachability, stability and consequence specifications. They are formulated as CTL queries¹ because of the symbolic verification algorithms, see [16] for details. They are summarized in Fig. 2.10 and inspired by the biologically meaningful query patterns of [18].

¹In hindsight we should have repeated them for LTL specifications.

Name	CTL formula
<i>Trivial</i>	$v_1 = 0 \vee v_1 = 1$
<i>Reachability</i>	$\mathbf{EF} (v_1 = 1)$
<i>Stability</i>	$\mathbf{EF} (\mathbf{AG} (v_1 = 1))$
<i>Fixpoint</i>	$\Delta = 0$
<i>Consequence</i>	$\mathbf{AG} (v_1 = 1 \rightarrow \mathbf{EF} (v_2 = 1))$
<i>Sequence 1</i>	$\mathbf{EF} (v_1 = 1 \wedge \mathbf{EF} (v_2 = 1))$
<i>Sequence 2</i>	$\mathbf{E} (v_1 = 1 \mathbf{U} v_2 = 1)$

Figure 2.10: The 7 basic queries that we tested for each random network. *Trivial* is a priori true because each component is Boolean. *Reachability* queries if there is a path along which v_1 is activated. *Stability* tests if finally v_1 is always active. *Fixpoint* tests, if used with the existential satisfaction relation, if there is a steady state. *Consequence* and *Sequence* test nested reachability properties.

Hence, our benchmark consists of 2 topologies, 7 CTL formulae, 3 transition relations and 2 in-degree parameters k . There are therefore $2 \cdot 7 \cdot 3 \cdot 2 = 84$ different scenarios. For each scenario we increased n , starting from $n = 20$, until 3 randomly generated networks with the same parameters failed to compute with NuSMV. We say that a computation fails if the process runs out of memory or if the answer is not computed within 1 hour.

The running time of NuSMV also depends on the enabled features and options it is called with. We tested two setups. The first one has the single option `-dcx` which disables the generation of so-called counterexamples, a useful analysis step that we are not interested in in this basic benchmark. The second one has in addition the options `-df` which enables the computation of the reachable states and `-dynamic` which enables the dynamic variable reordering for the binary decision diagrams (BDD), see [77] for details. It turned out that the second setup outperformed the first and only the results of the second are shown. The model checks were ran as single threaded processes on Linux desktop PCs with 30 GB RAM and 8 CPUs² with 3.00GHz and NuSMV 2.5.4. The timing was done with the Linux commands `time` and `timeout`.

Benchmark Results

The central result of the benchmark is that, averaged across all non-trivial queries and topologies, NuSMV can handle networks with $n \approx 41$ to 71 for the synchronous update and $n \approx 39$ to 55 for the mixed and asynchronous updates. As expected, the non-deterministic transition relations are harder to deal with than the synchronous update. Of course larger networks might be manageable if more memory and faster processors were used, but the order of magnitude for our encodings seems to be less than 100, unless the in-degrees are fixed to $k = 1$ or $k = 2$.

Finally, testing the trivial query shows that NuSMV does not use any form of pre-processing for the temporal specification. For, although the model checker deals well with the trivial query if the update is synchronous, it requires about the same time that is required for more complex queries if the update is non-deterministic. Hence, if we are interested in whether a transition system with

²Intel(R) Xeon(R) CPU X5450 @ 3.00GHz.

	fixed				homogeneous					fixed				homogeneous					fixed				homogeneous			
	K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		
sync.	99	99	99	99		78	45	80	44		68	42	80	42		52	42	47	37		52	40	43	37		
async.	54	36	57	39		55	37	51	39		52	42	47	37		52	42	47	37		52	40	43	37		
mixed	54	36	57	39		55	37	51	39		52	40	43	37		52	40	43	37		52	40	43	37		
(Trivial)					(Reachability)					(Stability)																

	fixed				homogeneous					fixed				homogeneous					fixed				homogeneous			
	K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		
sync.	56	38	53	34		78	41	63	41		70	42	84	36		57	39	54	39		57	39	54	39		
async.	54	38	47	39		55	37	48	38		57	39	54	39		57	39	54	39		57	39	54	39		
mixed	54	38	47	39		55	37	48	38		57	39	54	39		57	39	54	39		57	39	54	39		
(Fixpoint)					(Consequence)					(Sequence 1)																

	fixed				homogeneous					fixed				homogeneous					fixed				homogeneous			
	K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		K=2	K=3	K=2	K=3		
sync.	99	58	85	53		74	44	73	41																	
async.	55	41	49	41		55	39	49	39																	
mixed	55	41	49	41		55	39	48	39																	
(Sequence 2)					(Averages)																					

Figure 2.11: The maximal $n = |V|$ for which each of the 7 queries are still computationally feasible and an 8th table which shows the averages across the 6 non-trivial queries.

$n \approx 100$ satisfies a specification it may be worth it to ask if, with the help of, for example, projections or reductions, we can reduce the problem to smaller sub-problems. An example for this approach is the case-by-case analysis of Chap. 4.8.

A surprising result is that if $k = 3$ then the maximal $n = |V|$ does, apparently, not depend on whether the in-degrees are fixed or chosen at random. Our expectation was that the maximal n should be larger for fixed in-degrees.

2.5 Case Study: The Galactose Switch in Yeast

Yeasts produce energy, in the form of ATP, by breaking down sugars. The preferred nutrient of *S. cerevisiae*, for example, is glucose, but it can also utilize galactose if glucose is not available. The galactose switch is responsible for regulating the transcription of the genes which encode enzymes that are required for the galactose metabolism, see for example [80].

Motivated by the need for in vivo assessments of reverse-engineering and modeling approaches, the authors of [81] have created and inserted

”a five component regulatory network into *S. cerevisiae* that is negligibly affected by endogenous genes and that responds to galactose with gene transcription”.

The network assumed to be autonomous and can be switched on or off by growing the modified yeast in either galactose or glucose. The network is called IRMA which stands for *In vivo Reverse-engineering and Modeling Assessment*. The dynamics of IRMA was studied by perturbation experiments in which the

cells were shifted from glucose to galactose, called *switch-on*, and from galactose to glucose, called *switch-off*. Samples were collected every 10-20 minutes for 3-5 hours and altogether 9 experiments, 5 switch-on and 4 switch-off, were performed. The samples were amplified using a form of *polymerase chain reaction* (PCR) that detects mRNA as opposed to the actual proteins.

As in [81, 82] we assume that an increase or decrease in mRNA leads to a proportional change in protein concentration, although the mRNA might be over-expressed while nothing changes at the corresponding protein level.

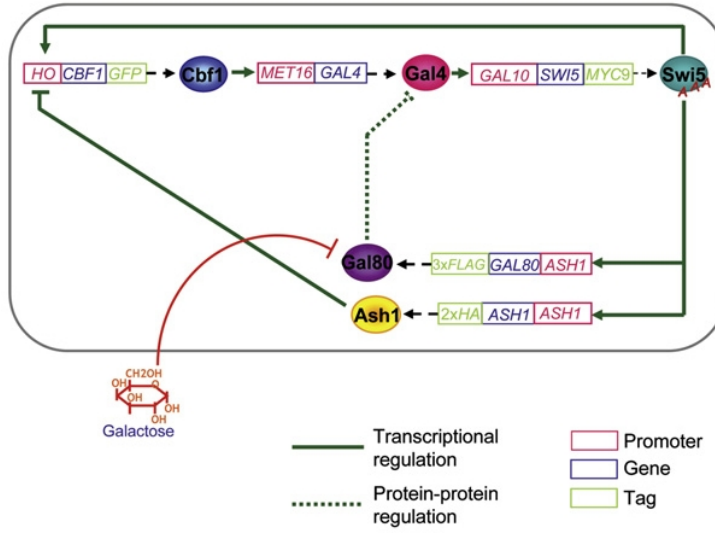


Figure 2.12: The IRMA network, illustration taken from [81]. It consists of 5 proteins *Cbf1*, *Gal4*, *Gal80*, *Ash1*, *Swi5* and the sugar *galactose*.

The aim of this section is to discuss the construction of alternative Boolean models that represent the IRMA network of Fig. 2.12. The construction is based on the information given in [81, 82]. The network consists of five internal components and an external signaling molecule, *gal*, that models whether galactose is available as a nutrient. The internal components, *Cbf1*, *Gal4*, *Gal80*, *Ash1* and *Swi5*, represent the promoters as well as the respective proteins. It is assumed that the components regulate each other's activities via only 8 interactions. Most of them are transcriptional but there is also a protein-protein interaction $Gal80 \rightarrow Gal4$ and a signaling interaction $gal \rightarrow Gal80$.

In [81] it is explained that the transcriptional regulations of the internal components are well-understood: *Cbf1* activates *Gal4* which activates *Swi5* which activates both *Gal80* and *Ash1*. The only complex promoter in the network is the one of *Cbf1*. It is activated by *Swi5* and inhibited by *Ash1*. From the equations given in [81, 82] it seems that it is neither clear whether *Swi5* is necessary for the activation of *Cbf1* nor whether *Ash1* is capable of fully inhibiting *Cbf1*. Hence, three different update functions that agree with the interaction graph of Fig. 2.12 are plausible for *Cbf1*.

$$\begin{aligned}
f_{Cbf1} &= Swi5 \cdot \overline{Ash1} \\
f_{Cbf1} &= Swi5 + \overline{Ash1} \\
f_{Cbf1} &= Swi5
\end{aligned} \tag{2.4}$$

We refer to them as the conjunctive update (*conj*), the disjunctive update (*disj*) and the *Swi5* update (*swi5*), respectively.

The signaling molecule *gal* is modeled as a stable input (see Sec. 2.2.1) with $f_{gal} = gal$. The interaction $Gal80 \rightarrow Gal4$ represents the formation of the protein complex $Gal80 - Gal4$ that inhibits the binding of $Gal4$ to the promoter of *Swi5*. Although *Gal80* and *Cbf1* both interact with *Gal4*, the interactions represent different processes. The latter changes the transcription rate of *Gal4* while the former regulates the concentration of unbound *Gal4*. We must therefore be clear about the semantics of the components activities. Different viewpoints can be taken depending on whether the activity of *Gal4* represents the activity of the respective gene, i.e., its transcription and translation, or whether it represents the concentration of *Gal4* protein that is not bound in a complex with *Gal80*. Each viewpoint entails its own update functions for *Gal4*.

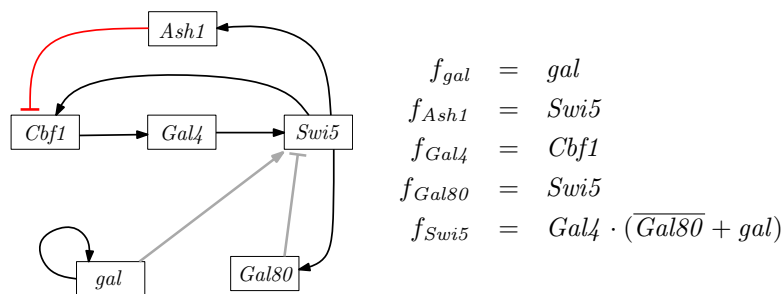


Figure 2.13: The transcriptional viewpoint where the interactions between *gal*, *Gal4* and *Gal80* are modeled by their effect on the promoter of *Swi5*.

In the first case, which we call the **transcriptional viewpoint**, processes like signal transduction and complex formation are implicit in the update function of the promoters of the affected genes. We replace the interactions $gal \rightarrow Gal80$ and $Gal80 \rightarrow Gal4$ by $gal \rightarrow Swi5$ and $Gal80 \rightarrow Swi5$. *Swi5* is expressed if its transcription factor *Gal4* is expressed and not bound to *Gal80*, which is the case if either *Gal80* is not expressed or the signaling molecule *gal* is present. The equations for *Gal4* and *Gal80* model the fact that each has a single transcription factor. The transcriptional viewpoint can also be seen as using a "separation of time scales" argument where it is assumed that the formation of the complex takes place in a negligible amount of time compared with the time required for a change in gene activity. The interaction graph and equations are shown in Fig. 2.13.

In the second case, which we call the **mechanistic viewpoint**, we change the semantics of the components that represent *Gal4* and *Gal80* slightly. Instead of "active" meaning "the gene is expressed" we let activity represent the "presence of unbound protein". Free *Gal4* is present if its transcription factor *Cbf1* is present and free *Gal80* is not present. *Gal80* can bind to *Gal4* if its transcription factor *Swi5* is present and the signal *gal* is absent. This viewpoint

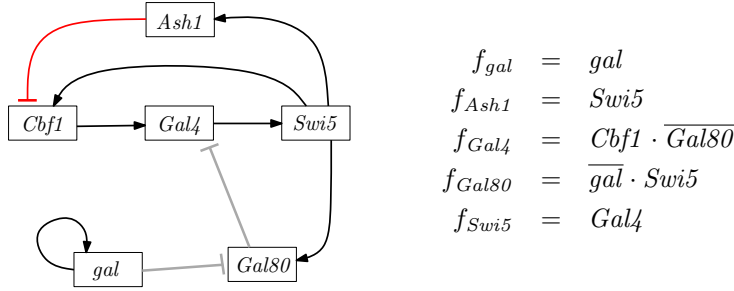


Figure 2.14: The mechanistic viewpoint where each interaction is a physical interaction and $Gal4$ represents the concentration of $Gal4$ proteins that are not bound to $Gal80$.

seems natural because every interaction models a physical binding of proteins. The equations are given in Fig. 2.14.

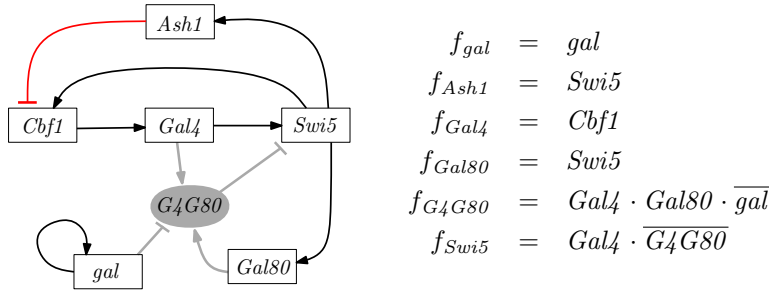


Figure 2.15: The explicit viewpoint introduces an additional component $G4G80$ that models whether the protein complex $Gal4$ - $Gal80$ is present.

The third case is called the **explicit viewpoint** in which the complex $Gal4$ - $Gal80$ is modeled by introducing an additional component $G4G80$. This is therefore the only of the three models that requires a transition for the decay of the $Gal4$ - $Gal80$. The explicit viewpoint can also be seen as introducing a component, namely $G4G80$, that memorizes the conditions required for the formation of the complex. $G4G80$ may therefore be still active while in a state that does not satisfy the conditions anymore. The equations are given in Fig. 2.15.

We are not aware of any formal result that relates models to each other that differ in the above sense, but are otherwise derived from the same information. The underlying connection between them appears to be related to model reduction techniques where some selected components are made implicit, see e.g. [30].

The assumption that the components are Boolean instead of multi-valued is sensible for the transcriptional and mechanistic models because the only component with more than one target is $Swi5$, but two of them $Gal80$ and $Ash1$ have the same promoter, see Fig. 2.12. Their thresholds in terms of $Swi5$ protein concentration can therefore be argued to be comparable. The interaction $Swi5 \rightarrow Cbf1$, however, introduces a second promoter and the thresholds might be different. The explicit viewpoint offers two additional choices for non-Boolean

components, namely *G4G80* and *Gal4*.

Since the three viewpoints, the three update functions for *Cbf1* (Eq. 2.4) and the three transition relations (Sec. 2.3.1) already define 27 different models and since in Sec. 3.7 we want to test all of them for compatibility with expression data (also available in [81]) we decided not to explore non-Boolean models in this thesis.

Chapter 3

Reachability Queries

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

– G. Batt et al. in [83]

A mathematical model is validated by testing it against experimental observations. If the observation belongs to the model’s behaviors then the data and model are said to be compatible. Otherwise, one usually seeks a revision that introduces a minimal number of changes to the original model and ensures that the data and the new, revised model are compatible.

The central motivation in this chapter is the first step: deciding whether a model and a particular type of observation, a time series, are compatible. Sec. 3.1 discusses time series specific issues like *data discretization* and the *sampling rate*. Sec. 3.2 introduces a path-based notion of compatibility for time series and models and suggests a temporal logic query for deciding it. Sec. 3.3 extends the previous notion by introducing the concept of monotonic paths. Sec. 3.4 applies the notion of monotony to assess the sampling rate. Sec. 3.5 discusses additional variations of compatibility. Sec. 3.6 introduces a Python package, `TEMPORALLOGICTIMESERIES`, that translates discrete time series into CTL and LTL formulae. Sec. 3.7 illustrates the ideas on a biological case study and Sec. 3.8 discusses the results. This chapter is a complete revision and extension of results published in [84–86].

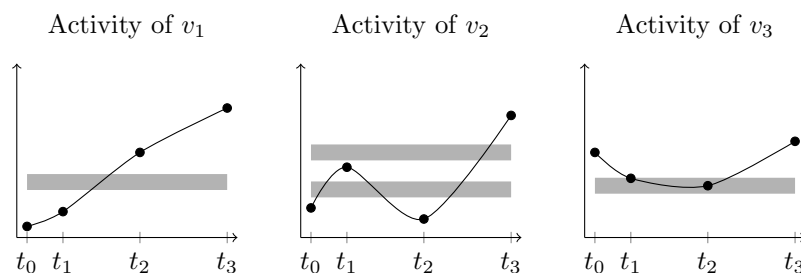
3.1 Introduction: Time Series Data

We collectively refer to the data obtained from recording the activities of the components of a model as *time series data*. Because of the high level of abstraction inherent to the qualitative modeling framework the various different types of time series data, i.e., gene expression profiles, protein or RNA concentration data, time-lapse microscopy, fluorescent reporters etc., are potentially all amenable to the subsequent discussion.

Apart from recording activities, time series data is also characterized by the *sampling rate*, i.e., the amount of time that elapsed in between measurements.

The sampling rate may in general be non-uniform with, for example, a high frequency in the beginning of an experiment and a lower frequency towards the end. An important computational problem for designing time series experiments is to determine the lowest sampling rate that is likely to capture the key events of the experiment. See [87,88] for a review on related questions and Sec. 3.4 for our contributions to the problem.

A discrete time series is obtained from continuous-valued data by a suitable discretization method, directly by qualitative observations, or a mixture of both. Including qualitative observations in the time series is a strength of discrete modeling as it may be hard to translate such assumptions into quantitative data required for continuous models.



(a) Time series data

$p_0 = 0_1 0_2 1_3$
 $p_1 = 0_1 1_2$
 $p_2 = 1_1 0_2$
 $p_3 = 1_1 2_2 1_3$

	v_1	v_2	v_3	
p_0				0
p_1				1
p_2				2
p_3				—

(b) Symbolic state representation

(c) Tabular representation

Figure 3.1: An example of a real-valued and discretized time series. (a) The numerical data of three components v_1, v_2, v_3 was taken at four time points t_0, \dots, t_3 with non-uniform sampling rate. The plots show possible expression curves that agree with the measured data points. Measurements inside the gray horizontal threshold areas are interpreted as uncertain. (b) The discrete time series that corresponds to the data and thresholds. (c) The time series in tabular form with the color code on the right, red indicating "uncertain value".

The issue of choosing a suitable discretization method for continuous data is crucial in the sense that the outcome of subsequent analysis steps is clearly dependent on it. Formally, continuous data is a sequence of real-valued vectors, one for each measurement, where each entry represents the activity of a single component at that time point. A *discretization* is then a mapping that is monotone in the values of those vectors onto integer vectors of the same dimension.

An important parameter of discretization is the number of cut-points, i.e., the integer domain that is permitted for each component. Some methods binarize only, others require the integer domain for each component as an input while others determine the domains automatically. For a formal discussion of data discretization in the context of systems biology see, for example, [89].

It is worth noting a couple of things: (1) Given that the data represents concentration levels and under the assumption that the interactions are switch-like, one has to ideally estimate the threshold values of the underlying sigmoidal curves. One may therefore consider to discretize the data by guessing the threshold values. (2) Although exponential, the total number of discretizations of a continuous-valued time series is finite for a fixed number of cut-points. In certain cases, for example if the aim is to binarize a short time series, it may therefore be feasible to consider many or all possible discretizations.

The starting point of this chapter is the following definition.

Definition 16. *A discrete time series for a given set of components V is a sequence $P = (p_0, \dots, p_m)$ of $m \in \mathbb{N}$ symbolic states $p_i \in \text{Sym}(V)$ where p_i represents the i^{th} measurement of the experiment.*

That is, we explicitly permit questionable or imprecise discretization results in the form of excluding those components from the domain of the respective symbolic state (see Def. 4). In practice, this has the advantage of deriving results based on varying levels of certainty.

The notion we aim to discuss and define is the compatibility of (V, F) with a time series $P = (p_0, \dots, p_m)$. In particular, we investigate additional monotony and stability assumptions and turn the question of compatibility into a formal decision problem that can be answered with model checking.

3.2 Compatibility

The issue of whether some time series data and a qualitative model are compatible has implicitly been addressed in the context of reverse engineering. Naturally, it is based on the STG of the model and therefore dependent on the particular transition rule one has chosen. To our knowledge compatibility has so far only been considered for the synchronous update \rightarrow in which case it is based on the existence of a transition between successive measurements, as in [41, 90–92].

In such settings the authors usually assume that every component of the model can be measured or guessed at every time point, i.e., that the data can be discretized into a sequence of states (x_0, \dots, x_m) , with $x_i \in S$, rather than symbolic states. The model and time series are then said to be compatible if for all $0 \leq i < m : x_i \rightarrow x_{i+1}$.

This notion has the advantage that the reverse engineering of a model from data becomes feasible for relatively large networks, see for example [92], but creates some complications regarding the discretization step: only time series that satisfy the consistency condition

$$\forall 0 \leq i < j < m : x_i = x_j \Rightarrow x_{i+1} = x_{j+1}$$

can potentially be compatible with a model, the others can not because \rightarrow is deterministic. Otherwise, in case there are inconsistencies, it appears that the best one can do is to simply discard them from the reverse engineering process, as discussed in [93].

But, even if the time series is consistent, a transition-based notion of compatibility seems inappropriate for quasi-continuous (see Sec. 2.3.1) and for

asynchronous STGs as we would have to additionally require that $\forall v \in V : \text{dist}_v(x_j, x_{j+1}) \leq 1$ in the former and $\text{Dist}(x_j, x_{j+1}) = 1$ in the latter case, for all measurements $0 \leq j < m$. Hence, we introduced in [85] the following path-based notion of compatibility.

Definition 17. A model (V, F) and its STG (S, \rightarrow) are compatible with a time series $P = (p_0, \dots, p_m)$ iff there is a sequence of states (x_0, \dots, x_m) with $x_i \in S[p_i]$ such that for every $0 \leq i < m$ there is a path $x_i \rightsquigarrow x_{i+1}$ in (S, \rightarrow) . We call $x_0 \rightsquigarrow x_1 \rightsquigarrow \dots \rightsquigarrow x_m$ a (P) -time series path and say that x_i represents the i^{th} measurement of P .

Note that since non-deterministic STGs are usually over-approximations of the modeled system, we assume that the presence of a single time series path is sufficient for the compatibility of model and data. Alternatively, one could consider requiring that every path that starts in $S[p_0]$ is a time series path.

Fig. 3.2 illustrates the problem of deciding compatibility.

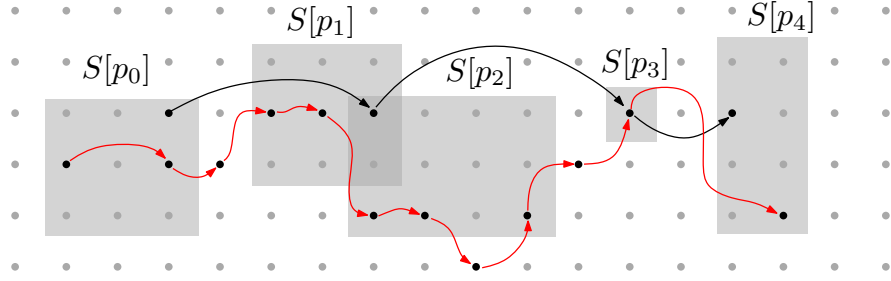


Figure 3.2: A schematic view on the state space of a model. Dots represent states, rectangles the subspaces which are referenced by the symbolic states of the time series. The model is compatible with (p_0, \dots, p_4) if there is a path that visits $S[p_0], \dots, S[p_4]$ in order. Because the sets $S[p_i]$ may contain more than one state and because update rules may be non-deterministic, there may be several time series paths.

We conclude with a couple of remarks. First, deciding whether a model and time series are compatible is in general a hard problem. At the end of Sec. 3.3 we argue that it belongs to the class of NP-hard problems.

Second, if a model and time series are compatible then there may be several different time series paths. The difference between them may, for example, concern the path length as indicated by the red and black paths in Fig. 3.2.

Third, if there is a time point $0 < i < m$ such that $|S[p_i]| = 1$ then a model (V, F) and (p_0, \dots, p_m) are compatible if and only if (V, F) is compatible with (p_0, \dots, p_i) and with (p_i, \dots, p_m) . The original problem therefore decouples into two independent sub-problems. This observation is exploited in [94] where a simple and efficient graph search algorithm is proposed that solves compatibility problems that decouple maximally, i.e., into m sub-problems. On the other hand, even the most favorable assumption, namely that the model satisfies every induced sub-problem

$$\forall 0 \leq i < j \leq m : \exists x_i \in S[p_i], x_j \in S[p_j] : x_i \rightsquigarrow x_j \quad (3.1)$$

is in general only necessary but not sufficient to guarantee that the model is compatible with (p_0, \dots, p_m) .

Fourth, the sets $S[p_i]$ may have a non-empty intersection, as for p_1 and p_2 in Fig. 3.2. The definition of compatibility does not require that the states that represent the measurements are distinct. Hence, a model and time series are trivially compatible if there is a state $x \in S$ with

$$x \in \bigcap_{0 \leq i \leq m} S[p_i].$$

In this case the measurements agree so well that a single state x is capable of representing all measurements and, independent of the update functions F , a model is compatible with the time series iff $x \in S$.

Finally, in case a model and time series are not compatible, it would be desirable to retrieve some information on why that is. Along this line we propose to check the necessary conditions of Eq. 3.1. In some cases, the incompatibility may be explained by the existence of i, j such that $0 \leq i < j \leq m$ and for all $x_i \in S[p_i], x_j \in S[p_j] : x_i \not\sim x_j$. In this case $S[p_j]$ is not reachable from $S[p_i]$, which may have an easy explanation: $S[p_i]$ may, for example, be a so-called *trap set* in which case every i such that $i < j \leq m$ and $S[p_j] \cap S[p_i] = \emptyset$ will cause an inconsistency. The concept of trap sets and how to detect those that are simultaneously subspaces is described in Chap. 4.

3.2.1 Temporal Logic Encoding

It is not difficult to see that the question of compatibility can be solved with model checking algorithms. Recall from Sec. 2.4 that we specify transition systems as 3-tuples $TS = (S, \rightarrow, I)$ with $I \subseteq S$ the initial states. The LTL formula that queries whether the time series (p_0, p_1, p_2) and a model are compatible is defined by nesting **F** operators in the following way:

$$\phi := \mathbf{F} (p_1 \wedge \mathbf{F} (p_2))$$

and checking whether the transition system $TS = (S, \rightarrow, S[p_0])$ satisfies $TS \models \phi$. In general, the *nested reachability query* is given by the following construction.

Definition 18. *The nested reachability query $R(P)$ for a time series $P = (p_0, \dots, p_m)$ is defined recursively by*

$$\begin{aligned} \phi_m &:= p_m \\ \phi_{m-t} &:= p_{m-t} \wedge \mathbf{F} \phi_{m-t+1}, \quad t = 1, \dots, m \end{aligned}$$

and $R(P) := \mathbf{F}(\phi_0)$.

The next statement observes that the nested reachability query does really solve the compatibility problem.

Observation 5. *A model (V, F) and time series $P = (p_0, \dots, p_m)$ are compatible if and only if*

$$TS \models R(p_1, \dots, p_m)$$

where $TS = (S, \rightarrow, S[p_0])$.

Note that by quantifying the path operator \mathbf{F} in Def. 18 we obtain two CTL variants of $R(P)$ which we denote by $R^{\mathbf{E}}(P)$ and $R^{\mathbf{A}}(P)$. It is not hard to see that $R(P)$ and $R^{\mathbf{E}}(P)$ are equivalent in the sense that for any transition system and time series both queries are either simultaneously true or false. Depending on the model checking software available or for reasons of efficiency, state space encoding or counter example generation, one may therefore choose either.

3.3 Monotonic Compatibility

The nested reachability query is a somewhat weak definition of compatibility because it accepts paths of an arbitrary length. There are several arguments for the preference of short time series paths.

The first one involves the assumption that there is a correspondence between a path's length and the energy required to perform its transitions¹. In particular, given a time series and two alternative models, each compatible with the time series, one is then tempted to favor the model whose STG contains the shorter time series path.

Similarly, but avoiding the notion of energy, one might argue that models with shorter time series paths offer a simpler explanation for the data and that path length is therefore an interesting criterion for model selection. There are of course modeling scenarios in which neither of the two arguments is applicable. Intuitively, the longer a time series path the more *oscillations* will necessarily appear in the activity profiles of the components until the trajectory reaches a steady state. This observation is illustrated in Fig. 3.3.

The relevance of oscillations in time series paths is that they implicitly establish a relationship between the sampling rate on the one hand, and the reaction rates of the biological processes, which are represented by the components, on the other. Intuitively, if for a process the time elapsed between successive measurements is small compared to its reaction rate then we expect its activity to change without oscillations. This observation motivates the need to query the existence of paths that are monotonic in a specific component.

Definition 19. *A path $x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_k$ is increasing, decreasing or steady in a component $v \in V$, iff*

$$\forall i < k : x_i(v) \leq x_{i+1}(v) \quad (\text{increasing})$$

$$\forall i < k : x_i(v) \geq x_{i+1}(v) \quad (\text{decreasing})$$

$$\forall i < k : x_i(v) = x_{i+1}(v) \quad (\text{steady})$$

Let $M : U \rightarrow \{\nearrow, \searrow, \bullet\}$ be a function that labels the components $U \subseteq V$ as increasing \nearrow , decreasing \searrow or steady \bullet . A path $x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_k$ is M -monotonic iff it is increasing, decreasing and steady in all $v \in U$ that are labeled respectively. M is called a monotony specification.

Since sampling rates are generally non-uniform and since a model may represent processes whose reaction rates are on different scales of magnitude, for example signal transduction and gene regulation, we would like to specify independently which components are monotonic and in which intervals. The

¹Assuming that each transition requires about the same amount of energy.

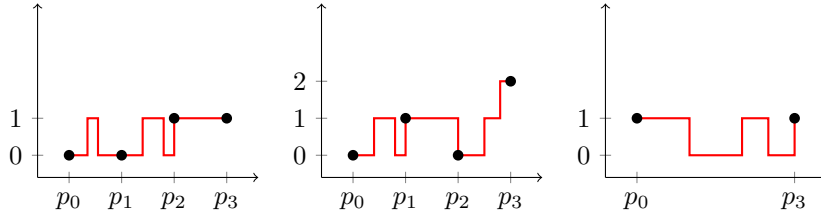
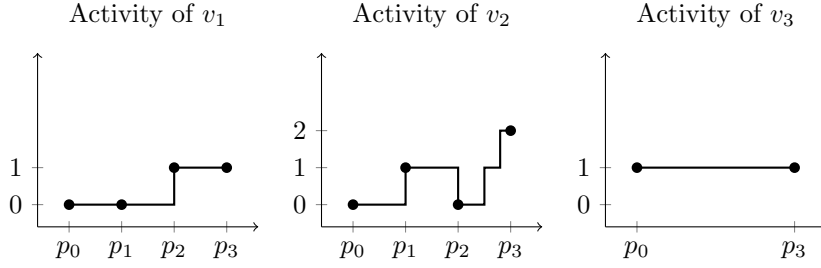


Figure 3.3: As the length of a time series path increases oscillations in between the activity profiles (p_0, \dots, p_3) will necessarily appear. The profiles in (a) have no oscillations while the profiles in (b) have each at least one oscillation.

following definition of *monotonic compatibility* achieves this with a sequence $M = (M_0, \dots, M_{m-1})$ of monotony specifications.

Definition 20. Let $P = (p_0, \dots, p_m)$ be a time series and $M = (M_0, \dots, M_{m-1})$ a sequence of monotony specifications with $U_i \subseteq V$ and

$$M_i : U_i \rightarrow \{\nearrow, \searrow, \bullet\}.$$

The STG (S, \rightarrow) of a model (V, F) is M -compatible with P iff there is a P -time series path $x_0 \rightsquigarrow \dots \rightsquigarrow x_m$ such that for every $0 \leq i < m$ the path segment $x_i \rightsquigarrow x_{i+1}$ is M_i -monotonic.

A time series and monotony specification are conveniently represented in tabular form by superimposing M_i on the respective cells in the table, as in Fig. 3.4. Note also that the monotony specifications are independent of whether the respective component has been measured.

3.3.1 Temporal Logic Encoding

As for the problem of compatibility there is a temporal logic encoding that decides whether a model is M -compatible with a time series P . The structure of the query is similar to that of nested reachability R but involves a sequence of temporal logic conditions $C = (\gamma_1, \dots, \gamma_m)$. With the intention of introducing further variations of the notion of compatibility in Sec. 3.5, we now define a general *conditional nested reachability query pattern* and then show how M -compatibility is decided by one particular choice of C . The idea of "high-level query templates" that capture recurring biological questions and can be

	v_1	v_2	v_3	
p_0		\nearrow		0
p_1	\nearrow	\nwarrow	\searrow	1
p_2	•	\nearrow	•	2
p_3				—

Figure 3.4: An example of a specification of a time series $P = (p_0, \dots, p_3)$ and monotones $M = (M_0, \dots, M_2)$ in tabular form. The values of M_i are superimposed on the table. $M_2(v_3)$ is for example shown in the (p_2, v_3) cell. Note that we can specify a component to be monotonic even if its activity is not measured. For example $M_1(v_3) = \searrow$ but $v_3 \notin V[p_1]$ and $v_3 \notin V[p_2]$.

automatically translated into temporal logic was advocated in [18]. We begin with an LTL encoding.

Definition 21. Let $P = (p_0, \dots, p_m)$ be a time series and $C = (\gamma_0, \dots, \gamma_m)$ any sequence of LTL formulae. The conditional nested reachability query $R(P, C)$ is defined recursively by

$$\begin{aligned} \phi_m &:= p_m \\ \phi_{m-t} &:= p_{m-t} \wedge (\gamma_{m-t+1} \mathbf{U} \phi_{m-t+1}), \quad t = 1, \dots, m \end{aligned}$$

and $R(P, C) := (\gamma_0 \mathbf{U} \phi_0)$.

The nesting of \mathbf{U} operators in this way ensures, for every $i < m$, that when a suitable $x_i \in S[p_i]$ is reached, condition γ_{i+1} is satisfied until the next suitable $x_{i+1} \in S[p_{i+1}]$ is reached and so on. A first observation is that if C is non-restrictive, e.g. if $C = (\text{true}, \dots, \text{true})$, then $R(P)$ and $R(P, C)$ are equivalent. The latter is therefore a generalization of the former.

Two CTL versions of $R(P, C)$ are obtained from Def. 21 by requiring that C is a sequence of CTL formulae and quantifying the operator \mathbf{U} . We denote them by $R^{\mathbf{E}}(P, C)$ and $R^{\mathbf{A}}(P, C)$.

Next, we want to show that for a time series P and monotony specification M , there is a sequence of conditions C such that the model is M -compatible with P if and only if it satisfies $R(P, C)$. The idea is to enforce the monotones by conditional next operators.

Definition 22. Let $M : U \rightarrow \{\nearrow, \searrow, \bullet\}$ with $U \subseteq V$ be a monotony specification. We define the LTL formula $\mu(M)$ by

$$\mu(M) := \bigwedge_{v \in U} \xi(v, M(v)),$$

where $\xi(v, M(v))$ is in turn defined by

$$\begin{aligned}\xi(v, \nearrow) &:= \bigwedge_{k=1}^{Max(v)} (v = k \Rightarrow \mathbf{X} v \geq k) \\ \xi(v, \searrow) &:= \bigwedge_{k=0}^{Max(v-1)} (v = k \Rightarrow \mathbf{X} v \leq k) \\ \xi(v, \bullet) &:= \bigwedge_{k=0}^{Max(v)} (v = k \Rightarrow \mathbf{X} v = k).\end{aligned}$$

In case $U = \emptyset$ then $\mu(M)$ is defined to be $\mu(M) := \text{true}$.

The formulae $\xi(v, \nearrow)$, $\xi(v, \searrow)$, $\xi(v, \bullet)$ are therefore satisfied iff the next transition is increasing, decreasing or steady in v , respectively. By taking the conjunction over $k \in \text{Dom}(v)$ it is ensured that the monotony is independent of the value of v in the current state. Note the slightly different index sets in each case that stem from the observation that $(v = 0 \Rightarrow \mathbf{X} v \geq 0)$ and $(v = k \Rightarrow \mathbf{X} v \leq k)$ for $k = \text{Max}(v)$ are trivially true. With the query pattern $\mu(M)$, we are now ready to define C of the conditional reachability query that decides the question of M -compatibility.

Observation 6. Let $P = (p_0, \dots, p_m)$ be a time series, $M = (M_0, \dots, M_{m-1})$ a sequence of monotony specifications and $TS = (S, \rightarrow, S[p_0])$. A model (V, F) is M -compatible with P if and only if

$$TS \models^{\exists} R(P', C)$$

where $P' := (p_1, \dots, p_m)$ and $C = (\mu(M_0), \dots, \mu(M_{m-1}))$ where $\mu(M_i)$ are the query patterns of Def. 22.

Note that in this construction the until operators \mathbf{U} of the pattern $\mu(M_i)$ ensure that $\xi(v, M_i(v))$ is satisfied not only for a single transition but for the whole path segment $x_i \rightsquigarrow x_{i+1}$ that connects p_i to p_{i+1} .

As for compatibility, it is natural to ask whether monotonic compatibility may be encoded in CTL. Unfortunately, this is problematic. To explain why, we briefly discuss the notion of *equivalence* for LTL and CTL formulae. A LTL formula φ and CTL formula Φ are said to be *equivalent*, denoted by $\varphi \equiv \Phi$, if for every transition system TS

$$TS \models \varphi \Leftrightarrow TS \models \Phi.$$

In [16], a theorem is presented that helps in deciding for a given formula in one language whether there exists an equivalent one in the other and what it might be.





Theorem (Clarke and Draghicescu, from [16] Sec. 6.3)). *Let Φ be a CTL formula, and φ the LTL formula that is obtained by eliminating all path quantifiers in Φ . Then:*

$\Phi \equiv \varphi$ or there does not exist any LTL formula that is equivalent to Φ .

Hence, if an LTL formula and a CTL formula are equivalent then the latter is obtained from the former by quantifying the path operators. An example is the equivalence $R(P) \equiv R^E(P)$ for compatibility which was mentioned in Sec. 3.2.1. To prove that monotonic compatibility can not be encoded in CTL it is therefore sufficient to find for every quantification ψ of the LTL encoding ϕ of a particular time series and monotony specification, a transition system that satisfies exactly one of ψ and ϕ .

Theorem 1. *Monotonic compatibility can not be encoded in CTL.*

Proof. Consider the following time series $P = (p_0, p_1)$ and monotony specification of a Boolean network with $V = \{v_1, v_2, v_3\}$.

	v_1	v_2	v_3	
p_0				0
p_1				1

The LTL formula that encodes the above M -compatibility problem is

$$\phi := (v_1 = 1 \Rightarrow \mathbf{X} v_1 = 1) \mathbf{U} p_1.$$

Since there are 2 operators, there are 2^2 different CTL formulae obtained from ϕ by quantifying the operators \mathbf{X} and \mathbf{U} .

$$\begin{aligned} \Phi_{EE} &:= \mathbf{E} (v_1 = 1 \Rightarrow \mathbf{E}\mathbf{X} v_1 = 1) \mathbf{U} v_1 = 1 \\ \Phi_{EA} &:= \mathbf{E} (v_1 = 1 \Rightarrow \mathbf{A}\mathbf{X} v_1 = 1) \mathbf{U} v_1 = 1 \\ \Phi_{AE} &:= \mathbf{A} (v_1 = 1 \Rightarrow \mathbf{E}\mathbf{X} v_1 = 1) \mathbf{U} v_1 = 1 \\ \Phi_{AA} &:= \mathbf{A} (v_1 = 1 \Rightarrow \mathbf{A}\mathbf{X} v_1 = 1) \mathbf{U} v_1 = 1 \end{aligned}$$

Consider the transition system TS_1 below which is the asynchronous STG of some Boolean network. It is not M -compatible because the path from 000 to 111 contains a decrease in v_1 . But, the initial state 000 satisfies the two CTL candidates Φ_{EE} and Φ_{AE} . This can be seen by checking that $(v_1 = 1 \Rightarrow \mathbf{E}\mathbf{X} v_1 = 1)$ is satisfied in every state along the black path. In particular, $101 \models (v_1 = 1 \Rightarrow \mathbf{E}\mathbf{X} v_1 = 1)$ because of the transition $101 \hookrightarrow 100$ in red.

The other two candidates are ruled out by TS_2 which is the asynchronous STG of another network. This system is M -compatible because the black path is non-decreasing in v_1 , but the initial state does not satisfy the other two CTL formulae. The reason is that $101 \not\models (v_1 = 1 \Rightarrow \mathbf{A}\mathbf{X} v_1 = 1)$ because of the red transition that is decreasing in v_1 . \square

We conclude with a couple of remarks. First, Thm. 3.3.1 of Clarke and Draghicescu only applies when equivalence is considered over the set of all transition systems. For infinite subsets of transition systems, for example the ones that are the STGs of logical networks, it may not be true that one can check the existence of an equivalent CTL encoding by quantifying the path operators. Hence, Thm. 1 does not prove that there is no CTL equivalent for M -compatibility for logical networks. We believe, however, that Thm. 3.3.1 also holds for the restriction to transition systems of logical networks and in particular Boolean networks.

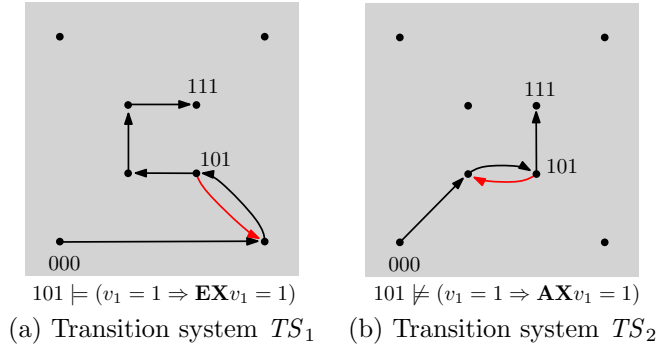


Figure 3.5: Two transitions systems that prove that monotonic compatibility can not be encoded by quantifying the LTL property for monotony.

Second, the statement "there is no equivalent CTL encoding" becomes meaningless when considered over a finite set of transition systems, as is the case in real life modeling scenarios, because then, for any LTL formula there is an equivalent CTL formula that is obtained, not by quantification, but instead by some form of explicit enumeration of all cases of transition systems for which the formula holds. Of course, such enumeration will result in overly long and complicated CTL formulae and is never actually useful.

Third, the definition of monotonic paths and the corresponding temporal logic encoding are independent of the type of update rule considered. In particular they work for the three updates strategies $\hookrightarrow, \Rightarrow, \succrightarrow$ of Sec. 2.3 as well as for the update by so-called *priority classes* which are defined in [95].

Finally, certain combinations of values of $Map_{p_i}(v)$ and $Map_{p_{i+1}}(v)$ entail a necessary condition for the value of $M_i(v)$ with regards to the existence of a monotonic path. For example, if $Map_{p_i}(v) < Map_{p_{i+1}}(v)$ then $M_i(v) \in \{\searrow, \bullet\}$ is clearly not satisfiable by any path.

3.3.2 Computational Complexity

The aim of this section is to answer the question

How hard is it to decide if a logical network and a time series are compatible?

Intuitively speaking, decision problems are problems that can be rephrased in terms of the existence of an object that lives in a space that is determined by the *problem input*. A *guess* to a decision problem is an element in this space. In our case the problem input is a logical network together with a time series. The question "are they compatible?" can be rephrased as "is there a time series path in the transition system?". A naive solution to decision problems is to test each guess, in our case to enumerate all paths and test each for whether it reproduces the time series or not.

A decision problem belongs to the class of *non-deterministic polynomial time* (NP) problems if guesses can be tested in polynomial time with respect to the size of the input. A *reduction* of one decision problem into another is a function that transforms instances of the first into instance of the latter in polynomial

time. A problem is *NP-complete* if it can be reduced to the *Boolean satisfiability problem* (SAT), which is defined in the main part of the section below, and vice versa. A problem is *NP-hard* if it is not in NP but there is a NP-complete problem that can be reduced to it. NP-hard problems are therefore at least as hard as NP-complete problems. For a formal discussion of computational complexity and reductions see [96].

Formally, the introductory question is the decision problem with the input being a logical network (V, F) and a time series $P = (p_0, \dots, p_m)$ and the output being whether there is a P -time series path in (S, \rightarrow) . We call it *Compatibility*, denoted by COMP. To determine its computational complexity we will focus on a special case of logical network and time series. Clearly, COMP is then at least as hard as the special case. The decision problem we focus on is called *Boolean Compatibility* (BCOMP).

Name	BCOMP
Input	A Boolean network (V, F) and a discrete time series $P = (p_0, p_1)$.
Output	Whether (S, \hookrightarrow) of (V, F) is compatible with P .

Note that we are only considering the asynchronous transition graph (S, \hookrightarrow) and time series with exactly two measurements. We will prove the BCOMP is NP-hard. The proof is the result of a discussion with A. Reimers² and based on a reduction of SAT to BCOMP.

First, it seems that BCOMP does not belong to NP because the length of a guess, in this case a path, is not polynomially bounded by the length of the input $((V, F), P)$. The reason is that paths are sequences in S that may grow exponentially with respect to (V, F) . Hence, even reading a guess might not be achievable in polynomial time, let alone checking if it is a time series path. To rigorously prove that BCOMP does not belong to NP would require us to construct a family $((V_i, F_i), P_i)$ of instances of BCOMP such that each (V_i, F_i) is compatible with P_i but the shortest time series paths do actually grow exponentially in the size of the input. We would also have to show that reading an exponential length fragment of the shortest path is necessary to decide whether it is a time series path.

We now sketch a polynomial time reduction of SAT to BCOMP.

The input of an instance of SAT is a set of Boolean variables $U = \{u_1, \dots, u_n\}$ and clauses $C = \{c_1, \dots, c_k\}$ where each c_i is a logical disjunction of some variables of U or their negation. A solution to SAT is an assignment of truth values to U that satisfies all clauses in C . We transform an instance of SAT into a Boolean network (V, F) with $n+1$ components, one for each variable $u \in U$ and an additional one for the clauses C , and a time series $P = (x_0, x_1)$. First the definitions of (V, F) . For ease of notation, define $V := U \cup \{u_{n+1}\}$ where u_{n+1} is the extra variable. We define the update functions $F = \{f_1, \dots, f_{n+1}\}$ by

$$f_i := \begin{cases} 1 & : 1 \leq i \leq n \\ c_1 \cdot \dots \cdot c_k & : \text{else} \end{cases}.$$

Hence, (V, F) is a network in which every component $u_i \in U$ can always transition from 0 to 1, but the component u_{n+1} can only change if the state of

²2013, Freie Universität Berlin

U satisfies all clauses C , see Fig. 3.6. The measurements of the time series $P = (x_0, x_1)$ are the 0 state and the 1 state $x_0, x_1 \in S$ that satisfy

$$\forall v \in V : x_1(v) = 1, x_0(v) = 0$$

respectively.

The reduction of SAT to BCOMP is correct:

Proposition 1. *A SAT instance (U, C) has a solution if and only if the corresponding (V, F) is compatible with the time series $P = (x_0, x_1)$ in (S, \hookrightarrow) .*

Proof. The proof is based on the observation that a solution of (U, C) , say $r : U \rightarrow \mathbb{B}$, is represented by a state $s \in S[u_{n+1} = 0]$ with $s(u) = r(u)$ for all $u \in U$. s is reachable from x_0 because every state in $S[u_{n+1} = 0]$ is reachable from x_0 . But $f_{n+1}(s) = 1$ because s satisfies all clauses and hence there is a transition $s \hookrightarrow s' \in S[u_{n+1} = 1]$. Since for any $y \in S[u_{n+1} = 1]$ there is a path to x_1 we have found a P time series path. The same argument holds the other way, if there is a path $x_0 \rightsquigarrow x_1$ then this path can be split in $x_0 \rightsquigarrow s \hookrightarrow s' \rightsquigarrow x_1$ where $s \in S[u_{n+1} = 0]$ and s represents a solution to (U, C) since $f_{n+1}(s) = 1$. An illustration is given in Fig. 3.6. \square

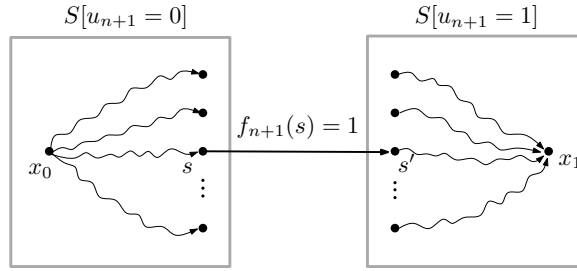


Figure 3.6: The transition graph (S, \hookrightarrow) of the logical network (V, F) that corresponds to a SAT instance (U, C) . The clauses C are encoded in $f_{n+1} \in F$. There is a solution to (U, C) iff there is a path $x_0 \rightsquigarrow s \hookrightarrow s' \rightsquigarrow x_1$. It is represented by $s \in S[u_{n+1} = 0]$ because $f_{n+1}(s) = 1$.

3.4 Assessment of the Sampling Rate

The monotony specification $M = (M_0, \dots, M_{m-1})$ was motivated by the need to decide if there is a time series path that is free of oscillations for a set of variables $U_i \subseteq V$ in between p_i and p_{i+1} and hence in agreement with the sampling rate. So far, it was therefore the data that determined whether a given model is acceptable or not. But the information flow can also be reversed: a compatible model can be used to assess the data.

In [87] it is emphasized that the determination of the lowest sampling rate that captures all key events is an important challenge in designing time series experiments. The argument is that under-sampling might misrepresent the component's activities while over-sampling is expensive and time consuming. Currently, the author remarks, the sampling rates are usually fixed by the intuition of the experimentalists.

In the following we discuss how a model that is M -compatible with a time series $P = (p_0, \dots, p_m)$ can be used to decide if specific intervals of the experiment were under-sampled for specific components. Note that M represents initial assumptions. The assessment includes the case when M is empty in which case the assessment is based on basic compatibility. The idea is to detect under-sampled intervals by checking if the model predicts a behavior that is *unexpected* with respect to the time series and initial assumptions M . Hence, the notion of what is to be expected defines whether an interval is under-sampled or not. Our notion of expectation is based on monotonies and changes in activities.

The procedure is to check for every component $v \in V$ and every sample p_i if v is currently decreasing, increasing or steady. This requires its last and next measured values, to infer the direction. The indices of the last and next measured values are s and t in the definition below. If these values exist then the expected monotony is clear and depends on their relative order. In general, however, it might be the case that there is no last value, or no next value or even both (if v is not measured at all in P). In this case we will make the ad hoc assumption that v was steady. Note that in the assessment procedure described below, this assumption can be overridden by any other monotony specification.

Definition 23. *Given a time series $P = (p_0, \dots, p_m)$ over the components V . Let (i, v) be a tuple with $0 \leq i \leq m - 1$ and $v \in V$. The expected monotony $E(i, v) \in \{\nearrow, \searrow, \bullet\}$ of v in between the measurements p_i and p_{i+1} is defined by the following cases: If*

$$\begin{aligned} s &:= \max\{j \in [0..m] \mid j \leq i, v \in V[p_j]\} \\ t &:= \min\{k \in [0..m] \mid k \geq i + 1, v \in V[p_k]\} \end{aligned}$$

exist then we define

$$E(i, v) := \begin{cases} \nearrow & : \text{Map}_{p_s}(v) < \text{Map}_{p_t}(v) \\ \bullet & : \text{Map}_{p_s}(v) = \text{Map}_{p_t}(v) \\ \searrow & : \text{Map}_{p_s}(v) > \text{Map}_{p_t}(v) \end{cases}$$

Otherwise p_i is the last or first measurement in the profile of v , or v is not measured at all, and we define $E(v, i) := \bullet$.

Recall that $\text{Map}_p : V[p] \rightarrow \text{Dom}(v)$ is the mapping representation of a symbolic state $p \in \text{Sym}$, see Sec. 2.1.1.

Assessment Procedure

Recall that we base our assessment on a model that is assumed to be M -compatible with $P = (p_0, \dots, p_m)$ where $M = (M_0, \dots, M_{m-1})$ represents the initial monotony assumptions. With the expected monotony defined we propose the following assessment procedure.

(1) Mark all positions (i, v) that are not already specified in M as under-sampled if the model disagrees with the expectation $E(i, v)$.

More precisely, let $(i, v) \in [1..m] \times V$ be a tuple such that $v \notin U_i$ where U_i is the domain of M_i . We derive the extension $M'(i, v) = (M'_0, \dots, M'_{m-1})$ of M and (i, v) by defining

$$U'_j := \begin{cases} U_j & : j \neq i \\ U_i \cup \{v\} & : j = i \end{cases}, \quad M'_j(u) := \begin{cases} M_j(u) & : (j, u) \neq (i, v) \\ M_i(v) := E(i, v) & : (j, u) = (i, v) \end{cases}$$

At this point two possible situations arise.

(2a) Some positions (i, v) were marked as under-sampled. It follows that every M -monotone path contains an unexpected behavior for v in between the measurements p_i, p_{i+1} . This might be either a full oscillation, a dip or peak in the activity profile of v , or a change in activity in case (i, v) is the first or last measurement or v was not measured at all. The assessment finishes with the suggestion that the experiment should be repeated with a higher sampling rate in the intervals and for the components that were marked.

(2b) No positions were marked. In this case the model and data seem to fit perfectly together. But, the extensions $M'(i, v)$ are merely a local test that does not guarantee the existence of a path that satisfies all expectations at the same time. The existence of such a path requires an additional test.

Definition 24. Let $P = (p_0, \dots, p_m)$ be a time series and $M = (M_0, \dots, M_{m-1})$ a monotony specification. The best fit specification $B = (B_0, \dots, B_{m-1})$ of P and M is defined by $B_i : V \rightarrow \{\nearrow, \searrow, \bullet\}$ with

$$B_i(v) = \begin{cases} M_i(v) & : v \in U_i \\ E(i, v) & : \text{else} \end{cases}.$$

Hence, if a model is B -compatible then it satisfies the initial monotony assumptions M and the additional expected monotonies $E(i, v)$ at the same time and we conclude that the sampling rate was sufficient. If the model is not B -compatible then we conclude that the sampling rate was too low *somewhere* but also that the exact interval and component for which unexpected behaviors appear can not be determined by the model, because for each tuple (i, v) there are some paths that behave as expected but also some that do not.

3.5 Variations of Compatibility

The conditional reachability pattern $R(P, C)$ of Def. 21 was used only to define an LTL encoding of a monotony specification M . Here, C was defined to be $C = (\mu(M_0), \dots, \mu(M_{m-1}))$ with $\mu(M_i)$ being conditional next expressions of Def. 22 that ensure the monotony of specific components. The motivation in this section is to explore other applications of $R(P, C)$ but also of its CTL variant $R^E(P, C)$.

3.5.1 Robust Compatibility

So far, a model is said to be compatible with a time series if its transition system satisfies the corresponding linear time property, i.e., if there is a time series path in its STG. Whereas the existence of such a path demonstrates that the model is capable of reproducing the time series, it does not offer any information on how *robust* the path is with respect to perturbations in the sequence of component updates that lead from the first to the last state of the path.

As it has often been said that the dynamics of biological processes is robust in this respect it is desirable to formulate such a property in temporal logic. That is, we would like to query the existence of a path with the property that branching at any state will permit a continuation that is again a time series path.

Definition 25. A model is branching time compatible with a time series $P = (p_0, \dots, p_m)$ iff there is a time series path $x_1 \rightarrow \dots \rightarrow x_t$ such that for every $1 \leq s \leq t$ and every $y \in S : x_s \rightarrow y$ there is a $z \in S[p_m]$ and a path $y \rightsquigarrow z$ such that

$$x_1 \rightarrow \dots \rightarrow x_s \rightarrow y \rightsquigarrow z$$

is again a P -time series path.

Note that this definition is independent of which state represents which measurement. As a consequence the representatives of the original path and the branched path may be different. Note also that differently quantified, weaker versions of Def. 25 are possible and encoded similarly.

As its name suggests, we need the branching time logic to encode this property. To do so we use the existential CTL variant $R^E(P)$ of the nested reachability query $R(P)$ that was discussed in Def. 18. For clarity we define the queries that encode the existence of the path continuations separately. Denote by β_i for $0 \leq i \leq m$ the CTL formula

$$\beta_i := \mathbf{AX} (R^E(p_i, \dots, p_m)).$$

Hence, if $x \models \beta_i$ for a state $x \in S$ then for all its successors $y \in \text{Succ}(x)$ there is a P_i time series path with $P_i := (p_i, \dots, p_m)$, see Fig. 3.7 for an illustration.

Observation 7. Let $P = (p_0, \dots, p_m)$ be a time series. A model (V, F) is branching time compatible with P if and only if

$$TS \models R^E(P', C)$$

where $P' := (p_1, \dots, p_m)$, $C := (\beta_1, \dots, \beta_m)$ and $TS = (S, \rightarrow, S[p_0])$.

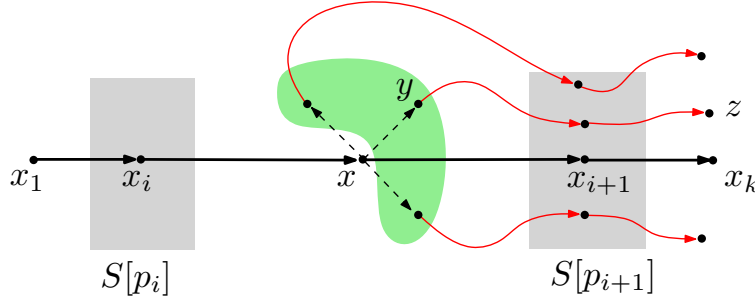


Figure 3.7: A model is branching time compatible with a time series if there is a time series path $\pi = x_1 \rightsquigarrow \dots \rightsquigarrow x_k$, such that for every successor $y \in \text{Succ}(x)$ of every state x of π there is a continuation $y \rightsquigarrow z$ to a state $z = z(y) \in S$ such that $x_1 \rightsquigarrow x \rightarrow y \rightsquigarrow z$ is also a time series path. The states along π must therefore satisfy $\beta_{i+1} = \mathbf{AX} (R^E(p_{i+1}, \dots, p_m))$.

Def. 25 can be seen as a first-order version of robustness with one path being robust and its continuations being regular paths. A second-order version, which requires that the continuations are themselves branching time compatible, is also possible. It is constructed by replacing $R^E(p_i, \dots, p_m)$ in the definition of β_i with the first-order query of Obs. 7. Inductively this can be generalized to any degree of robustness.

The next natural question is whether M -compatibility can also be generalized to branching time M -compatibility. This would be straightforward if a CTL formula existed that encodes the M -compatibility which is, by Thm. 1, impossible for general transition systems and very likely to be impossible for logical networks, as discussed in Sec. 3.3.1.

There is, however, a variation on the notion of monotony, which we call *strict monotony*, that is encodable in CTL. Recall, from Def. 19 that a path $x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_k$ is increasing in $v \in V$ if $x_i(v) \leq x_{i+1}(v)$ for all $i < k$. We say that v is *strictly monotone* if it is monotone for all successors of states along the path, not only the one that is next on the path. A component $v \in V$ is, for example, *strictly increasing* if $x_i(v) \leq y(v)$ for all $y \in \text{Succ}(x)$ and $i < k$.

Strict M -compatibility is easy to encode, in either LTL or CTL, using the tendency propositions $\delta_v \diamond k \in AP$ for $k \in \{-1, 0, 1\}$. It is achieved by modifying the sub-formulae $\xi(v, \nearrow)$ of Def. 22 by $\xi'(v, \nearrow)$ by

$$\begin{aligned}\xi'(v, \nearrow) &:= (\delta_v \geq 0) \\ \xi'(v, \searrow) &:= (\delta_v \leq 0) \\ \xi'(v, \bullet) &:= (\delta_v = 0)\end{aligned}$$

Also, strict M -compatibility, since it is encodable in CTL, can be extended to branching time strict M -compatibility by following the above constructions for branching time compatibility.

Note that the variations of branching time compatibility all deal and struggle with the non-determinism of the STG. A deterministic M -monotonic time series path $x_1 \rightsquigarrow \dots \rightsquigarrow x_m$ trivially satisfies all of the above definitions. So, how about querying the existence of deterministic time series paths?

3.5.2 Attracting Pathways

An *attracting pathway* is a trajectory with a restricted out-degree. The notion and terminology are taken from [97] where it is defined as "a pathway in which all or almost all edges with one vertex on the pathway are directed toward it". The authors state that attracting pathways are expected to be present in the STGs of models of biological systems because a "defining characteristic of biological systems is a remarkable insensitivity to stochastic fluctuations both in the environment and in the organism itself".

With propositions of the form $\Delta \diamond k$ (see Sec. 2.4) all of the above queries, compatibility, M -compatibility and branching time compatibility, can be extended to accept only time series paths with a restricted out-degree along specific path segments. To query, for example, the existence of a deterministic M -compatible path, using LTL, we replace each condition $\mu(M_i)$ of $C = (\mu(M_1), \dots, \mu(M_m))$ by

$$\mu(M_i) \wedge (\Delta = 1)$$

We make two remarks. First, the out-degree restriction can be chosen independently for each pair p_i, p_{i+1} . To query a path that is "increasingly deterministic" we could for example use the conditional nested reachability query with the conditions:

$$C = (\Delta = 3, \Delta = 2, \Delta = 1).$$

Second, the propositions $\Delta = 1$ and $\Delta = 0$ can be used to detect *limit cycles* and fixpoints in STGs, see Chap. 4.

3.5.3 Stability and Partial Stability

In addition to the monotony specification M_i in between the measurements p_i and p_{i+1} we may also demand that some components $W_i \subseteq V$ are stable when the time series path reaches measurement p_i .

This assumption is, for example, relevant in modeling perturbation experiments. Here, the set up is that some components are the so-called *external control* components. They may be activated or inhibited during or before the experiment by gene perturbations, stimulations of receptors, and so on. The remaining components are called *internal components* and their dynamics can not be influenced directly. It is then assumed that for some initial values for the control components, the internal network has settled into a fixpoint, i.e., that $f_v(x) = x(v)$ for all internal components v . Then a change in the activities of some external control components will change the conditions of some internal components and lead the network to the next internal fixpoint.

Definition 26. A time series path $x_0 \rightsquigarrow \dots \rightsquigarrow x_m$ satisfies the stability specification $W = (W_0, \dots, W_m)$ with $W_i \subseteq V$ iff for all $0 \leq i \leq m$:

$$\forall v \in W_i : f_v(x_i) = x_i(v).$$

The *control problem* for Boolean networks, on which the above description is based, is due to [98] and solutions using model checking and counterexamples were suggested in [79]. We add to their results that the control problem can be extended by all previous concepts of this chapter, i.e., by stability and monotony specifications, a restriction of out-degrees and robustness.

Stability specifications may be encoded in the conditional nested reachability queries $R(C, P)$, $R^E(C, P)$, $R^A(C, P)$ by using the tendency propositions $\delta_v \diamond k$ and modifying Def. 21. To query, for example, the existence of an M -compatible path that satisfies the stability specifications $W = (W_0, \dots, W_m)$ using LTL, replace each condition $\mu(M_i)$ of $C = (\mu(M_1), \dots, \mu(M_m))$ by

$$\mu(M_i) \wedge \bigwedge_{v \in W_i} \delta_v = 0.$$

A special case of stability specification is to require that the path ends in a fixpoint. This requirement is specified by

$$W = (\emptyset, \dots, \emptyset, V).$$

3.6 Software

A Python package, called TEMPORALLOGICTIMESERIES [54], with which discrete time series data can automatically be translated into the various temporal logic encodings is available. The input format is text-based, an example is given in Fig. 3.8.

The package includes the script "macros.py" which translates the input into temporal logic formulae by following one of four common use cases. They are

(1) compatibility, (2) monotonic compatibility, (3) branching time compatibility, and (4) the best fit specification. Each use case takes a discrete time series as an input and requires some parameters that decide the "flavor" of the encoding. **AsymptoticallyStable** is a Boolean parameter that determines whether the time series path should end in a fixpoint or not. **OutDegree** is a natural number that restricts the number of successors of each state along the time series path to a constant. Use case (1) has an additional parameter **TemporalLogic** that determines whether the encoding should be done in CTL or LTL and use case (2) requires, as an additional input, a matrix of monotony specifications $M = (M_0, \dots, M_{m-1})$ of Def. 20.

Input:

	v_1	v_2	v_3	
p_0				0
p_1				1
p_2				2
p_3				—

Output:

- (1) $v1=0 \ \& \ v2=0 \ \& \ v3=1$
- (2) $((((v2=1 \rightarrow X(v2 \geq 1)) \ \& \ (v2=2 \rightarrow X(v2 \geq 2))) \cup (v1=0 \ \& \ v2=1 \ \& \ ((v1=1 \rightarrow X(v1 \geq 1)) \ \& \ (v2=0 \rightarrow X(v2 \leq 0)) \ \& \ (v2=1 \rightarrow X(v2 \leq 1)) \ \& \ (v3=0 \rightarrow X(v3 \leq 0)))) \cup (v1=1 \ \& \ v2=0 \ \& \ (((v1=1 \rightarrow X(v1=1)) \ \& \ (v2=1 \rightarrow X(v2 \geq 1)) \ \& \ (v2=2 \rightarrow X(v2 \geq 2)) \ \& \ (v3=1 \rightarrow X(v3=1))) \cup (v1=1 \ \& \ v2=2 \ \& \ v3=1))))))$

Figure 3.8: An example of the input format for discrete time series and monotony specifications as used by `TEMPORALLOGICTIMESERIES` [54]. The symbols $<$ and $>$ are interpreted as *increasing* and *decreasing* respectively while $=$ is interpreted as *steady*. (top left) The text-based input file. (top right) The interpretation as a discrete time series with monotony specifications. (bottom) The output of the function `macros.monotonic_compatibility` is divided into the initial constraint (1) and the LTL formula (2).

3.7 Case Study: The Galactose Switch in Yeast

This section picks up where the case study of Sec. 2.5 left off. Here we test each of the models of the three viewpoints of the IRMA network against the various time series encodings. We focus on: (1) Compatibility, (2) Branching time compatibility, (3) Compatibility with restriction of out-degrees, (4) Compatibility with asymptotic stability, and the (5) Best fit specification. The goal is to discuss how each of the 9 models fits to the data and possibly select a "good model". As we will see, this is quite a challenge that involves many decisions regarding data discretization and the temporal encoding of the data. The questions regarding data discretization are where to place the threshold and how much uncertainty to permit. Regarding the temporal encoding of data, it is

not a priori true that we should expect that a "good model" is branching time compatible with the data or asymptotically stable. Although one may argue that these properties are biologically meaningful, it may be the case that the data is represented by a trajectory that is characterized by not being robust in our sense.

The first steps will therefore be to investigate how the encodings perform in an ideal scenario, on simulated data, and to discretize the data. The goal in the former is to understand what kind of compatibility can be expected between data and the candidate models. Finally we demonstrate the procedure for the assessment of the sampling rate of Sec. 3.4. Implications for model revision are discussed.

All encodings were generated using `TEMPORALLOGICTIMESERIES`, see [54] and the previous section.

3.7.1 Simulations

To get a sense for how well a match between data and model can be expected we generated artificial asynchronous switch-on and switch-off data for each of the 9 models. We followed the *random component semantics* for generating the time series which is also used in [44] and not the *random successor semantics*. In the former, the next state x_{t+1} in the simulation is generated from x_t by randomly selecting a component $v \in V$ to update, whether or not $tend_v(x_t) \neq 0$. If a component with $tend_v(x_t) = 0$ is selected then $x_{t+1} = x_t$ which seems realistic since we can not guarantee that the system has changed from one measurement to the next. On the other hand, the successor semantics selects a random successor x_{t+1} of x_t in (S, \rightarrow) . Here $x_{t+1} = x_t$ if and only if x_t is a fixpoint. Examples of the difference in simulating time series data are given in Fig. 3.9.

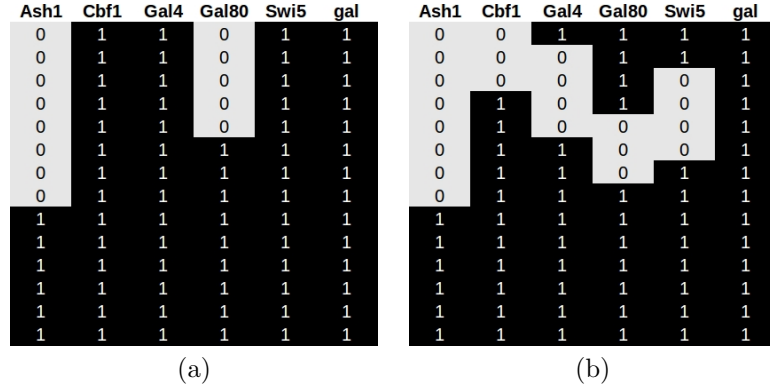


Figure 3.9: Two examples for simulated switch-on data ($gal = 1$) derived from the transcriptional model with $f_{Cbf1} = \overline{Ash1} + Swi5$. The time series in (a) was generated by the random component semantics while for (b) we used the random successor semantics that produces a true random walk in the STG. Our computer experiments are based on simulations of type (a).

For each of the 9 models we have generated 1000 time series for the switch-on and 1000 for the switch-off experiments. The time series were of the same lengths as the number of samples in [81], 14 and 18 respectively, and characterized by

a random initial state $x \in S[gal = 1]$ for switch-on and $x \in S[gal = 0]$ for switch-off simulations. Since the data is derived from the models we do not have to test compatibility or whether there is a best fit time series path. Both answers are a priori positive. Hence we tested only the remaining 3 questions: How many of the 1000 time series are (1) branching time compatible with the model under investigation, (2) have an asymptotically stable time series path, and (3) have a time series path with a restricted out-degree?

Regulation of Cbf1	Switch-On			Switch-Off		
	Ash1 AND Swi5	Ash1 OR Swi5	Swi5	Ash1 AND Swi5	Ash1 OR Swi5	Swi5
Mechanistic	342	352	348	262	964	258
Transcriptional	206	212	407	331	1000	363
Explicit G4G80	113	133	240	175	961	219

(a) Branching Time Compatibility

Regulation of Cbf1	Switch-On			Switch-Off		
	Ash1 AND Swi5	Ash1 OR Swi5	Swi5	Ash1 AND Swi5	Ash1 OR Swi5	Swi5
Mechanistic	1000	1000	1000	996	0	974
Transcriptional	997	996	999	993	0	917
Explicit G4G80	995	993	998	982	0	919

(b) Asymptotically Stable

Figure 3.10: The number of times, out of 1000, that a simulation is robust in (a) and asymptotically stable in (b). The numbers are very similar between rows (model type) but vary across columns (regulation of *Cbf1*). Highlighted in yellow is the prediction that, across all modeling assumptions and regulations of *Cbf1*, the switch-on data should be reproduced by a path that ends up in a fixpoint. Highlighted in blue is the prediction that whether or not the switch-off data is reproduced by a stable path is linked to the regulation of *Cbf1*.

The results for (1-2) are summarized in Fig. 3.10. A first observation is that for each of the 9 models and both types of experiments there are some simulations that are robust and some that are asymptotically stable. A perhaps surprising observation is that the numbers seem to be independent of the model type as the numbers do not vary much across rows. They do however vary across the columns, which correspond to different update functions of *Cbf1*.

Regarding the stability, about 99% of the simulated switch-on experiments end up in a fixpoint, independent of the type of model (rows) or the regulation of *Cbf1* (columns). We therefore note the following simulation-based prediction:

- *We expect to see that the switch-on data is reproduced by a time series path that is asymptotically stable.*

The results for the switch-off data suggest an equivalence between the regulation of *Cbf1* and asymptotic stability. Each number is either 0 or around 1000 with variations across columns but not rows. The prediction is:

- *If and only if $f_{Cbf1} \neq \overline{Ash1} + Swi5$ do we expect to see that the switch-off data is reproduced by an asymptotically stable path.*

The results for branching time compatibility are less conclusive. About 30% of the simulations are robust. A noteworthy remark is that in case of the switch-off data the regulation of *Cbf1* has, again, a strong effect on the robustness. If $f_{Cbf1} = \overline{Ash1} + Swi5$ (middle column) then almost all simulations are robust.

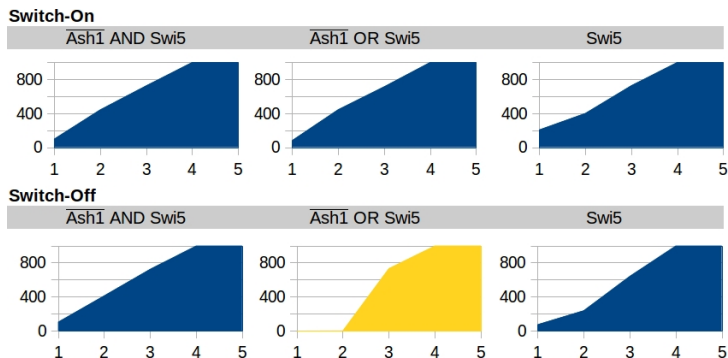


Figure 3.11: The number of out-degree restricted time series paths for the transcriptional model. The plots show the restriction $k = 1, \dots, n = |V|$ on the x -axis against the number of acceptable simulations on the y -axis. The data for the other two model types, the mechanistic and explicit viewpoints, is very similar and not shown. Highlighted in yellow is the scenario switch-off and $f_{Cbf1} = \overline{Ash1} + Swi5$ in which none of the 1000 simulations has a k -restricted path with $k = 1, 2$.

A possible explanation for these results may be that the regulation of *Cbf1* determines whether the STG contains fixpoints or sustained oscillations in the form of large SCCs.

The results for the existence of out-degree restricted paths are given in Fig. 3.11. As the simulations hardly vary across different model types, only the ones for the transcriptional models are shown. The plots have the restriction parameter k on the x -axis and the number of simulations that have a k -restricted time series path on the y -axis. The plots are naturally cumulative: if there is a k -restricted path then the same path is $(k + 1)$ -restricted. Note that any path is trivially $k = |V| = 5$ restricted but for the considered models also $k = 4$ restricted, because *gal* is a stable input that can not change along any path. The shape of the plots is very similar across different update function for *Cbf1* with about 10% of simulations being deterministic ($k = 1$) and an apparent linear increase. An exception is the switch-off data and $f_{Cbf1} = \overline{Ash1} + Swi5$, highlighted in yellow in Fig. 3.11. In that case $k \geq 3$ is required for compatibility. We hence note the prediction:

- If $f_{Cbf1} = \overline{Ash1} + Swi5$ then we expect the switch-off data to be incompatible for $k = 1, 2$.

3.7.2 Data Discretization

The next step is to discretize the switch-on and switch-off data. The PCR data of the 5 switch-on and 4 switch-off experiments is available as a spreadsheet in the supplementary materials of [81]. According to [81] the replicates were made to reduce the influence of noise on the data. Averaged and normalized switch-on and switch-off time series are available. In Fig. 3.12 we illustrate the available data. As an example we plotted the 9 individual and the 2 normalized time series of *Ash1*. The mRNA concentrations are given in terms of $2^{-\Delta Ct}$ values,

for details see [81]. As in [81, 82] we also assume that an increase or decrease in mRNA leads to a proportional change in protein concentration.

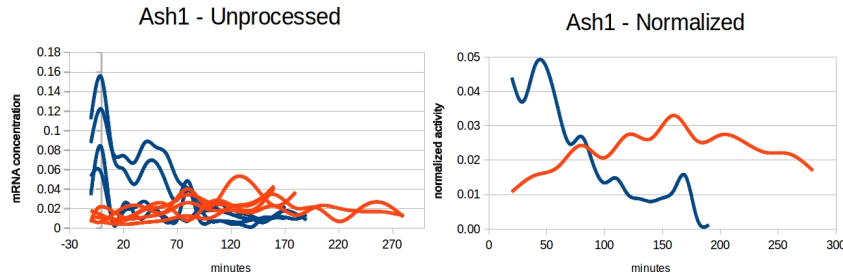


Figure 3.12: An example for the data available in [81]. On the left, the unprocessed PCR data of *Ash1* which consists of 5 switch-on time series in red and 4 switch-off time series in blue. On the right, the normalized data. Although the unprocessed curves seem generally increasing for the switch-on and decreasing for the switch-off data, the trends are much more pronounced in the normalized curves.

To binarize the data we assume that all 8 interactions have highly non-linear response profiles for varying regulator concentrations, i.e., that there is a threshold value for each interaction below which the regulator is ineffective and above which it is effective at a constant rate. The goal of the binarization step is to determine the threshold value of each interaction. In addition, since we are interested in a Boolean model for the IRMA network, we assume that for *Swi5*, which is the only component that regulates several other components, all its thresholds are at about the same value.

As a method for computing the threshold for each component we have tested 3 naive approaches and 4 algorithms. As naive approaches we used: taking the mean value, the median value, and the midpoint $\frac{1}{2}(\max - \min)$ between the maximum and minimum values. The algorithms we used are k -means clustering, the first edge and max edge detector methods published in [99] and a scan statistic method. All algorithms are implemented in the software package BOOLNET [44].

As discussed in [81] the initial peaks in *Gal4* and *Gal80* could be due to carbon starvation effects. The values may be much above the actual regulation thresholds which would bias the computation of the thresholds. But, since we can not judge whether these values are outliers or not we have decided to be passive and keep the values.

Since there are two time series for each component we have a choice as to whether we want to discretize the joint data or each time series separately. In general it should be preferable to discretize the joint data because it potentially contains a larger range of activity values and hence a clearer picture of what should be considered to be an effective or ineffective concentration level. We make the assumption that there is a single threshold that works for both the switch-on and the switch-off data. To demonstrate the possibility of allowing uncertainties in the thresholds we removed all values that lie within the interval $(t - u, t + u)$, where t is the threshold and $u := 0.05 \cdot (\max - \min)$ is 5% of the total activity range, from the discrete time series. It should be remarked that implicit to all discretization methods is the assumption that each component does indeed

cross its regulation threshold somewhere in the data. This assumption is slightly problematic (why should constant components be disallowed?) but probably only avoidable by insights into the biological background that we are lacking. The thresholds are illustrated in Fig. 3.13.

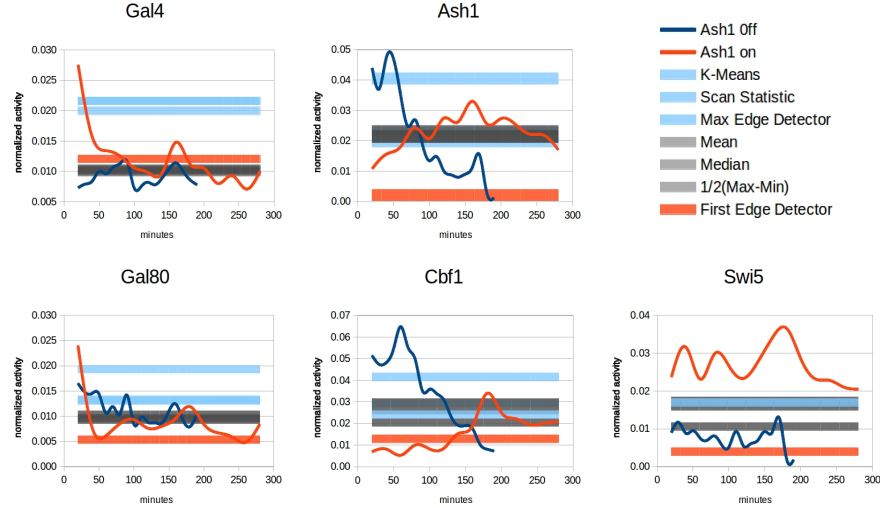


Figure 3.13: The normalized data and thresholds obtained by the mean, median or midpoint method and the k -means, edge detector (first and max) and scan statistic method. We have grouped the discretization methods into three groups. Those that tend to be *high* are colored in blue, those that tend to be *low* in red and those in between, which we refer to as *mean*, in gray. The width of the thresholds indicate a 10% uncertainty and values inside the threshold are marked as *uncertain* in the discrete time series.

The thresholds fall, qualitatively speaking, into three groups depending on whether they tend to be above, below or at the mean value of the data. *High* thresholds lead to "calm" time series with many 0's, *mean* threshold values lead to "erratic" time series with many changes from 0 to 1 and *low* threshold values lead to "calm" time series with many 1's as illustrated in Fig. 3.19. The "erratic" behavior of mean threshold time series occurs because the measurements accumulate and oscillate around the threshold, as can be seen for *Gal4* and *Gal80* in Fig. 3.13.

The thresholds obtained from the naive approaches are often located where a majority of the measurements accumulate, i.e., at the steady plateaus, if there are any. Since the measurements usually oscillate around these plateaus the naive approaches seem to be emphasizing the noise in the data. The result are "restless" time series that require a lot of transitions. A good example for this effect can be seen in the discretization of *Gal4*. The "restlessness" can be countered by increasing the value of u .

As an illustration for what can be done in case of uncertainties regarding the discretization method, which we are also facing in this case study, we will construct a so-called *consensus time series* for each group of thresholds (low, mean, high) which records the values that are agreed upon by all discretization methods in the respective group. Formally, the consensus time series of two

time series $P = (p_1, \dots, p_m)$ and $Q = (q_1, \dots, q_m)$ is $C = (c_1, \dots, c_m)$ where $Set_{c_i} := Set_{p_i} \cap Set_{q_i}$ for all $1 \leq i \leq m$ and $Set_p \subset V \times \mathbb{B}$ is the set representation of a symbolic state $p \in Sym$, as defined in Sec. 2.1.1. The consensus time series of a set of time series is then obtained by applying the pairwise construction iteratively. The results are shown in Fig. 3.14.

Note that a consensus time series has two types of uncertain measurements, those that are uncertain because they are uncertain for every time series of the group (red cells in Fig. 3.14) and those that are uncertain because there are two time series in the group whose values are not equal (blue cells in Fig. 3.14).

Note also that results based on the consensus time series are "cautious". That is, if a model is compatible with any time series in the group then it is also compatible with the consensus time series. This is true for all notions of compatibility discussed in this chapter.

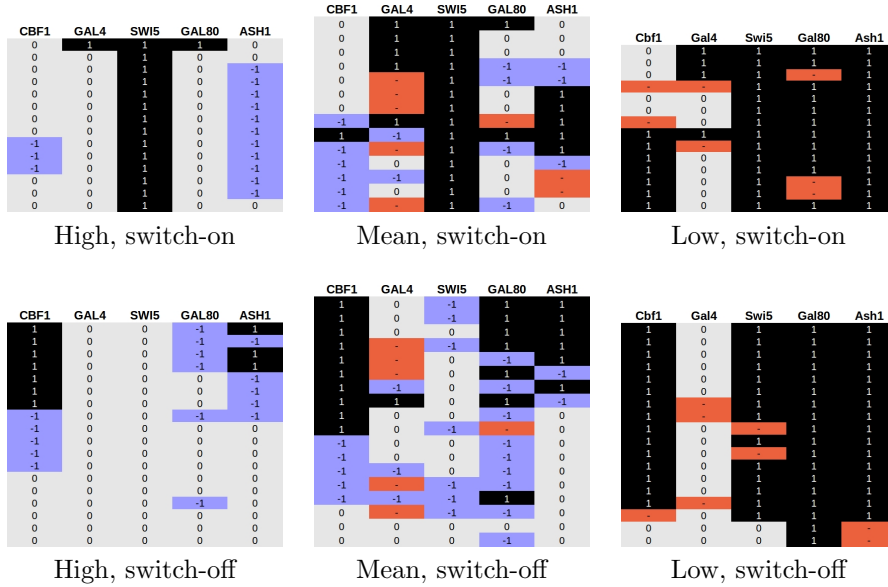


Figure 3.14: The consensus time series for the high, mean and low threshold values. Blue cells show disagreements among the discretization methods and are interpreted, like red cells, as uncertain values in the temporal logic encodings.

Finally, we remark that it is encouraging to observe that although discretization is a sensitive step that potentially has a strong influence on the outcome of the time series encodings and subsequent analysis, it seems that the threshold values are likely to belong to a smaller number of groups (high, mean, low in our case). In practice it may therefore be feasible to test all discretization methods in question and work with a small number of consensus time series.

3.7.3 Compatibility

In this section we test each combination of models, transition relation and time series for compatibility. The model parameters are: 3 models types, 3 update functions for *Cbf1* (see Eq. 2.4) and 3 transition relations. The temporal logic encoding parameters are: 3 pairs of consensus time series (switch-on, switch-off)

and 3 variations of compatibility (basic, branching time, best fit), each of which can be tested with an out-degree restriction of $k = 1, 2, 3$. Since the parameters can be chosen independently, we are faced with and have tested

$$3 \times 3 \times 3 \times 3 \times 3 \times 3 = 3^6 = 729$$

different combinations. It turns out that there are only 36 compatible combinations. They are listed in Fig. 3.20. Note that we say that a model satisfies a variation of compatibility with a consensus time series if it does so for both, the switch-on and the switch-off.

The aim of this section is describe some properties of the compatible combinations that caught our attention, to compare them with the simulation-based predictions of Sec. 3.7.1 and possibly to select a "best" model or a "best" discretization.

First, some properties that caught our attention.

- None of the synchronous models are compatible with any time series. This may be so because the time series contain a measurement twice but with different successors or simply because it is unlikely that one of our 9 pre-defined synchronous models is compatible with a data-based time series. It seems to us that this will in general be the case. This kind of model validation is hard with synchronous updates.
- None of the models have asymptotically stable time series paths. This is in direct opposition to the expectation, which was raised in Sec. 3.7.1, that the switch-on data should be reproduced by an asymptotically stable path.
- There are no best fits between any model and any time series. This should be interpreted as an overall poor fit between the data and the models.
- The time series paths in this case study are highly non-deterministic. They contain at least one state with an out-degree $k \geq 4$ except when considering the mixed transition relation and the mechanistic type model in which case there is a path with out-degree $k \leq 3$, see Fig. 3.20.
- Since $x \hookrightarrow y \Rightarrow x \rightarrow y$ we asked whether the mixed update \rightarrow ever made a difference in terms of compatibility compared with the same asynchronous system. There are two cases. First, as might have been expected, \rightarrow can increase the "determinism" of a path by decreasing the out-degrees along the states of a path. See for example the mechanistic model with conjunctive update and high thresholds in Fig. 3.20. Second, (S, \rightarrow) may be compatible with a time series whereas (S, \hookrightarrow) is not, see the mean thresholds for the mechanistic model.
- There is almost a correspondence between the disjunctive update of *Cbf1* and the existence of a robust time series path. The only robust time series path that is not linked to the disjunctive update of *Cbf1* is the explicit type model with high thresholds values and $f_{Cbf1} = Swi5$, see also Fig. 3.20.

Now, regarding model selection. We would like to demonstrate that this kind of exhaustive testing can be used for defining selection criteria. As examples consider the following scenarios

- (1) We are interested in the model that is compatible with all three threshold levels. There are 3 models that have this property. They are the explicit type models with the asynchronous or mixed update strategy.
- (2) We are interested in the model with the "best" time series path. The disjunctive, mixed update, mechanistic model is the only one that has a robust, out-degree restricted path (with $k \leq 3$). The corresponding time series are discretized by high-valued thresholds.

One may also count the number of times each update of *Cbf1* occurs amongst the compatible models (*disj*:13, *swi5*:13, *conj*:10), how often each model type occurs (*explicit*:16, *transcr*:12, *mech*:8), and so on. For a discussion on analyzing sets of models and a software for their management, see Chap. 5.

3.7.4 Assessment of Sampling Rate

To assess the data quality we followed the procedure described in Sec. 3.4. It requires to test for all components $v \in V$ and all time points $1 \leq i \leq m$ of a given time series $P = (p_1, \dots, p_m)$ and model whether there is a $E(v, i)$ -monotone time series path where $E(v, i)$ is the expected monotony specification of Def. 23.

The first step is to choose the model that the assessment should be based on. We have tested several scenarios and decided to use the transcriptional model with $f_{Cbf1} = Swi5 \cdot \overline{Ash1}$, the mean thresholds and the asynchronous update \hookrightarrow for an illustration because in this scenario there are three under-sampled intervals. Following the assessment procedure we generated for each (v, i) , of which there are $5 \cdot (14 - 1) = 65$ for the switch-on and $5 \cdot (18 - 1) = 85$ for the switch-off time series, the respective LTL specification and tested whether there is a corresponding monotone time series path.

For each time series we recorded those positions (v, i) for which no such path exists because, as discussed in Sec. 3.4, each such (v, i) implies a change in activity of v on the path segment $x_i \rightsquigarrow x_{i+1}$ where x_i, x_{i+1} represent the measurements p_i, p_{i+1} of the time series, respectively, that is not inherent in the data. Depending on $x_i(v)$ it will be $0 \rightarrow 1 \rightarrow 0$ if $x_i(v) = 0$, which we call an unobserved *peak*, or $1 \rightarrow 0 \rightarrow 1$ if $x_i(v) = 1$, which we call an unobserved *dip*, or a single transition, either $0 \rightarrow 1$ or $1 \rightarrow 0$, if p_i is the last measurement that includes v or p_{i+1} the first. The prediction for these (v, i) in terms of the sampling rate is that:

- *A replicate of the experiment with an increased sampling rate in the interval $[i, i + 1]$ will reveal previously unobserved dips below the discretization threshold or peaks above the threshold for the activity of v .*

For the chosen scenario, the assessment predicts three unobserved oscillations, two for the switch-off experiment and one for the switch-on experiment. We illustrate them by inserting an additional row into the tabular representation of the time series that shows which components are predicted to oscillate, see Fig. 3.15 and Fig. 3.21. To get an impression for what the oscillations might look like in terms of the expression curves, we have modified the original curves (by hand) in Fig. 3.16.

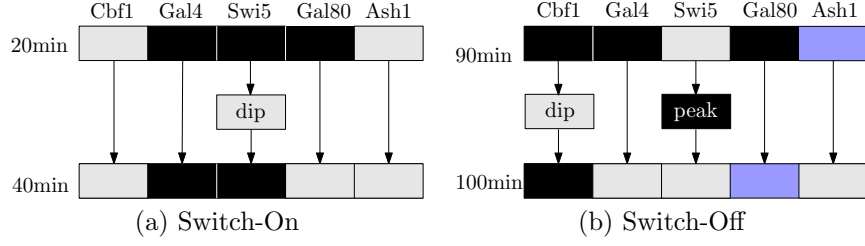


Figure 3.15: Cut-outs of the mean threshold time series that show the intervals for which the sampling rate assessment predicts oscillations. The complete time series is shown in Fig. 3.21. The prediction for the switch-on time series in (a) is a dip in activity for *Swi5* in between the 1st and 2nd measurements, in the interval [20min..40min]. For the switch-off time series (b) a dip and a peak are predicted for *Cbf1* and *Swi5* respectively in between the 8th and 9th measurements in the interval [90min..100min].

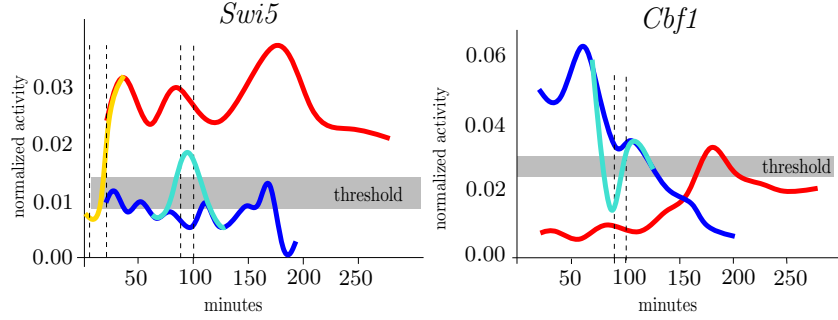


Figure 3.16: Based on the assessment of the sampling rate we manually modified the expression curves of *Swi5* and *Cbf1* to fit the predictions. Predictions for the red switch-on curve are given in yellow, corrections for the blue switch-off curve in turquoise. The corresponding time intervals are indicated by the dashed vertical lines.

Finally, we asked whether the unobserved oscillations can be explained in terms of the model. Consider, for example, the dip of *Swi5* in the switch-on time series between the measurements p_0 and p_1 . The discretization for these measurements is precise in all components and so the problem of deciding whether there is a time series path decouples into two separate compatibility problems $P_1 := (p_0, p_1)$ and $P_2 := (p_1, \dots, p_{13})$ as discussed in Sec. 3.2. In this case it is easy to see why there is no P_1 time series path that is monotone in *Swi5*. First, the monotony requires that *Swi5* is steady at 1 since $\text{Map}_{p_0}(\text{Swi5}) = \text{Map}_{p_1}(\text{Swi5}) = 1$ along the path. Second, the measurements require that *Gal80* decreases since $\text{Map}_{p_0}(\text{Gal80}) = 1 > \text{Map}_{p_1}(\text{Gal80}) = 0$, i.e., that $f_{\text{Gal80}} = 0$ somewhere along the path. Third, the model states that $f_{\text{Gal80}} = \text{Swi5} = 1$ everywhere along the path, a contradiction.

The same argument can be used for the predicted oscillation of *Cbf1* in the switch-off data. Here *Gal4* is required to decrease while *Cbf1* is steady at 1, which contradicts $f_{\text{Gal4}} = \text{Cbf1} = 1$. Not all cases of predicted oscillations are as easy to explain. Requirements can propagate, for example, where a

change in activity of one component triggers an oscillation in its regulators and in turn in their regulators, and so on. Explanations for the predictions are further obstructed because they may rely on components that were recorded as uncertain but have become fixed on every feasible path. Hence, to explain a prediction we might have to check which states of $S[p_i]$ are actually reachable from $S[p_0]$.

We conclude with two observations. (1) Regarding the assessment based on different models or time series, the predictions are very similar across all types of models and update functions for *Cbf1*. Often the assessment returns no predictions but if it does then the problematic intervals are in the range $[90min - 100min]$ for the switch-off and $[20min - 40min]$ for the switch-on data. (2) In practice one might want to allow uncertainties in the model that the assessment is based on. This is possible and the natural approach is to take the union of the (v, i) , i.e., all (v, i) that are predicted by *some* model, which is also called *brave reasoning*, or the intersection of the individual results, i.e., all (v, i) that are predicted by *every* model, which is also called *cautious reasoning*. The question of how to reason over sets of models is treated in more detail in Chap. 5.

3.8 Discussion

The starting point of this chapter were discrete time series observations and the question of compatibility between data and a given model. We suggested various notions of compatibility involving assumptions about the stability, robustness and monotony of the system. All time series encodings followed essentially the same nested reachability pattern $R(P, C)$ with symbolic states P and conditions C . In general, P and C can be arbitrary expressions of a given temporal logic. Measurements may, for example, be arbitrary state descriptions instead of symbolic states. A list of the ingredients we have discussed in this chapter is given in Fig. 3.17.

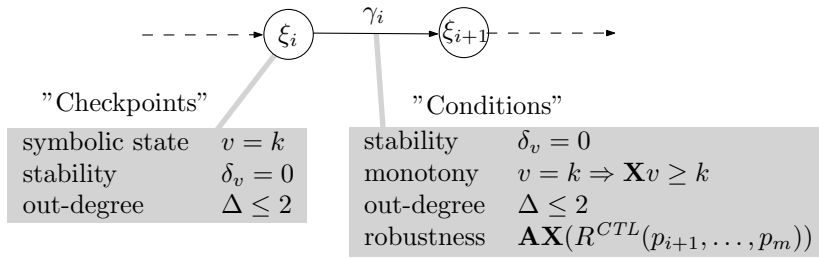


Figure 3.17: The time series encodings consist of "checkpoints" and "conditions" with examples for the properties discussed in this chapter.

One aspect of the linear time series properties that has only been addressed implicitly so far is their length in terms of number of transitions. As discussed in [86], it may serve as a selection criterion if two models are otherwise equally suitable. Ordinary model checkers like NUSMV can not report the length of the shortest witness of an LTL property. Extended algorithms are required to do so, see for example [86] and the model checker PARSYBONE [100]. A

similar shortcoming is the inability of reporting all states that satisfy a CTL property which lead to the development of ANTELOPE [74]. A workaround for determining the shortest witness of an LTL property is to encode the length as a part of state space, as was proposed in [79], and then to iteratively test the validity of the property using different bounds.

An alternative is to use bounded LTL model checking (BMC) as introduced in [101] which iteratively increases the depth k of the search for a witness until the property is satisfied for the first time, the computation runs out of memory or an upper bound K is reached. BMC relies on SAT solvers rather than decision diagrams and it has frequently been pointed out that BMC and symbolic model checking are complementary in the sense that either may fail where the other can still compute the answer. We would also like to point out that bounded model checking was originally introduced for LTL and has, to our knowledge, not been transferred to CTL. Since CTL properties can, in general, not be proved or disproved with paths it is not so obvious how to even define a bound for the validity of a branching time property. As discussed in [102], tree-like witnesses exist only for the existential fragment ECTL where only the path quantifier \mathbf{E} is allowed and negation is restricted to atomic sub-formulae.

A third alternative is to use constraint programming instead of model checking. Examples are [46, 47] with the Prolog-based tool GNBOX and [48] with the ASP-based tool SYSBIOX. Although the aim of this work is to infer knowledge from data and partially specified models, the same algorithms can of course be used for single, fully specified models. The authors of [46–48] have already defined path predicates to query the existence of a bounded path between two symbolic states. In [47] one can, for example, query $path(M, [S0, \dots, S1], 48)$ to find out if there is a path $x \rightsquigarrow y$ of at most 48 transitions in the asynchronous STG of $M = (V, F)$ such that x and y satisfy the descriptions $S0$ and $S1$, respectively. According to [47]

”The most computer-intensive queries are those involving paths. For such queries our approach is limited to networks of medium size.”

As is the case for BMC and symbolic LTL model checking, it seems that declarative approaches and model checking may complement rather than replace each other.

A challenging question related to model validation is the design of *model revision* algorithms. Consider for example the assessment of the sampling rate in the previous section. If additional experiments do not reveal the predicted oscillations (or if the sampling rate is known to be sufficient) one may also attempt to revise the interaction thresholds or the model itself.

The data assessment for *Cbf1* indicates problems in the interval $[90min - 100min]$. They could possibly be resolved by either increasing the threshold from about 0.03 to about 0.05 (units of $2^{-\Delta C^t}$) or by introducing another threshold at 0.05 and hence changing the model and state space by increasing the maximal activity of *Cbf1*. See Fig. 3.18 for an illustration. It would be desirable to develop a *threshold assessment method*, based also on the idea of monotone paths, that is capable of suggesting a ”correction” to thresholds by either changing their values or adding additional thresholds.

Finally, questions for future work might address to the following issues:

- (1) Can the missing or uncertain value values in the discrete time series be predicted by a sequence of queries similar to the sampling rate assessment?

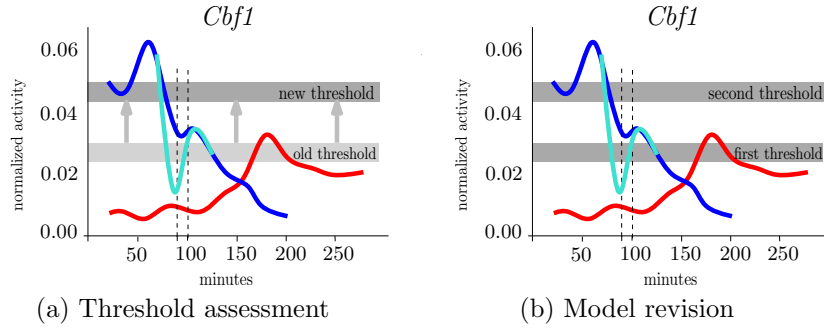
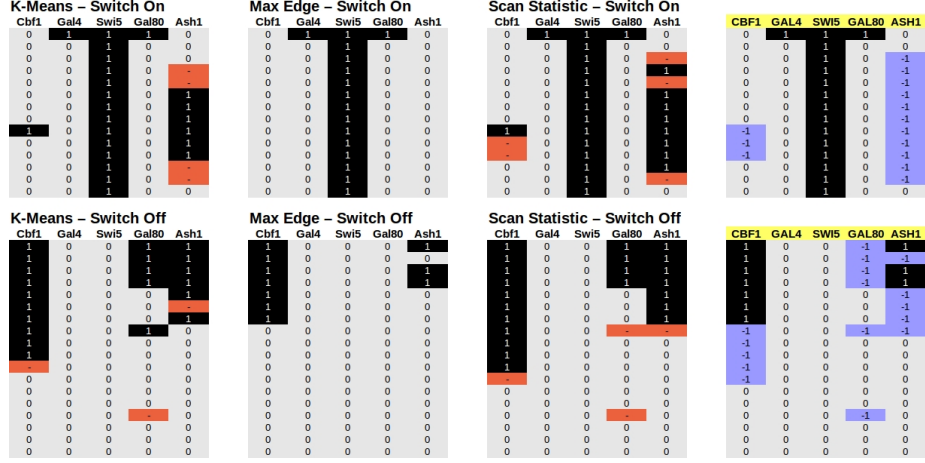
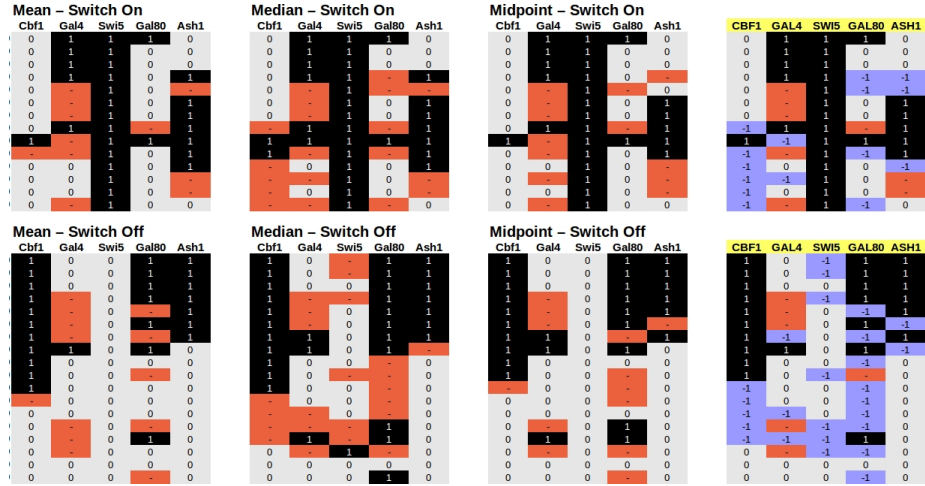


Figure 3.18: The sampling rate assessment procedure predicts a dip in the activity of *Cbf1* below its discretization threshold of 0.03 (units of $2^{-\Delta C^t}$). A *threshold assessment method* might propose either an increase of the given threshold from 0.03 to 0.05 as shown in (a) or introduce an additional threshold at 0.05 as shown in (b).

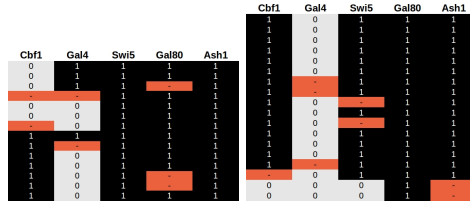
- (2) What can be learned about the sequence of events? Which components change first, which ones next?



(a) High threshold values, many 0's.



(b) Mean threshold values, "erratic" behavior.



(c) Low threshold values, many 1's.

Figure 3.19: The original time series and the consensus time series on the right of (a) and (b). Black, gray and red cells indicate values above, below and within a 5% range of the respective thresholds. Blue cells in the consensus time series indicate disagreements among time series in the respective group.

	f_{Cbf1}	STG	Thresholds	Robust	k	#
Transcr. Models	<i>conj, swi5</i>	$\hookrightarrow, \rightrightarrows$	mean, high			8
	<i>disj</i>	$\hookrightarrow, \rightrightarrows$	mean, high	✓		4
Mech. Models	<i>conj, swi5</i>	\rightrightarrows	high		≤ 3	2
	<i>disj</i>	\rightrightarrows	high	✓	≤ 3	1
	<i>swi5</i>	\rightrightarrows	mean		≤ 3	1
	<i>disj</i>	\rightrightarrows	mean	✓		1
	<i>conj, swi5</i>	\hookrightarrow	high			2
	<i>disj</i>	\hookrightarrow	high	✓		1
Explicit Models	<i>conj, swi5</i>	\hookrightarrow	mean, high			4
	<i>disj</i>	$\rightrightarrows, \hookrightarrow$	mean, high	✓		4
	<i>disj, swi5</i>	$\rightrightarrows, \hookrightarrow$	low			4
	<i>conj</i>	\rightrightarrows	high			1
	<i>swi5</i>	\rightrightarrows	high	✓		1
	<i>conj, swi5</i>	\rightrightarrows	mean			2
						36

Figure 3.20: Key to symbols: \hookrightarrow and \rightrightarrows are the asynchronous and mixed transition relations. The three different update functions for *Cbf1* are referred to by *conj*, *disj* and *swi5*, see Eq. 2.4. They represent $f_{Cbf1} = Swi5 \cdot \overline{Ash1}$, $f_{Cbf1} = Swi5 + \overline{Ash1}$ and $f_{Cbf1} = Swi5$, respectively. The table shows all compatible combinations of models, transition relations and discretization methods. The table is horizontally divided into the three model types: transcriptional, mechanistic and explicit (see Sec. 2.5). The column " f_{Cbf1} " refers to the update function of *Cbf1*, the column "STG" to the transition relation, "Threshold" indicates which consensus time series was used (see Sec. 3.7.2), a check mark in "Robust" means that the respective models are branching time compatible, the column " k " indicates whether the respective models are compatible with an out-degree restriction of k and "#" counts the number of individual combinations referred to by each row.

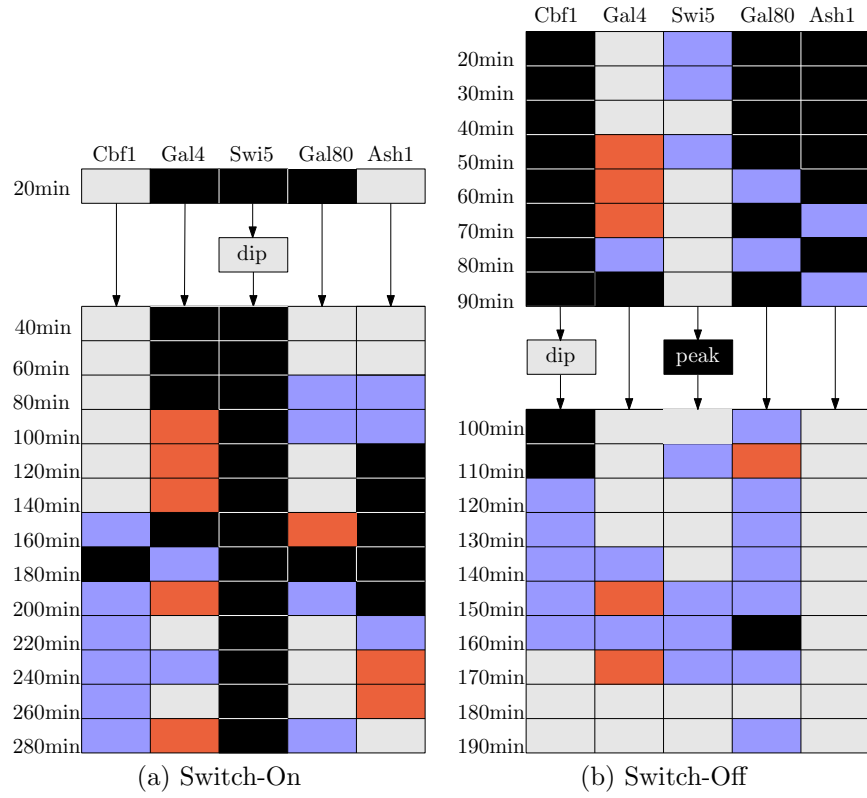


Figure 3.21: The positions of the predicted oscillations of the sampling rate assessment within the discrete time series. The assessment is based on mean thresholds, the transcriptional model with $f_{Cbf1} = Swi5 \cdot Ash1$ and the asynchronous transition relation.

Chapter 4

Asymptotics of a Model

The topic of this chapter are the attractors of logical networks. Attractors are used to predict, for example, the outcome of virus infections where in some cells the virus multiplies and kills its host and in some cells the virus and host establish a symbiosis, see [6]. They are also used to model processes like cell differentiation. In [7] they are computed to predict stable patterns of gene expression which correspond to different cell types of *A. thaliana*.

In Sec. 4.1 we define attractors in terms of recurring states on infinite paths. We then ask how model checking can help in predicting the asymptotics of logical models. Sec. 4.2 collects various queries related to the location in state space and the stability of components. The main contribution of this chapter begins in Sec. 4.3 with a recapitulation of the notion of a seed. The relationship between seeds and attractors is discussed in Sec. 4.4. We then introduce the prime implicant graph in Sec. 4.5. It is the basis of our method for finding seeds in Sec. 4.6. The chapter concludes with suggestions for a complete analysis of large networks and a case study of a MAPK signaling network.

This chapter is an extension of results published in [103].

4.1 Attractors

The asymptotic behaviors of a model are defined via the long-term properties of the paths of its state transition graph. Given a path $\pi \in Paths_\infty(S)$, it is natural to say that those states that are visited infinitely often, i.e., belong to the suffix $\pi[k..]$ for every $k \in \mathbb{N}$, make up a particular asymptotic behavior of the model. If a state $x \in S$ belongs to an asymptotic behavior it must therefore also belong to a strongly connected component $U \in SCC(S, \rightarrow)$, see Def. 10. For (S, \rightarrow) , and deterministic STGs in general, the SCCs are exactly the cycles in the transition graph. For (S, \leftrightarrow) , and other non-deterministic STGs, the situation is a bit more involved because now cycles may intersect. Among the SCCs of a model the *attractors* are particularly interesting.

Definition 27. A strongly connected component $X \in SCC(S, \rightarrow)$ such that for all $\pi \in Paths_\infty(X)$ and for all $i \in \mathbb{N}$: $\pi[i] \in X$ is called an attractor of (S, \rightarrow) . The set of attractors is denoted by $Attr = Attr(S, \rightarrow)$.

Attractors are therefore $X \in SCC(S, \rightarrow)$ such $y \in X$ for all $y \in Succ_{\rightarrow}(x)$

with $x \in X$. In the context of the SCC graph of (S, \rightarrow) , see Def.12, they are also called *terminal SCCs* because their out-degree in the SCC graph is 0.

Attractors capture long-term behaviors that are robust with respect to branching in time. That is, given a path $\pi \in Paths_\infty(S)$ that visits the states of an attractor $X \in Attr$ at some time j , i.e., $\pi[j] \in X$, then any path π' that is obtained from π by branching at some time $j' > j$, i.e., $\pi[..j'] = \pi'[..j']$ but $\pi[j'+1] \neq \pi'[j'+1]$, will necessarily remain within X . This property is not true for non-terminal SCCs. There is always at least one path obtained by branching that will leave a non-terminal SCC and lead to a different long-term behavior. Hence, the special role of attractors is that one can argue that asymptotic states obtained from them are stable with respect to the order with which the activities of components are updated.

We distinguish two types of attractors depending on their size, the *steady states* and the *cyclic attractors*:

$$\begin{aligned} Steady &= Steady(S, \rightarrow) := \{x \in S \mid \{x\} \in Attr(S, \rightarrow)\} \\ Cyclic &= Cyclic(S, \rightarrow) := \{X \subseteq S \mid X \in Attr(S, \rightarrow), |X| \geq 2\}. \end{aligned}$$

A definition that is related to attractors is that of a *trap set*. Trap sets do not require the states in question to be strongly connected.

Definition 28. . A trap set in (S, \rightarrow) is a subset $T \subseteq S$ such that $\pi[i] \in T$ for all paths $\pi \in Paths_\infty(T)$ and $i \in \mathbb{N}$.

Note that every trap set contains at least one attractor and that trap sets can be characterized by the following condition.

Observation 8. A set $T \subseteq S$ is a trap set in (S, \rightarrow) iff $y \in T$ for all $y \in Succ_{\rightarrow}(x)$ with $x \in T$.

The following section explores how model checking can help in understanding the asymptotics of a logical network (V, F) .

4.2 Attractor Queries

Many algorithms for the detection of attractors of a given system exist, see for example [19–29]. They can be grouped into different themes depending on whether they focus on a particular transition relation and on whether they impose a size limit on the computed attractors. Some work only for synchronous transition systems (e.g. [19, 21]) while others assume the asynchronous update (e.g. [25]). Some are specifically designed to detect steady states (e.g. [20]), others can detect cyclic attractors but only of a certain size (e.g. [22]). Furthermore, some are deterministic and exhaustive, while others are stochastic and incomplete: they rely on sampling and partial state space exploration.

In addition, most algorithms include a reduction step during which the original problem is transformed into an equivalent smaller problem, for example by replacing cascade components which can be shown to be irrelevant for the detection step. Since model reduction techniques potentially affect the efficiency of any analysis algorithm they have been addressed explicitly (e.g. [30, 31, 60]). Intuitively, the computational complexity of finding attractors depends strongly

on the number of feedback circuits in the interaction graph. It is, for example, easy to find all attractors of a network whose interaction graph is feedback-free.

The aim of this section is to gather basic results regarding the use of model checking when deciding if a model (V, F) is capable of a specified asymptotic behavior. Each of the following queries, except Obs. 16, relies on CTL model checking. LTL versions, obtained by removing the quantifiers, are also possible but will relate to the weaker interpretation of asymptotic behaviors and therefore non-terminal SCCs in (S, \rightarrow) . The principle difference to the above mentioned algorithms is that most of the following queries require a specification that describes some aspect of the asymptotic behavior.

It is worth noting the capability of model checking algorithms to produce a *counterexample* if $TS \not\models \phi$ or, equivalently, a *witness* if $TS \models \phi$, see [16]. For LTL, a witness or counterexample is a path fragment which proves that TS satisfies or refutes ϕ . For CTL, a state formula is proved or refuted by returning an initial state $x \in I$ such that $x \models \phi$, resp. $x \not\models \phi$. For each of the following attractor queries we will mention how counterexamples and witnesses can be used for further analysis. Although CTL proofs can be extended to paths, which are called *traces* [16], and even trees [102], we restrict our remarks to the basic notion. NUSMV is capable of producing state-based counterexamples and witnesses for CTL.

The first question that can be answered with CTL queries is deciding if there is a cyclic attractor or a steady state in a given subset of states. The queries are based on propositions of the form $\Delta \diamond k$ which refer to the number of successors of a state, see Sec. 2.4. For the next four queries, let $d \in StateDesc$ be a state description (see Def. 2) and $TS = (S, \rightarrow, I)$ with the initial states $I = S[d]$.

Observation 9. *There is $x \in Steady$ such that $x \in S[d]$ iff $TS \models (\Delta = 0)$. A witness is $x \in Steady$ with $x \in S[d]$.*

Observation 10. *There is $X \in Cyclic$ such that $X \subseteq S[d]$ iff $TS \models \mathbf{AG}(\Delta \geq 1 \wedge d)$. A witness is $x \in S[d]$ such that $X \in Cyclic$ for all $X \in Attr$ that are reachable from x .*

Note that to ensure that $X \subseteq S[d]$ we need to test $\mathbf{AG}(\Delta \geq 1 \wedge d)$ instead of $\mathbf{AG}(\Delta \geq 1)$. It can also be decided whether some set $T \subseteq S$ contains, or is itself, a trap set. The difference in the respective queries is the satisfaction relation.

Observation 11. *There is a trap set $T \subseteq S[d]$ iff $TS \models \mathbf{AG}(d)$. A witness is $x \in S[d]$ such that $x \in T$.*

Observation 12. *$S[d]$ is a trap set iff $TS \models \mathbf{AG}(d)$. A counterexample is $x \in S[d]$ such that there is a path $x \rightsquigarrow y$ for some $y \notin S[d]$.*

We can also base our queries on the *stability* of the components in an attractor.

Definition 29. *Let $X \in Attr(S, \rightarrow)$. A component $v \in V$ is said to be *stable* in X iff $x(v) = y(v)$ for all $x, y \in X$. Components that are not stable are called *unstable* or *oscillating* in X .*

Asking whether there is an attractor such that some components are stable or unstable is different from the previous queries because we do not have to specify a set $S[d] \subseteq S$. For the following query let $TS = (S, \rightarrow, S)$.

Observation 13. Let $U_1, U_2 \subseteq V$. There is $X \in \text{Attr}$ such that all $u \in U_1$ are stable in X and all $v \in U_2$ are unstable in X iff

$$TS \models \exists \mathbf{AG} \left(\bigwedge_{u \in U_1} \delta_u = 0 \right) \wedge \mathbf{AG} \left(\bigwedge_{v \in U_2} \mathbf{EF}(\delta_v \neq 0) \right).$$

A witness is $x \in S$ such that all attractors $X \in \text{Attr}$ that are reachable from x are stable in U_1 and unstable in U_2 .

Note that the additional operator **EF** for unstable components is required because an unstable component may temporarily satisfy $\delta_u = 0$.

A different kind of question concerns the number of attractors. Once we know that some set $T \subseteq S$ is a trap set, we are usually also interested in how many $X \in \text{Attr}$ there are such that $X \subseteq T$. A single CTL query that decides this question is in practice not feasible because it would rely on a form of state enumeration of T , which results in an exponential length query. If, however, a state $x \in X$ of an attractor $X \in \text{Attr}$ with $X \subseteq T$ is already known then we can test if it can be reached by every state in T . If the answer is positive then X must be the only attractor, otherwise there is a second one.

Observation 14. Let $T \subseteq S$ be a trap set and $x \in X$ with $X \in \text{Attr}(S, \rightarrow)$ and $X \subseteq T$. X is the only attractor in T iff

$$TS \models \forall \mathbf{EF}(x), \quad \text{for } TS = (S, \rightarrow, T).$$

Note that this query can be modified by $TS \models \forall \mathbf{EF}(x) \vee \mathbf{EF}(y)$ to query if $X, Y \in \text{Attr}$ with $x \in X$ and $y \in Y$ are the only attractors of a network.

Finding a state $x \in T$ such that $x \in X \in \text{Attr}$ and $X \subseteq T$ for a given trap set $T \subseteq S$ may theoretically be difficult. In practice, however, it seems that a random walk $x_1 \rightarrow \dots \rightarrow x_k$ in (S, \rightarrow) with initial state $x_1 \in T$ and of sufficient length, for example $k = 10|V|$, usually results in a final state $x_k \in X$ with the desired property. To be sure, model checking can be used to decide if x_k is inside an attractor:

Observation 15. Let $x \in S$. There is $X \in \text{Attr}(S, \rightarrow)$ such that $x \in X$ iff

$$TS \models \forall \mathbf{AG}(\mathbf{EF}(x)), \quad \text{for } TS = (S, \rightarrow, \{x\}).$$

Note that since there is a unique initial state $I = \{x\}$ this query is equivalent to the one that uses the existential version of " \models ".

Finally, model checking can also be used to detect *deterministic cycles*.

Definition 30. A path $x_1 \rightarrow \dots \rightarrow x_k$ in (S, \rightarrow) is a deterministic cycle iff $x_1 = x_k$ and $|\text{Succ}_{\rightarrow}(x_i)| = 1$ for all $1 \leq i \leq k$.

Usually, these cycles are associated with synchronous transition systems in which case they are also called *limit cycles*, they may however also exist in non-deterministic STGs. They can be detected by either LTL or CTL model checking with the proposition $\Delta = 1$.

Observation 16. There is a deterministic cycle in (S, \rightarrow) iff

$$TS \models \exists \mathbf{G}(\Delta = 1), \quad \text{for } TS = (S, \rightarrow, S).$$

Equivalently, any of the two CTL formulae $\mathbf{AG}(\Delta = 1)$ or $\mathbf{EG}(\Delta = 1)$ can be used. Note that this approach for detecting deterministic cycles is similar to [26] but more basic as it relies on standard model checking algorithms.

4.3 Seeds and Trap Sets

The queries of the previous section can be combined to detect the attractors of a network. We can use the queries of Obs. 9 and Obs. 10 with the initial states $S[d] = S$ to find the first $X \in \text{Attr}(S, \rightarrow)$. The returned witness either belongs to X or it can be the starting point for a sufficiently long random walk that will find a state $x \in X$. The query of Obs. 14 can then be used to decide if there is another attractor. If so, the new witness will be the starting point for the next random walk that finds $y \in Y$ with $Y \in \text{Attr}(S, \rightarrow)$ (again Obs. 14), and so on.

But, the pure model checking approach is limited to networks with about 70 components for the synchronous update and about 50 for the asynchronous and mixed updates, see Sec. 2.4.3, due to the state explosion problem. Since steady states can be efficiently computed for much larger networks (see e.g. [23, 43]) scalable methods are in particular required for determining or estimating the number of cyclic attractors including where in state space they are. In Sec. 4.8 we develop such a method. This section introduces the necessary background. The goal is to find a useful characterization for symbolic states that reference trap sets.

Definition 31. A subspace $S[p]$ for $p \in \text{Sym}$ that is a trap set is called trap space.

Compared with general trap sets, trap spaces are particularly simple in terms of their specification. An arbitrary set $T = S[d] \subseteq S$ may need a description $d \in \text{StateDesc}$ that is much larger than $|V|$ whereas a trap space $S[p]$ can always be specified by $|p| \leq |V|$ atomic propositions. In addition, finding trap spaces scales well to networks with hundreds of components, as will be remarked in Sec. 4.6. The intuition behind trap spaces is illustrated in Fig. 4.1.

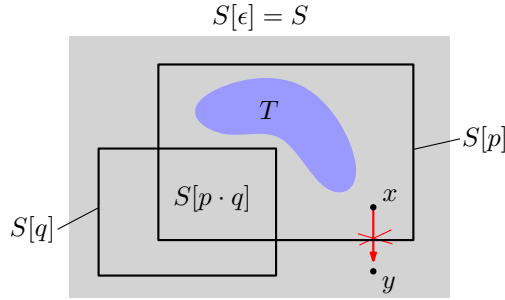


Figure 4.1: A schematic view of a state transition graph (S, \rightarrow) . A subspace $S[p]$ for $p \in \text{Sym}$ that satisfies the trap set condition (see Obs. 8), i.e., $x \not\rightarrow y$ for all $x \in S[p]$ and $y \notin S[p]$, is called a trap space. Trap spaces may contain other trap sets $T \subseteq S[p]$ whose geometry in terms of the length of a shortest $d \in \text{StateDesc}$ such that $S[d] = T$, is more complicated. Trivially $S[\epsilon] = S$ is a trap space and the intersection $S[p] \cap S[q] = S[p \cdot q]$ of two trap spaces is again a trap space.

The background begins with the notion of *symbolic steady states*, as defined in [40]. Since our work is based on [40] we stick to the terminology used therein. As the name suggests, symbolic steady states are first of all symbolic states and secondly steady - but in what sense? The idea in [40] was to extend the

notion of transitions from regular states to transitions between symbolic states by extending F from S to Sym . The requirement was that the extension $F : Sym \rightarrow Sym$ should be a true generalization, i.e., that if $x = p_x$ for $x \in S$ and $p_x \in Sym$ then the image of x and the image of p_x are also equal. Recall from Sec. 2.1.1 that $x = p_x \Leftrightarrow S[p_x] = \{x\}$.

So how is the image of a symbolic state defined? The idea in [40] was to define it in terms of the update functions $f \in F$ that are constant in $S[p]$.

Definition 32. *The image $q = F[p] \in Sym$ of a symbolic state $p \in Sym$ with respect to F is defined by*

$$(c, v) \in Set_q \quad :\Leftrightarrow \quad \forall x \in S[p] : f_v(x) = c.$$

Note that this definition is an extension in the above sense:

Observation 17. *If $p = x$ for $p \in Sym$ and $x \in S$ then $F[p] = F(x)$ holds because every $f \in F$ is trivially constant in $S[p] = \{x\}$ and equal to $f(x)$.*

With Def. 32 we can immediately define symbolic steady states as the fix-points of Sym under F .

Definition 33. *A symbolic state $p \in Sym$ that satisfies $F[p] = p$ is called a symbolic steady state. The set of all symbolic steady states is denoted by*

$$SymSteady = SymSteady(V, F) := \{p \in Sym \mid F[p] = p\}.$$

We will prove in Thm. 2 that $p \in SymSteady$ have the property we are after, namely that $S[p]$ is a trap set if $F[p] = p$. There may, however, be symbolic states that are not steady and nevertheless reference trap sets. $F[p] = p$ is therefore not a good characterization for symbolic states that reference trap sets. The one we are looking for is $F[p] \geq p$ where " \geq " is the partial order defined in Def. 5.

This property was also mentioned in [40] where symbolic states that have it were called *seeds*. We stick to this name and will for the remainder of this chapter focus on seeds rather than symbolic steady states.

Definition 34. *A symbolic state $p \in Sym$ that satisfies $F[p] \geq p$ is called a seed. The set of all seeds is denoted by*

$$Seeds = Seeds(V, F) := \{p \in Sym \mid F[p] \geq p\}.$$

The name "seed" is motivated by the observation that they can be used as starting points to finding symbolic steady states by an iterative procedure that is sometimes referred to as *percolation*. It is based on the observation that if $F[p] \geq p$ then $F^2[p] \geq F[p]$.

Proposition 2. *If $p \in Seeds$ then $F[p] \in Seeds$.*

Proof. Let $q := F[p]$. Since $q \geq p$ it follows that $S[q] \subseteq S[p]$. Therefore, every $f \in F$ that is constant in $S[p]$ is also constant in $S[q]$. Hence $F[q] \geq F[p] = q$ and $q \in Seeds$. \square

Since V is finite, there is $k \in \mathbb{N}$ such that $F^{k+1}[p] = F^k[p] \in \text{SymSteady}$. Hence, we can assign to each seed a unique symbolic steady state that is obtained by percolation. Note also that the definition of $F[p]$ is almost like a synchronous transition relation on the set of symbolic states. What is missing is the notion of a distance between symbolic states, which we could define much like the Manhattan distance of Def. 1. The asynchronous, synchronous and mixed STGs are therefore extendable to the set of symbolic states Sym . The symbolic steady states are then exactly the fixpoints of these symbolic STGs $(\text{Sym}, \rightarrow)$.

The central result in this section is that seeds correspond to trap spaces.

Theorem 2. $p \in \text{Seeds}$ if and only if $S[p]$ is a trap space in (S, \rightarrow) .

Proof. For the direction " \Rightarrow ", assume $p \in \text{Seeds}$ but $S[p]$ is not a trap set. Then there is $x \rightarrow y$ such that $x \in S[p]$ and $y \notin S[p]$. But then there is $(v, c) \in \text{Set}_p : f_v(x) \neq c$ which contradicts $p \leq x$ and $f_v(z) = c$ for all $z \in S[p]$, in particular for $z = x$.

For the direction " \Leftarrow ", assume $S[p]$ is a trap set. Then for all $x \in S[p]$ and $x \rightarrow y$ it holds that $y \in S[p]$. In particular, for all $(v, c) \in \text{Set}_p$ and all $x \in S[p]$ it holds that $f_v(x) = c$. Hence $\text{Set}_{F[p]} \supset \text{Set}_p$ and therefore $F[p] \geq p$ which implies $p \in \text{Seeds}$. \square

What we have gained by this characterization is that by solving the inequality $F[p] \geq p$ of Def. 34 for $p \in \text{Sym}$ we can find all trap spaces in (S, \rightarrow) . Since $F[p]$ is defined in terms of $f \in F$ that are constant in $S[p]$ we need to understand under which circumstances $f \in F$ becomes constant. This question will be answered in Sec. 4.5.

A somewhat surprising corollary of Def. 32 and Thm. 2 is that trap spaces (as opposed to trap sets) are identical for all update strategies that satisfy Eq. (P1) in Sec. 2.3.1. An example is given in Fig. 4.2.

Corollary 1. Let $p \in \text{Sym}$. The following statements are equivalent:

- (i) $S[p]$ is a trap set in (S, \twoheadrightarrow) .
- (ii) $S[p]$ is a trap set in (S, \hookrightarrow) .
- (iii) $S[p]$ is a trap set in (S, \rightharpoonup) .

Proof. The definition of $F[p]$ for $p \in \text{Sym}$ (Def. 32) and the proof of Thm. 2 are independent of the transition relation. \square

Note that Cor. 1 is a generalization of the observation that steady states are identical for the synchronous and asynchronous STGs, i.e., that $\text{Steady}(S, \hookrightarrow) = \text{Steady}(S, \twoheadrightarrow)$. The size $|p|$ of $p \in \text{Seeds}$ contains information regarding the location and size $|X|$ of attractors $X \subseteq S[p]$. Also, given $p \in \text{Seeds}$ we can read off the components that must be stable: every $v \in V[p]$ is stable at $\text{Map}_p(v)$ in every $X \subseteq S[p]$. But, there may be additional $u \in V$ with $u \notin V[p]$ that are also stable in some $X \subseteq S[p]$. One reason why this can happen is that trap spaces can be *nested*, i.e., one contained in another. We introduce the following notation for the extremal elements of Seeds under the partial order of Def. 5.

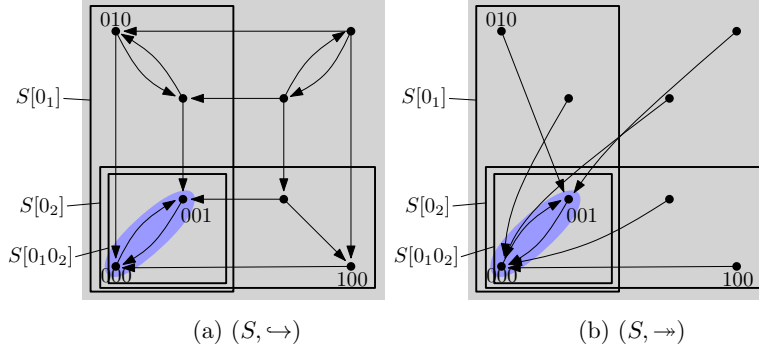


Figure 4.2: The existence of trap spaces is independent of the transition relation. Shown are the synchronous and asynchronous STGs of the same Boolean network. Each STG has the same 4 trap spaces $S[\epsilon] = S, S[0_1], S[0_2]$ and $S[0_1 0_2]$. The same holds for the mixed STG (not shown as it has too many transitions).

Definition 35. We denote by $\max(\text{Seeds})$ and $\min(\text{Seeds})$ the following sets:

$$\begin{aligned} \max(\text{Seeds}) &:= \{p \in \text{Seeds} \mid \forall q \in \text{Seeds} : p, q \text{ comparable} \Rightarrow p \geq q\} \\ \min(\text{Seeds}) &:= \{p \in \text{Seeds} \mid \forall \epsilon \neq q \in \text{Seeds} : p, q \text{ comparable} \Rightarrow p \leq q\} \end{aligned}$$

It follows that $\epsilon \in \max(\text{Seeds})$ if and only if $\text{Seeds} = \{\epsilon\}$ because ϵ is comparable with every $p \in \text{Seeds}$ and $\epsilon \leq p$. For the definition of $\min(\text{Seeds})$ we require $\epsilon \neq q$ because otherwise $\min(\text{Seeds}) = \{\epsilon\}$ (independent of Seeds) for the same reason. Note that $\epsilon \in \min(\text{Seeds})$ holds. Note also that $\min(\text{Seeds})$ reference, in terms of number of states, the *largest* trap spaces and $\max(\text{Seeds})$ reference the *smallest* trap spaces.

Finally, by Obs. 17 the set inclusion

$$\text{Steady} \subseteq \max(\text{Seeds}) \tag{4.1}$$

holds. Our method for computing $\max(\text{Seeds})$ in Sec. 4.6 is therefore also a new method for computing steady states. Note that the problem of computing $\max(\text{Seeds})$ is therefore at least as hard as computing Steady , which is known to be NP-hard (see e.g. [104]).

The next section explores the connection between seeds and cyclic attractors.

4.4 Seeds and Cyclic Attractors

If we compute all maximal seeds then we have, by Eq. 4.1, also found all steady states of a network. How about the cyclic attractors? We can derive a lower bound for their number based on the following set.

Definition 36. We denote by $\max_{\mathcal{S}}(\text{Seeds})$ all maximal seeds that are not steady states:

$$\max_{\mathcal{S}}(\text{Seeds}) := \{p \in \max(\text{Seeds}) \mid \forall x \in \text{Steady} : p \neq x\}$$

In practice, to compute $\max_S(\text{Seeds})$ from $\max(\text{Seeds})$ we simply remove all p with $|p| = |V|$. The intuition behind the following lower bound on $|\text{Cyclic}|$ is that trap sets that contain no steady states must contain cyclic attractors.

Theorem 3. *The maximal, non-regular seeds are a lower bound on the number of cyclic attractors: $|\max_S(\text{Seeds})| \leq |\text{Cyclic}|$.*

Proof. Let $p \in \max_S(\text{Seeds})$. By Thm. 2, $S[p]$ is a trap set and therefore contains an attractor $X \subseteq S[p]$. If $X = \{x\}$ then $x \in \text{Steady}$ and $p \leq x$. But since $p \neq y$ for all $y \in \text{Steady}$ it follows that $p < x$ which contradicts the maximality of p . Hence $|X| \geq 2$. \square

We continue by answering four questions that concern the number and location of cyclic attractors with respect to trap spaces. Each question is answered by an example network. Recall from Sec. 2.1.1 that we specify symbolic states $p \in \text{Sym}$ by a sequence of $|p|$ values with subscripts that indicate the respective component.

Example 1. *Is every cyclic attractor contained in some $S[p]$ with $\epsilon \neq p \in \text{Seeds}$?*

No, there may be cyclic attractors that are not contained in a non-trivial trap space.

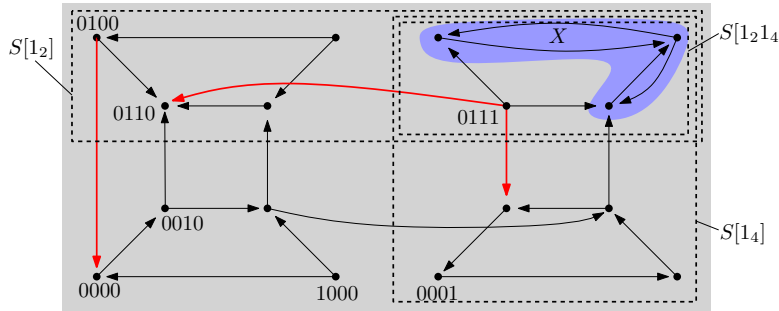


Figure 4.3: A network with a cyclic attractor not contained in a non-trivial trap space, see Ex. 1.

The network in Fig. 4.3 consists of 4 variables and there is 1 cyclic attractor X . It is a subset of exactly 3 non-trivial subspaces, namely $S[1_21_4]$, $S[1_4]$ and $S[1_2]$. But, none of them is a trap set: $0111 \leftrightarrow 0110$ violates the trap set condition of Obs. 8 for $S[1_21_4]$ and $S[1_4]$, and $0100 \leftrightarrow 0000$ for $S[1_2]$. It may therefore happen that the only trap space that a cyclic attractor is contained in is the trivial $S[\epsilon] = S$.

Example 2. *Is there a unique cyclic attractor in $S[p]$ if $p \in \text{Seeds}$?*

No, even if $p \in \max_S(\text{Seeds})$ there may be more than 1 cyclic attractor in $S[p]$. Hence the inequality of Thm. 3 may be strict.

The network in Fig. 4.4 has exactly one non-trivial seed 1_4 but $S[1_4]$ contains two cyclic attractors Y_1, Y_2 . In this case the attractors can not be separated, because $Y_1 \subseteq S[p]$ implies $Y_2 \subseteq S[p]$ for every $p \in \text{Seeds}$. In particular, the smallest subspace that contains Y_1 is $S[0_11_4]$ but $0111 \in Y_2$ and $0111 \in S[0_11_4]$. The same holds for the smallest subspace $S[1_21_4]$ that contains Y_2 because $0101 \in$

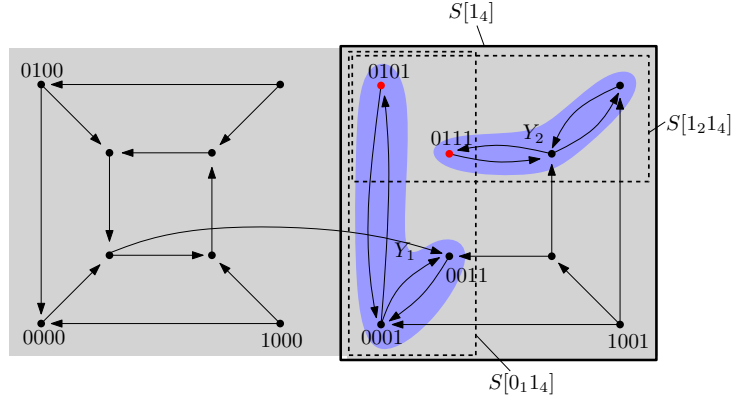


Figure 4.4: An example of cyclic attractors that can not be separated by trap spaces, see Ex. 2.

Y_1 . Hence, the number of cyclic attractors in $S[p]$ may be greater than 1 even if $p \in \max_{\mathcal{S}}(\text{Seeds})$.

The motivation for the next question is whether we can focus on $\max_{\mathcal{S}}(\text{Seeds})$ when searching for cyclic attractors (based on seeds) or whether we have to consider, potentially all of Seeds .

Example 3. If $X \in \text{Cyclic}$ is contained in $S[q]$ for $q \in \text{Seeds}$ is there always $p \in \max_{\mathcal{S}}(\text{Seeds})$ such that $X \subseteq S[p]$?

No, such p may not exist.

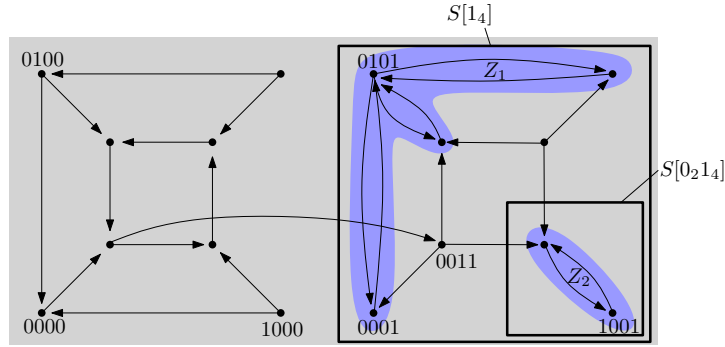


Figure 4.5: A network with a cyclic attractor that is not contained in a trap space of a maximal seed, see Ex. 3.

The network defined in Fig. 4.5 has two cyclic attractors Z_1, Z_2 and two non-trivial trap spaces $S[1_4]$ and $S[0_2 1_4]$. But since $1_4 < 0_2 1_4$ it follows that $\max_{\mathcal{S}}(\text{Seeds}) = \{0_2 1_4\}$ and hence for Z_1 there is no $p \in \max_{\mathcal{S}}(\text{Seeds})$ such that $Z_1 \subseteq S[p]$. This example also shows that the smallest trap space that contains a cyclic attractor may be referenced by neither a maximal nor a minimal seed.

The final question is motivated by the observation that if $S[p]$ is a trap space then every $v \in V[p]$ is stable in every $X \in \text{Cyclic}$ with $X \subseteq S[p]$. But is every $v \in V$ with $v \notin V[p]$ also unstable in every $X \subseteq S[p]$? At the end of Sec. 4.3

we already mentioned that this is not true for $p \in \text{Seeds}$ because they can be nested. The next example shows that it is false even when considering only maximal seeds.

Example 4. *Is every cyclic attractor $X \subseteq S[p]$ with $p \in \max_{\mathcal{S}}(\text{Seeds})$ unstable in all components $v \in V$ that satisfy $v \notin V[p]$?*

No, additional components $v \notin V[p]$ may be stable in X .

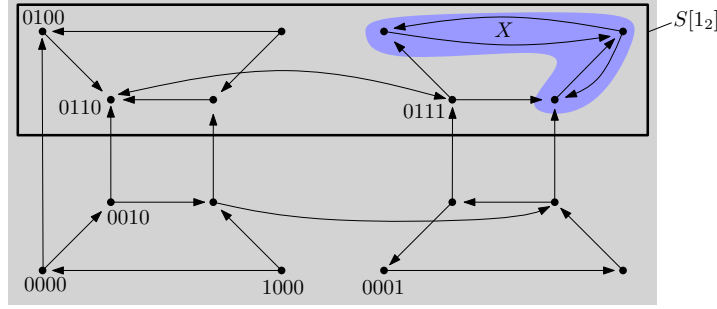


Figure 4.6: A network with a cyclic attractor X that is stable in v_4 but $v_4 \notin V[p]$ where $p = 1_2 1_4$ is the maximal seed that describes X , see Ex. 4.

The network in Fig. 4.6 has one cyclic attractor X and one non-trivial trap space $S[1_2]$. But v_4 is stable in X because $X = \{1111, 1101, 0101\}$ although $v_4 \notin V[1_1] = \{v_1\}$. The scope $V[p]$ of $p \in \text{Seeds}$ is therefore only a superset of the variables that oscillate in the attractors contained in $S[p]$.

The above examples prove that the exact number and location of the cyclic attractors can not be deduced from the seeds alone. But they can be used as the input to the CTL query patterns of Sec. 4.2. We will discuss the combined approach of computing seeds and model checking for the detection of cyclic attractors in Sec. 4.8. It will also become apparent that, in practice, seeds are often a very good description of the cyclic attractors, in particular for the asynchronous transition relation.

The next two sections address the problem of computing seeds. An important observation will be that seeds can be efficiently computed even for large networks.

4.5 The Prime Implicant Graph

Trap spaces $S[p]$ are characterized by the inequality $F[p] \geq p$, see Def. 34 and Thm. 2. Since $F[p]$ is defined in terms of those $f \in F$ that are constant in $S[p]$, see Def. 32, we will first discuss the conditions that ensure that $f \in F$ becomes constant. Minimal such conditions, called prime implicants, are represented in a directed hypergraph which we call the *prime implicant graph*. The main result in this section is Thm. 4. It proves that every seed is represented by a set of prime implicants. It is essential for our method of computing seeds in Sec. 4.6.

Minimal size implicants of Boolean functions were named *prime implicants* when they were studied by W. Quine in [105]. We stick to the terminology and define the following generalization. First, we define what an implicant is and then the condition under which implicants are prime.

Definition 37. Let $v \in V$ and $c \in \text{Dom}(v)$. A c -implicant of $f_v \in F$ is a symbolic state $p \in \text{Sym}$ such that $f_v(x) = c$ for all $x \in S[p]$.

A c -implicant p of $f \in F$ can therefore be interpreted as a conjunction of assumptions that guarantee that f becomes constant at a value c . Since we want to be economical in computing the seeds we are only interested in the implicants that make the smallest number of assumptions. The definition of prime implicants distinguishes between constant and non-constant functions. Recall that we write $f = c$ for $f \in F$ and $c \in \mathbb{N}$ to indicate that $f(x) = c$ for all $x \in S$.

Definition 38. Let $v \in V$ and $c \in \text{Dom}(v)$. A c -prime implicant of a non-constant f_v is a c -implicant $p \in \text{Sym}$ such that there is no other c -implicant $q \in \text{Sym}$ that satisfies $q < p$. For constant $f_v = c$ we define that p with $\text{Set}_p := \{(v, c)\}$ is its unique c -prime implicant and that f_v has no other prime implicants. The set of all prime implicants of a regulatory network (V, F) consists of all triplets (p, c, v) such that p is a c -prime implicant of $f_v \in F$:

$$\text{Primes} = \text{Primes}(V, F) := \{(p, c, v) \mid p \text{ is a } c\text{-prime implicant of } f_v\}.$$

Example 5. Consider a Boolean network with $V = \{v_1, v_2, v_3\}$ and $f_1 = v_1 + v_2$. So $p := 1_1 1_2$ is a 1-implicant of f_1 because $f_1(x) = 1$ for all $x \in S[1_1 1_2] = \{110, 111\}$. But, p is not prime because $q := 1_1$ is also a 1-implicant of f_1 and $q < p$. f_1 has two 1-prime implicants, namely 1_1 and 1_2 , and one 0-prime implicant, namely $0_1 0_2$.

Observation 18. The set of prime implicants of a regulatory network is finite because the set of candidate triplets (p, c, v) is finite and we can decide for every one whether it represents a prime implicant or not.

Quine was interested in prime implicants because they are useful, as an initial step, for constructing a minimal representation of a given Boolean function which in turn makes for a cost effective design of a corresponding electric circuit. Our interest is that we can use the prime implicants to find all seeds of a network. We now briefly state Quine's terminology to show that Def. 38 is really a generalization of the same idea to discrete functions.

A *clause* is a conjunction of literals, e.g. $u\bar{v}w$ for Boolean variables $u, v, w \in V$. An *implicant of a non-constant Boolean expression* is a clause such that there is no assignment to the variables that makes the clause true but the expression false. An implicant is *prime* if there is no clause of fewer literals that is also an implicant. Therefore, if we identify literals with the corresponding atomic equalities, i.e., v with $(v = 1)$ and \bar{v} with $(v = 0)$, then our 1-prime implicants of Def. 38 and Quine's prime implicants coincide for Boolean expressions.

We found that a useful representation of Primes is in the form of a directed hypergraph $(\mathcal{N}, \mathcal{A})$ in which each arc $a \in \mathcal{A}$ represents exactly one $(p, c, v) \in \text{Primes}$ and each node \mathcal{N} an atomic assumption $(v = c)$.

Definition 39. The prime implicant graph of (V, F) is the directed hypergraph $(\mathcal{N}, \mathcal{A})$ where

$$\mathcal{N} = \mathcal{N}(V) := \{p \in \text{Sym} \mid |p| = 1\}$$

consists of all size 1 symbolic states. The arcs $\mathcal{A} = \mathcal{A}(V, F) \subset 2^{\mathcal{N}} \times 2^{\mathcal{N}}$ are defined by the mapping

$$\alpha : \text{Primes} \rightarrow 2^{\mathcal{N}} \times 2^{\mathcal{N}}, (p, c, v) \mapsto (\{p_1, \dots, p_{|p|}\}, \{q\}),$$

where

- (1) $p = p_1 \cdot \dots \cdot p_{|p|}$ is the unique decomposition of p into size 1 symbolic states,
- (2) $q \in \text{Sym}$ is defined by $\text{Set}_q := \{(v, c)\}$

The prime implicant graph has one arc for every prime implicant:

$$\mathcal{A} = \mathcal{A}(V, F) := \{\alpha(p, c, v) \mid (p, c, v) \in \text{Primes}\}$$

The head of an arc $a = (\{p_1, \dots, p_{|p|}\}, \{q\})$ is denoted by $H(a) := q$, and its tail by $T(a) := p$.

An illustration of the representation of prime implicants as hyperarcs is given in Fig. 4.7.

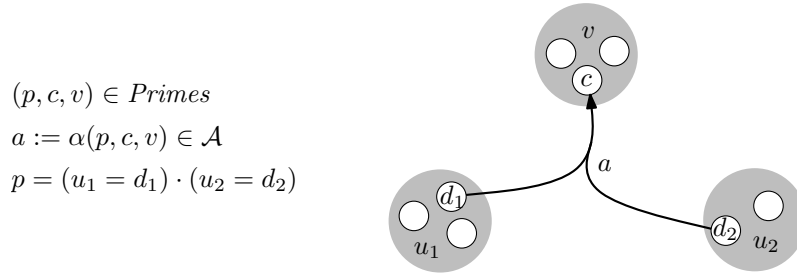


Figure 4.7: Each $(p, c, v) \in \text{Primes}$ is represented by $a = \alpha(p, c, v) \in \mathcal{A}$ in the prime implicant graph $(\mathcal{N}, \mathcal{A})$. The elements of \mathcal{N} are white discs and hyperarcs have a common arrowhead. For the layout, we place all $q \in \mathcal{N}$ with $V[q] = \{u\}$ for some $u \in V$ inside gray discs that are labeled with u .

Note that there is a one-to-one correspondence between arc sets $A \subseteq \mathcal{A}$ and sets of prime implicants $P \subseteq \text{Primes}$ and that in Boolean networks, \mathcal{N} corresponds to the literals of propositional logic and that $|\mathcal{N}|$ is polynomial in V (as opposed to $|S|$).

An illustration of a prime implicant graph is given in Ex. 6 below. Now we establish a relationship between subsets $A \subseteq \mathcal{A}$ and the seeds of a network (V, F) . To do so we need the notions of *consistency* and *stability*.

Definition 40. A subset $A \subseteq \mathcal{A}$ is consistent iff for all $a_1, a_2 \in A$ the symbolic states $H(a_1)$ and $H(a_2)$ are consistent.

If $A = \{a_1, \dots, a_m\} \subseteq \mathcal{A}$ is consistent then the conjunction $H(a_1) \cdot \dots \cdot H(a_m)$ is called the *induced symbolic state* of A and denoted by $H(A)$. For the special case $A = \emptyset$ we define $H(A) := \epsilon$.

Definition 41. A subset $A \subseteq \mathcal{A}$ is stable iff for every $a \in A$ there is $B_a \subseteq A$ such that $T(a) \leq H(B_a)$.

In this case the requirements $T(a)$ for each implication $a \in A$ to become effective are met by some assumptions $H(B_a)$. An illustration of consistency and stability is given in Fig. 4.8 and an example is given in Ex. 6.

The central idea for the computation of *Seeds* is given in the next result:

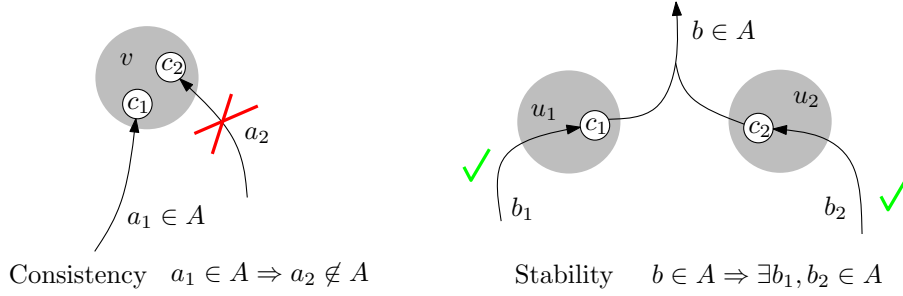


Figure 4.8: Intuitively, an arc set $A \subseteq \mathcal{A}$ is consistent if $a \in A$ that targets v at c_1 implies $a_2 \notin A$ if a_2 targets v at another value c_2 . A is stable if $b \in A$ implies that each tail end of b is targeted by another arc $b_i \in A$.

Theorem 4. $p \in \text{Seeds}$ if and only if there is a stable and consistent $A \subseteq \mathcal{A}$ such that $H(A) = p$.

Proof. The statement is true by definition for $p = \epsilon$ and $A = \emptyset$.

For the direction " \Rightarrow ", let $p \neq \epsilon$ and $(v, c) \in \text{Set}_p$.

Since $F[p] \geq p$ it follows that $f_v[p] = c$ and hence that there is a c -prime implicant q_v of f_v that satisfies $q_v \leq p$. The set $A := \{\alpha(q_v, c, v) \in \mathcal{A} \mid (v, c) \in \text{Set}_p\}$ is consistent because p is consistent. By construction it satisfies $H(A) = p$. It is also stable because $T(a) \leq p$ for all $a \in A$.

For the direction " \Leftarrow ", let $\emptyset \neq A \subseteq \mathcal{A}$ be stable and consistent. Since A is consistent, $p := H(A)$ is well-defined. Then for all $(v, c) \in \text{Set}_p$ there is $a \in A$ such that $H(a) = c$. Let $a = (q, c, v)$. Since A is stable it follows that $q \leq p$ and so $f_v(x) = c$ for all $x \in S[p]$. Hence $F[p] \geq p$ and $p \in \text{Seeds}$. \square

Corollary 2. Inclusion-wise maximal stable and consistent arc sets induce maximal seeds.

Example 6. Consider the Boolean network (V, F) with $V = \{v_1, v_2, v_3, v_4\}$ and

$$f_1 = v_1 + v_2, \quad f_2 = v_1 \cdot v_4, \quad f_3 = \overline{v_1} \cdot v_4, \quad f_4 = \overline{v_3}.$$

The prime implicant graph $(\mathcal{N}, \mathcal{A})$ and the set Primes are given in Fig. 4.9. The 6 stable and consistent arc sets are

$$\begin{aligned} A = \emptyset, \quad B = \{a_3, a_5\}, \quad C = \{a_1\}, \quad D_1 = \{a_1, a_4, a_8, a_{10}\} \\ D_2 = \{a_2, a_4, a_8, a_{10}\} \\ D_3 = \{a_1, a_2, a_4, a_8, a_{10}\} \end{aligned}$$

where $H(A) = \epsilon, H(B) = 0_1 0_2, H(C) = 1_1$ and for $i = 1, 2, 3$: $H(D_i) = 1101$ are the 4 seeds of the system:

$$\text{Seeds} = \{\epsilon, 0_1 0_2, 1_1, 1101\}, \max(\text{Seeds}) = \{1101\}, \min(\text{Seeds}) = \{1_1, 0_1 0_2\}.$$

Note that the prime implicant graph can also be seen as generalization of the forcing graph of S. Kauffman [38] and F. Fogelman [39]. A special case of seed was first studied by these authors under the name of *self-freezing circuits* whose existence relies on *canalizing effects*. Those circuits occur if, in

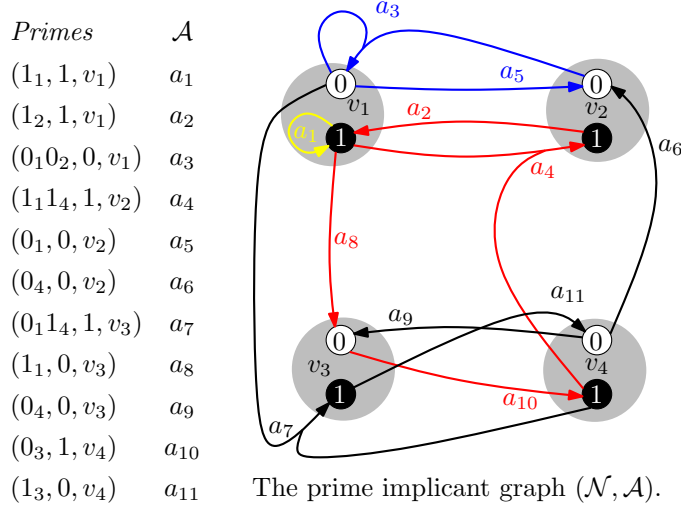


Figure 4.9: The set *Primes* and the prime implicant graph $(\mathcal{N}, \mathcal{A})$ of Ex. 6. For Boolean components we draw $q \in \mathcal{N}$ that correspond to positive literals in black and negated literals in white. The colors of the arcs indicate sets that are consistent and stable.

our terminology, there is a stable and consistent arc set $\emptyset \neq A$ that contains exclusively size 1 prime implicants, i.e., for all $a \in A$ we have $|T(a)| = 1$.

Note also that the *prime implicant graph* is an object that usually lives in between the large STG and small IG of (V, F) . For, although the number of prime implicants of even a Boolean $f \in F$ may grow exponentially with the number of variables it depends on, see e.g. [106], we found that for typical biological models the size, the number of arcs and nodes, is in between that of the STG and IG.

4.6 Computing Seeds

The previous section translated the problem of finding seeds by solving $F[p] \geq p$ into finding stable and consistent arc sets in the prime implicant graph, see Thm. 4. In this section we show that those arc sets can be computed with existing solvers for integer linear or answer set programs.

We propose an optimization-based method for finding all maximal stable and consistent arc sets in $(\mathcal{N}, \mathcal{A})$. As a preliminary step, the set *Primes* has to be computed. This can be achieved for Boolean networks with any implementation of the Quine-McCluskey algorithm, see e.g. [107]. Although for sufficiently complex expressions $f \in F$ the enumeration of *Primes* itself can be a hard problem, we found that the complexity of F in typical biological models is low enough for this step to be negligible.

For non-Boolean $f \in F$ we suggest to transform f into a Boolean function g and use the Boolean approach. Since f may depend on other non-Boolean components whose values must be transformed into additional Boolean variables that g depends on, this procedure is not likely to be very scalable or efficient.

An alternative is to modify existing algorithms for the computation of Boolean prime implicants, for example the one in [107].

We now formulate a 0-1 optimization problem to compute maximal or minimal stable and consistent arc sets $A \subseteq \mathcal{A}$ of Boolean networks. For every arc $a = (p, c, v) \in \text{Primes}$ we introduce a variable $x_a \in \{0, 1\}$ indicating whether or not a is a member of the set $A \subseteq \mathcal{A}$ that we want to compute. We denote these variables by $X := \{x_a \mid a \in \text{Primes}\}$.

In addition, we introduce a variable $y_v^c \in \{0, 1\}$ for every $v \in V$ and $c \in \text{Dom}(v)$ that indicates whether v is in the domain of the induced symbolic state and, if so, what value it takes. We denote them by $Y := \{y_v^c \mid c \in \text{Dom}(v), v \in V\}$. For any $v \in V$ and every $c \in \text{Dom}(v)$, we require $y_v^c = 1$ if and only if $(v, c) \in \text{Set}_p$ for $p := H(A)$. To encode this requirement, we use the logical constraints

$$y_v^c \iff \bigvee_{a \in B_v^c} x_a, \quad \text{for all } c \in \text{Dom}(v), v \in V. \quad (\text{C1})$$

Here, $B_v^c := \{a \in \mathcal{A} \mid \text{Set}_{H(A)} = \{(v, c)\}\}$ denotes the arcs inducing v to take the value c , and \Rightarrow and \vee are the standard logical connectives for implication and disjunction.

Next, we want to enforce the set $A := \{a \in \mathcal{A} \mid x_a = 1\}$ to be stable and consistent. To achieve this, we add the following constraints (C2) resp. (C3):

$$x_a \Rightarrow y_v^c, \quad \text{for all } a \in \mathcal{A}, (v, c) \in \text{Set}_{T(a)} \quad (\text{C2})$$

$$\overline{y_v^c} \vee \overline{y_v^d}, \quad \text{for all } v \in V, c \neq d \in \text{Dom}(v). \quad (\text{C3})$$

To find maximal stable and consistent sets $A \subseteq \mathcal{A}$, we solve the 0-1 optimization problem (here \sum denotes addition)

$$\text{maximize } \sum_{y_i^c \in Y} y_i^c, \text{ such that (C1), (C2), (C3).} \quad (0-1)$$

The above formulation can also be used to compute *Seeds* and $\min(\text{Seeds})$. To compute $p \in \text{Seeds}$ we solve the satisfiability problems (without optimization) and to find $p \in \min(\text{Seeds})$ we minimize the objective function in problem (0-1) rather than maximizing it. To solve problem (0-1) in practice, we can reformulate the constraints (C1)-(C3) as linear 0-1 inequalities or as an answer set program. The next sections present two possible encodings.

4.6.1 ILP Encoding

An *integer linear program* (ILP) belongs to the class of problems that involve integer variables, linear constraints and a linear objective function. A solution to an ILP satisfies all the constraints and optimizes the objective. ILP has previously been suggested as a method for solving problems arising in the study of Boolean networks, see e.g. [98] and references therein.

To encode the constraints as linear inequalities we use the following common translation into linear inequalities. The constraint (C1) that describes the indicator variables Y translates to (L1).

$$\begin{aligned} y_v^c &\leq \sum_{a \in B_v^c} x_a, & \text{for all } c \in \text{Dom}(v), v \in V. \\ y_v^c &\geq x_a, & \text{for all } a \in B_v^c. \end{aligned} \quad (\text{L1})$$

The stability and consistency constraints (C2), (C3) translate to (L2), (L3) respectively.

$$x_a \leq y_v^c, \quad \text{for all } a \in \mathcal{A}, (v, c) \in \text{Set}_{T(a)} \quad (\text{L2})$$

$$\sum_{c \in \text{Dom}(v)} y_v^c \leq 1, \quad \text{for all } v \in V. \quad (\text{L3})$$

All maximal seeds can be enumerated by iteratively solving problem (0-1) of the previous section, $\min(\text{Seeds})$ can be computed by minimizing (0-1). Whenever a new solution $z : X \cup Y \rightarrow \{0, 1\}$ is found, we add a so-called *no-good cut*, which prevents this solution from being computed again. For example, we can use the constraint

$$1 \leq \sum_{y_v^c \in G(z)} y_v^c, \quad \text{where } G(z) := \{y_v^c \in Y \mid z(y_v^c) = 0\}.$$

A prototype Python implementation, called `BOOLNETFIXPOINTS`, for Boolean networks using the integer programming solver `GUROBI` [108] is available [109]. Regarding the efficiency of using this encoding and `GUROBI` we remark that we have tested Boolean networks that were randomly generated by the same method as for the NuSMV benchmark in Sec. 2.4.3. Preliminary results show that the complete sets $\min(\text{Seeds})$ and $\max(\text{Seeds})$ can be computed for networks with hundreds of components within seconds to minutes depending on the size of $\text{Primes}(V, F)$. We also tried enumerating all seeds but found that $|\text{Seeds}|$ grows very quickly as $p, q \in \text{Seeds}$ implies $p \cdot q \in \text{Seeds}$.

4.6.2 ASP Encoding

Answer set programming (ASP) is a form of logic programming that is based on the so-called answer set semantics of logic programs, see [110]. Our prototype Software [109] is capable of translating a given Boolean network into a standard ASP input file that can be processed with the solver `POTASSCO` [50]. To encode the arcs \mathcal{A} we introduce two ternary predicates, `head(v, c, ID)` and the `tail(v, c, ID)`, where `v` refers to a component, `c` to an activity and `ID` is an index that determines whether a tail and a head belong to the same arc $a \in \mathcal{A}$. Each arc $a \in \mathcal{A}$ is then translated into a number of so-called *facts* by stating all the tail elements and the head element it consists of. For example, an arc $a_3 = (q, 0, v_1)$ with $\text{Set}_q = \{(v_2, 0), (v_3, 1)\}$ becomes

```
tail(v2,0,a3).
tail(v3,1,a3).
head(v1,0,a3).
```

Note that the index `a3` in each predicate links the data together. In the "generate and test" fashion of defining ASP problems we generate all possible subsets $A \subseteq \mathcal{A}$ and introduce an unary predicate `in_set(ID)` that indicates whether the arc with index `ID` belongs to the solution $A \subseteq \mathcal{A}$ or not. It encodes the variables $x_a \in X$ in the formulation of the (0-1) problem above.

$$\{\text{in_set}(\text{ID}) : \text{head}(\text{v}, \text{c}, \text{ID})\}.$$

The consistency constraint (C2) is translated into a so-called filter that forbids certain combinations of assignments to predicates. For a Boolean component v the following filter is sufficient:

`:- in_set(ID1), in_set(ID2), head(v,1,ID1), head(v,0,ID2).`

It forbids that two arcs with identifiers ID1 and ID2 that target the same component v but at different values (here 0 and 1), belong to the same solution. To encode consistency for non-Boolean components requires an additional filter for every $c_1, c_2 \in \text{Dom}(v)$ such that $c_1 \neq c_2$.

The stability constraint (C3) is translated into the filter

`:- in_set(ID1), tail(v,c,ID1), not in_set(ID2) : head(v,c,ID2).`

It forbids the existence of $a \in A$ (ID1) that has a tail $(v, c) \in \text{Set}_{T(a)}$ which is not the head of another arc $b \in A$ (ID2). Note that

`not in_set(ID2) : head(v,c,ID2)`

means "there is no $b \in A$ such that $H(b) = (v, c)$ ". So far we have not tested the efficiency and scalability of the ASP framework and this encoding but we plan to do a comparison between ASP and ILP in the future.

4.7 Seeds and Positive Feedback

This section is a short excursion to the so-called *circuit analysis* of logical networks. We restrict ourselves to Boolean networks but generalizations are possible. A *circuit* is a path $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$ in (V, \rightarrow) such that $v_i \neq v_j$ for $1 \leq i < j < k$ and $v_1 = v_k$, i.e., a circuit is a non-empty path in which the first and last nodes are identical. A circuit is *positive* if there is a sequence $s_i \in \{+, -\}$ of signs for $1 \leq i < k$ such that $s_i \in \text{Sign}(v_i \rightarrow v_{i+1})$ and $|\{1 \leq j < k \mid s_j = -\}|$ is even. A circuit is *negative* if there is a sequence of signs such that $|\{1 \leq j < k \mid s_j = -\}|$ is odd. Note that since interactions can be activating and inhibiting, i.e., $\text{Sign}(u \rightarrow v) = \{+, -\}$, circuits can be both positive and negative.

R. Thomas' conjectured the following relationship between circuits and the asymptotic behaviors of a network:

- (1) If $|\text{Attr}(S, \hookrightarrow)| \geq 2$ then there is a positive circuit in (V, \rightarrow) .
- (2) If $|\text{Cyclic}(S, \hookrightarrow)| \geq 1$ then there is a negative circuit in (V, \rightarrow) .

Both have since been proved. For a discussion about the conjectures and formal proofs see [33, 34] and references therein. The interest in proving statements that relate the interaction graph and the state transition graph, i.e., to discover commonalities in the dynamics of all models with the same IG, is motivated by the need to predict the behavior of a model without knowing all of its logical parameters, i.e., the exact target values of each $f \in F$. Note that Thomas' theorems are rarely restrictive in real biological case studies since practically any model contains a positive and negative circuit in the IG. Practically any model is therefore, in theory, capable of producing *multistability* (several attractors) and *sustained oscillations* (cyclic attractors).

The central result in this section is Thm. 5 which proves that seeds require positive circuits in the interaction graph (V, \rightarrow) . The proof is based on the following lemma which states that prime implicants induce signed interactions. Recall that $u \xrightarrow{+} v$ if there is a state $x \in S[u = 0]$ such that $0 = f_v(x) < f_v(x \oplus e_u) = 1$, see Sec. 2.2.

Lemma 1. *Let (V, F) be a constant-free Boolean network. For each $(p, c, v) \in \text{Primes}$ and each $(u, d) \in \text{Set}_p$ the following implications hold:*

$$\begin{aligned} d = c &\Rightarrow u \xrightarrow{+} v \\ d \neq c &\Rightarrow u \xrightarrow{-} v \end{aligned}$$

Proof. There are four different combinations of values for c and d . We prove the case $c = 1$ and $d = 1$, the others are derived similarly. To illustrate the claim, consider the hyperarc representation in Fig. 4.10. Let

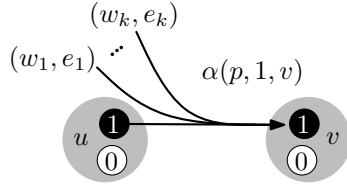


Figure 4.10: The arc representation of $(p, c, v) \in \text{Primes}$ with $c = 1$ and $d = 1$.

$\text{Set}_p = \{(u, 1), (w_1, e_1), \dots, (w_k, e_k)\}$ and define the symbolic state r by $\text{Set}_r := \{(u, 0), (w_1, e_1), \dots, (w_k, e_k)\}$. We claim that r can not be a 1-implicant of f_v . For a contradiction, assume it is.

If $|p| = 1$ then $S[p] \cup S[r] = S[u = 1] \cup S[u = 0] = S$ and so $f_v(x) = 1$ for all $x \in S$, but F is constant-free.

If $|p| \geq 2$ then consider $z \in \text{Sym}$ defined by $\text{Set}_z := \{(w_1, e_1), \dots, (w_k, e_k)\}$. Since $S[z] = S[p] \cup S[r]$ and since $|z| \geq 1$ it follows that z is also a 1-implicant of f_v . But $z < p$ contradicts the minimality of p . Hence r is not a 1-implicant and so there is $x \in S[r]$ such that $f_v(x) = 0$. But since $y := x \oplus e_u$ satisfies $y \in S[p]$ and therefore $f_v(y) = 1$ we get

$$0 = f_v(x) < f_v(x \oplus e_u) = f_v(y) = 1$$

and therefore the interaction $u \xrightarrow{+} v$. □

It therefore makes sense to speak of the subgraph of (V, \rightarrow) that is induced by a set of prime implicants $P \subseteq \text{Primes}$. It is obtained by applying Lem. 1 to each $p \in P$. The following theorem is the main result of this section. It states that stable and consistent arc sets $A \subseteq \mathcal{A}$ induce positive circuits in the IG. Intuitively, the existence of a circuit follows from the stability of A : for each implicant there are other implicants that stabilize it and hence, by Lem. 1, for each $u \rightarrow v$ there is a $w \in V$ s.t. $w \rightarrow u$. The positiveness of all such circuits is a consequence of the consistency of A : an odd number of changes in the implications would introduce inconsistencies.

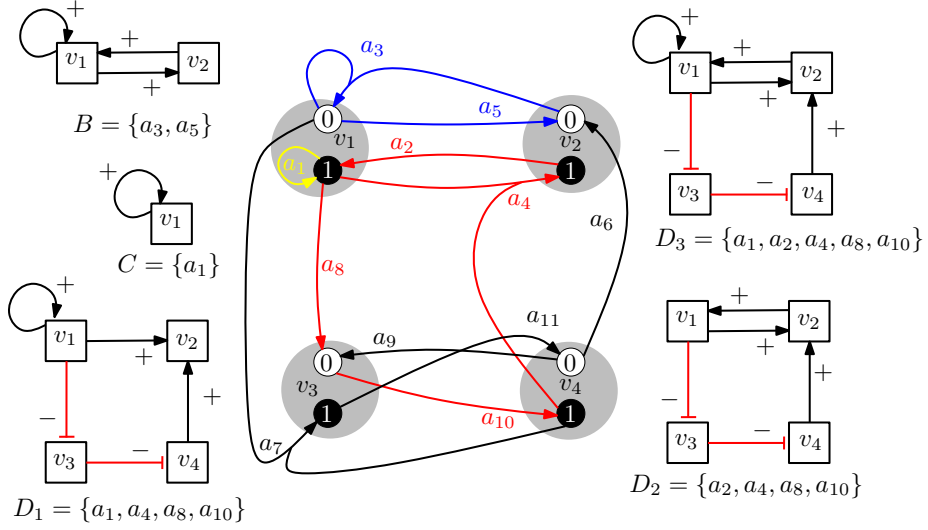


Figure 4.11: The prime implicant graph of Ex. 6, its stable and consistent arc sets B, C, D_1, D_2, D_3 and the induced interaction graphs. Note that each IG contains a circuit and that all circuits are positive.

Theorem 5. Let (V, F) be a constant-free Boolean network and $A \subseteq \mathcal{A}$ with $A \neq \emptyset$ a stable and consistent arc set. The interaction graph that is induced by the corresponding prime implicants must contain a circuit and every such circuit is positive.

Proof. Let $P \subseteq \text{Primes}$ be the prime implicants that correspond to A via the bijection α given in Def. 39. P is non-empty because $A \neq \emptyset$.

Existence of a circuit: By Lem. 1 each $(p, c, v) \in P$ induces $|p|$ interactions and because A is stable it holds that for each induced interaction $u \rightarrow v$ there is $(q, d, u) \in P$ that induces an interaction $w \rightarrow u$ for some $w \in V[q]$. Since the overall number of induced interactions is finite and every interaction has this property there must be a circuit in the IG.

Positiveness: Let $v_1 \xrightarrow{s_1} v_2 \dots v_k \xrightarrow{s_k} v_1$ be a circuit where each $v_i \xrightarrow{s_i} v_{i+1}$ is induced by $(p_i, c_i, v_{i+1}) \in P$. If $|\{1 \leq j \leq k \mid s_j = -\}|$ is odd then, according to the condition for the signs in Lem. 1, the number of times that $c_i \neq \text{Map}_{p_i}(v_i)$ is also odd. But then there must be (after possibly renaming the indices) a sequence of interactions $v_i \xrightarrow{s_i} v_{i+1} \xrightarrow{s_{i+1}} v_{i+2}$ and corresponding primes $(p_i, c_i, v_{i+1}), (p_{i+1}, c_{i+1}, v_{i+2}) \in P$ such that $c_i \neq \text{Map}_{p_{i+1}}(v_{i+1}) =: d$. Since P is stable there must be $(q, d, v_{i+1}) \in P$. But then P is not consistent because $(p_i, c_i, v_{i+1}), (q, d, v_{i+1}) \in P$ with $c_i \neq d$, a contradiction. Hence $|\{1 \leq j \leq k \mid s_j = +\}|$ is even and the circuit positive. \square

The theorem is illustrated for the network of Ex. 6 in Fig. 4.11.

By the correspondence between *Seeds* and stable and consistent arc sets we get the following corollary.

Corollary 3. Let (V, F) be a constant-free Boolean network. If there is a non-empty $p \in \text{Seeds}$ then there must be a positive circuit in (V, \rightarrow) .

Proof. By Thm. 4 there is a stable and consistent arc set $A \neq \emptyset$ such that $H(A) = p$ for such $p \in \text{Seeds}$. The statement follows by applying Thm. 5 to A . \square

Since steady states are seeds, a special case of Cor. 3 is that the existence of a positive circuits is necessary for $|\text{Steady}| \geq 1$ in constant-free Boolean networks. Note that a constant-free network can be obtained from a network with constants by percolating $p \in \text{Sym}$ defined by $\text{Set}_p := \{(v, c) \mid f_v = c\}$, see Prop. 2.

Another way of interpreting Thm. 5 is that each $s \in \text{Seeds}$ requires a positive circuit whose components are a subset of $V[s]$. This can be exploited when computing inclusion-wise minimal seeds.

Corollary 4. *If $p \in \min(\text{Seeds})$ then there is a $U \in \text{SCC}(V, \rightarrow)$ such that $V[p] \subseteq U$ and U contains at least one positive circuit.*

A good pre-processing for finding minimal seeds is therefore to enumerate all $U \in \text{SCC}(V, \rightarrow)$ that contain at least one positive circuit and then to search for stable and consistent arc sets in the respective prime implicant subgraphs.

Since seeds, which are associated with steady states and positive feedback, are characterized by stable and consistent arc sets, it is natural to ask if cyclic attractors and negative feedback are also associated to subsets $A \subseteq \mathcal{A}$ with similar properties. For networks whose interaction graph consists of an isolated circuit this is the case. Here, the existence of cyclic attractors and negative feedback correlates with *inconsistent and stable* arc sets, see Fig. 4.12. But, for networks whose IGs are not isolated circuits the connection between inconsistent and / or stable arc sets and cyclic attractors remains an open problem.

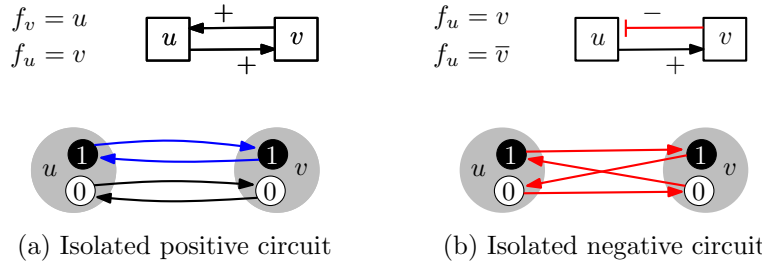


Figure 4.12: The network in (a) has 2 stable and consistent arc sets and 2 steady states. The network in (b) has 1 *stable and inconsistent* arc set and 1 cyclic attractor. This observation holds for isolated circuits of arbitrary size.

4.8 Completeness and Univocality

Let us briefly summarize how seeds are useful to predict the asymptotics of a network. First, our method for computing seeds scales to networks with hundreds of components and finds all minimal or maximal seeds within seconds or minutes (for randomly generated networks, data not shown). By Eq. 4.1 the maximal seeds include all steady states of the system, and by Thm. 3 the number of maximal non-regular seeds is a lower bound to the number of cyclic attractors.

The examples in Sec. 4.4 demonstrate that the lower bound may in general be weak in the sense that $|Cyclic(S, \rightarrow)|$ can be much larger than $|\max_S(Seeds)|$. In practice, however, $\max(Seeds)$ is often a very good description of $Attr(S, \rightarrow)$ with $|\max(Seeds)| = |Attr(S, \rightarrow)|$ and a unique attractor X in every $S[p]$ for $p \in \max(Seeds)$. We capture the properties of "a good description" in the following definitions.

Definition 42. A subset $P \subseteq Seeds$ such that for every $X \in Attr(S, \rightarrow)$ there is $p \in P$ with $X \subseteq S[p]$ is called *complete* in (S, \rightarrow) .

Definition 43. $p \in Seeds$ such that $|\{X \in Attr(S, \rightarrow) \mid X \subseteq S[p]\}| = 1$ is called *univocal* in (S, \rightarrow) .

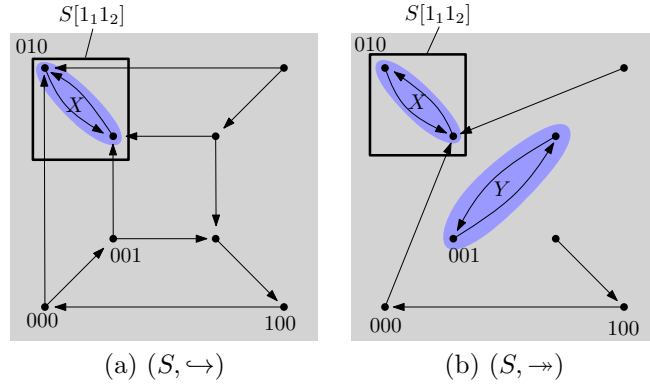


Figure 4.13: The synchronous and asynchronous STGs of the same network. The set $P := \{1_1 1_2\} \subseteq Seeds$ is complete in (S, \leftrightarrow) because $X \subseteq S[1_1 1_2]$ but not complete in (S, \Rightarrow) because the cyclic attractor Y is not contained in $S[1_1 1_2]$. Note also that $\epsilon \in Seeds$ is univocal in (S, \leftrightarrow) but not univocal in (S, \Rightarrow) .

Note that both definitions depend on the transition relation \rightarrow . A seed p may be univocal in (S, \leftrightarrow) but not univocal in (S, \Rightarrow) , see the example in Fig. 4.13. Note also that even if $p \in Seeds$ is univocal it may be that $S[p]$ is not the smallest subspace that contains the respective attractor $X \subseteq S[p]$, see Ex. 4.

We can use CTL model checking to decide whether $P \subseteq Seeds$ is complete, whether $p \in P$ is univocal and also whether $S[p]$ is the smallest subspace containing an attractor X . To test whether $S[p]$ is the smallest subspace we use the query in Obs. 13. Completeness is decided by the following slight modification of the query in Obs. 14.

Observation 19. Let $P \subseteq Seeds$ and $TS = (S, \rightarrow, S)$. P is complete in (S, \rightarrow) if and only if

$$TS \models \bigvee_{p \in P} \mathbf{EF}(p).$$

To decide whether $p \in Seeds$ is univocal we first observe that if $p = x$ with $x \in Steady$ then p is trivially univocal. Otherwise, we generate an initial state $x \in X$ such that $X \subseteq S[p]$ and $X \in Attr(S, \rightarrow)$ by a random walk, see Sec. 4.2 and Obs. 15, and then use the query of Obs. 14 to decide whether p is univocal.

If a set $P \subseteq \text{Sym}$ is not complete, or if $p \in \text{Sym}$ is not univocal then the counterexamples can be used as starting points for further analysis, e.g. in the latter case detecting an attractor outside of the subspaces referenced by P and in the former detecting a second attractor inside $S[p]$.

If the transition system of the network in question is manageable by the model checking software we can therefore use the above approach. If it is too large¹ we may still be able to decide completeness and univocality by a suitable *model reduction* and modified CTL queries. The goal of the remainder of this section is to introduce two basic reduction methods that will be crucial for the case study in Sec. 4.9. For more involved reduction methods, see for example [30, 31, 60] and references therein.

4.8.1 Model Reduction 1: Substitution

The motivation for this reduction method is that if we are interested in the trajectories inside a trap space $S[p]$ then we can remove all components that are stable in $S[p]$ from the system and consider the projected trajectories in the reduced network, see e.g. [40]. Its definition requires the notion of *substituting* p into a function $f \in F$.

Definition 44. Let $f \in F$ and $p \in \text{Sym}$. The function $f[p]$ is defined by $f[p] : U \rightarrow \text{Dom}(v)$ with $U := S(V \setminus V[p])$ and

$$f[p](y) := f(p \cdot y), \quad \forall y \in S(U),$$

where $p \cdot y := x \in S(V)$ is well-defined because $V[p] \cup V[y] = V$ and $V[p]$ and $V[y]$ are disjoint. We say that $f[p]$ is obtained from f by substitution of p .

Intuitively speaking, the network (V_p, F_p) is then obtained from (V, F) by "dividing out" the seed p :

Definition 45. The network (V_p, F_p) that is obtained from (V, F) by substitution of $p \in \text{Seeds}(F, V)$ is defined by

$$\begin{aligned} V_p &:= \{v \in V \mid v \notin V[p]\} \\ F_p &:= \{f_v[p] \mid f_v \in F, v \in V_p\}. \end{aligned}$$

There is then a one-to-one correspondence between transitions in $S[p]$ of the original network (V, F) and transitions in the reduced network (V_p, F_p) , see [40] for proofs. Relevant for our purposes is that $p \in \text{Seeds}(V, F)$ is therefore univocal in the STG of the (V, F) if and only if $\epsilon \in \text{Seeds}(V_p, F_p)$ is univocal in the STG of (V_p, F_p) . This reduction method is in practice useful because (V, F) may be too large for model checking while (V_p, F_p) may be feasible, if $|p|$ is large enough. An example of model reduction by substitution is given in Fig. 4.14.

4.8.2 Model Reduction 2: Input Percolations

The second reduction method concerns only networks with stable input components, see Sec. 2.2.1. The idea is that we can immediately derive a set of

¹About 50 components for non-deterministic and 70 components for deterministic STGs, see the NuSMV benchmark in Sec. 2.4.3

complete seeds by fixing each input to some value and then percolating the combined effect. Recall that for every $p \in \text{Seeds}$ there is a unique symbolic steady state $q \in \text{Seeds}$ that is obtained from p by percolation, see Sec. 4.3. Proofs are given in [40].

Theorem 6. *Let (V, F) be a network with stable input components v_1, \dots, v_k and $k \geq 1$. Define for each combination $(c_1, \dots, c_k) \in \text{Dom}(v_1) \times \dots \times \text{Dom}(v_k)$ a corresponding $p \in \text{Sym}$ by $p := (v_1 = c_1) \cdot \dots \cdot (v_k = c_k)$. Then $p \in \text{Seeds}$ for each such p and the set of all such p is complete in (S, \rightarrow) .*

Intuitively, this theorem states that the inputs partition the state transition graph into trap spaces defined by the values of the inputs. The following theorem states that replacing each seed of a complete set by the corresponding percolated symbolic steady state results again in a complete set.

Theorem 7. *If $\{p_1, \dots, p_m\} \subseteq \text{Seeds}$ is complete in (S, \rightarrow) then $\{q_1, \dots, q_m\} \subseteq \text{Seeds}$, where q_i is the symbolic steady state obtained from p_i by percolation, is also complete in (S, \rightarrow) .*

Note that the above statement is independent of the transition relation.

In practice, this reduction method is combined with the reduction by substitution in the following way. Let P be the set of $p \in \text{Seeds}$ obtained by percolating the different input combinations. Since P is complete and since there is a one-to-one correspondence between the transitions and therefore the attractors between (V, F) in $S[p]$ and the reduced network (V_p, F_p) , see [40], we can consider each network (V_p, F_p) separately. An example is given in Fig. 4.14.

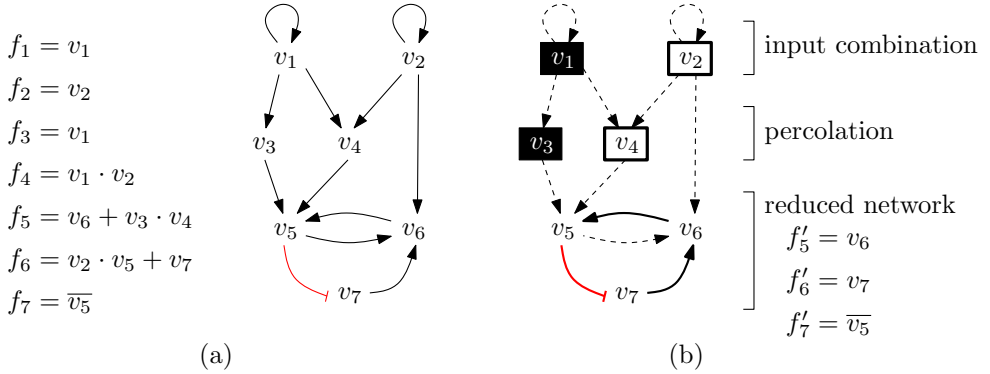


Figure 4.14: An example for model reduction by input percolation and substitution. (a) A network (V, F) with inputs $v_1, v_2 \in V$. (b) The reduced network (V_p, F_p) with $V_p = \{v_5, v_6, v_7\}$ and $F_p = \{f'_5, f'_6, f'_7\}$ where $p = (v_1 = 1)(v_2 = 0)(v_3 = 1)(v_4 = 0)$ is obtained by fixing the inputs to $v_1 = 1$ and $v_2 = 0$ and percolation. Note that the activities of p are represented by black (active) and white (inactive) boxes in the interaction graph. Dashed interactions are not observable in the reduced network.

4.9 Case Study: A MAPK Signaling Network

The aim of this section is to describe the asymptotic behaviors of the *Mitogen-Activated Protein Kinase* (MAPK) network published in [11]. The network is

a generic response map for the influence of different stimuli involving growth factors and DNA damage on processes such as cell proliferation, apoptosis and cell differentiation. Its components represent signaling proteins, genes and phenomenological components. It is stated that

”The results of systematic simulations for different signal combinations and network perturbations were found globally coherent with published data.”

We add to these results a complete description of $Attr(S, \hookrightarrow)$ for the unperturbed network by a combined approach of computing seeds and CTL model checking. For model checking we used NUSMV 2.5.4 and the seeds were computed with BOOLNETFIXPOINTS [109]. All computations were done with a Linux desktop PC with 30 GB RAM and 8 CPUs @ 3.00GHz.

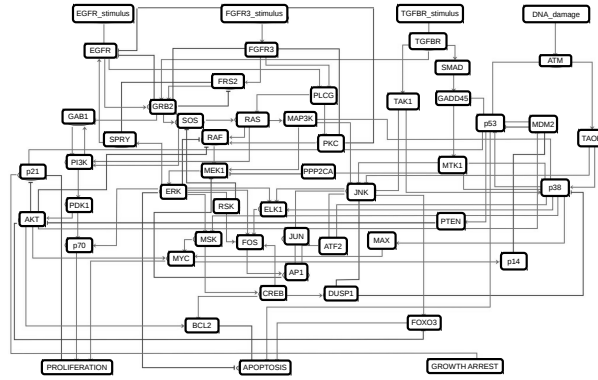


Figure 4.15: The original interaction graph of the *MAPK* network as given in [11].

Network Size and Structure

The network consists of 53 Boolean components and 108 interactions. The original interaction graph is given in Fig. 4.15 and the update functions are specified in [11]. There are 4 input components *DNAdamage*, *EGFRstim*, *FGFR3stim* and *TGFBRstim* whose activities represent different stimuli, and 3 output components *Apoptosis*, *GrowthArrest* and *Proliferation* that represent the response in terms of proliferation or cell death. The remaining components form one large SCC (37 components) and several cascades (9 components) that connect the inputs to the SCC and the outputs. The condensation of the interaction graph is given in Fig. 4.16.

First, we computed $\max(\text{Seeds})$ (within seconds) and found that $|\text{Steady}| = 12$ in agreement with [11] and $|\max_S(\text{Seeds})| = 6$. Hence, by Thm. 3, we immediately deduce $|\text{Cyclic}(S, \rightarrow)| \geq 6$ and each $S[p]$ with $p \in \max_S(\text{Seeds})$ contains at least 1 cyclic attractor.

Next, we asked whether $\max(\text{Seeds})$ is complete in (S, \rightarrow) (for any \rightarrow) but found that NUSMV runs out of memory for the query of Obs. 19. Since the net-

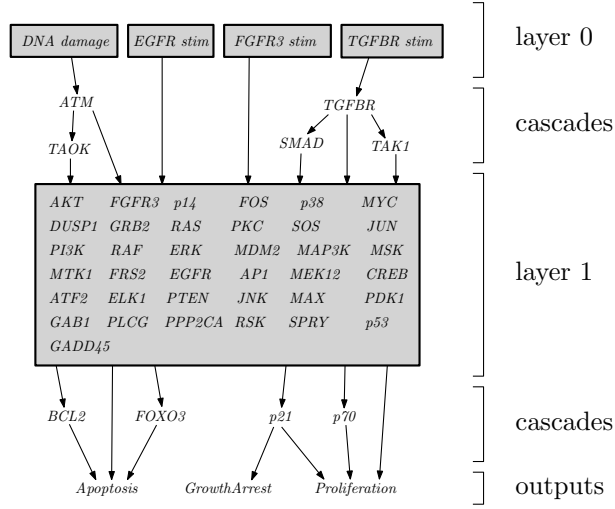


Figure 4.16: The condensation graph of the *MAPK* network consists of the 4 input components in layer 0 and one SCC with 37 components in layer 1. The remaining components connect the inputs to the SCC and the SCC to the output components.

work has input components we decided to continue with the reduction method described in Sec. 4.8.2. The 4 inputs give rise to $2^4 = 16$ different input combinations and 16 corresponding reduced networks obtained by percolating the input values. Note that since the number of maximal seeds is greater than the number of input combinations we can immediately deduce that there must be an input with several asymptotic behaviors. We then computed for each input combination the symbolic steady state that is obtained by percolation, see also Sec. 4.8.2. The corresponding reduced networks are just small enough (number of components) to be manageable with NUSMV. The goal for the remainder of this section is to present a case-by-case analysis of each reduced network that correspond to one of the 16 different input combinations. We grouped the results into 4 different classes that lead to arguably similar behaviors. The section is concluded with a short discussion.

Input Type 1: Fast Convergence

This type of stimulus is characterized by $TGFBRstim = 1$ and consists therefore of 8 different input combinations. In each case the percolation resulted in $p \in Seeds$ with $|p| = 53$, i.e., in a steady state. We checked how well these steady states agree with each other and found that they are largely identical. Hence we tested the effect of $TGFBRstim = 1$ without any assumptions on the other inputs and found that almost all components stabilize, see Fig. 4.17. Note that this is not necessarily so. As stated in Sec.4.8.2, components that stabilize due to percolation do so independently of the update strategy.

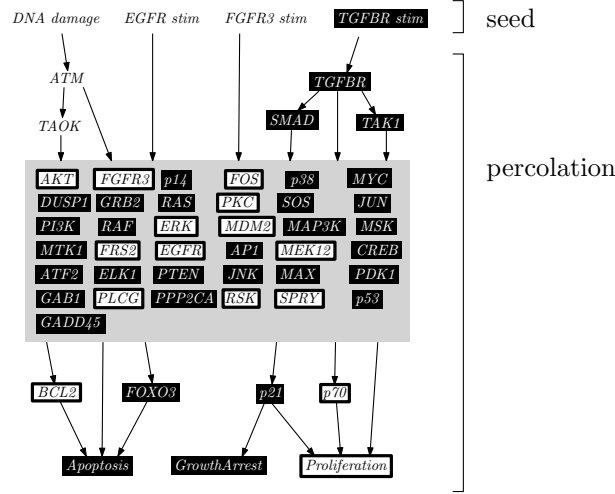


Figure 4.17: $TGFBRstim = 1$ alone drives almost all components including the outputs into stable values (black=1, white=0). The 8 steady states that correspond to this type of input differ only in the other inputs and *ATM* and *TAOK*.

Type 2: Large Cyclic Attractors

This type of stimulus is characterized by the absence of $TGFBRstim$, the absence of $DNA damage$ and the presence of $EGFRstim$ or $FGFRstim$ and consists therefore of 3 cases.

For each of the 3 cases we computed the symbolic steady state $p \in Seeds$ that corresponds to the input percolation and the maximal seeds of the respective reduced model (V_p, F_p) . We found that each reduced model has a single maximal seed $q \in Seeds(V_p, F_p)$ which was confirmed to be complete in (S, \hookrightarrow) of (V_p, F_p) by model checking. The condensation graphs (see Def. 11) of the 3 reduced models are shown in Fig. 4.18. In each case we generated a random walk of length 200 and confirmed that the resulting state belongs to some $X \in Cyclic(S, \hookrightarrow)$. We could also confirm that X is the only attractor and that every component of the reduced network oscillates in X . We grouped these stimuli because almost all components of the original network, namely 39, 41 and 41, oscillate in the resulting attractors, including the three output components *Apoptosis*, *GrowthArrest* and *Proliferation*.

Type 3: Small Cyclic Attractors

This type of stimulus consists of 3 cases and is characterized by the absence of $TGFBRstim$, the presence of $DNA damage$ and the presence of $EGFRstim$ or $FGFRstim$. The reduced networks consist of only 8, 9 and 10 components and their interaction graphs are given in Fig. 4.19. Each network has a single cyclic attractor in (S, \hookrightarrow) in which every component is unstable. We grouped these stimuli because only 8, 9 and 10 components (out of 53) oscillate in the corresponding cyclic attractors of the original network and the output components are identically stable at $Apoptosis = 1$, $GrowthArrest = 1$ and $Proliferation = 0$.

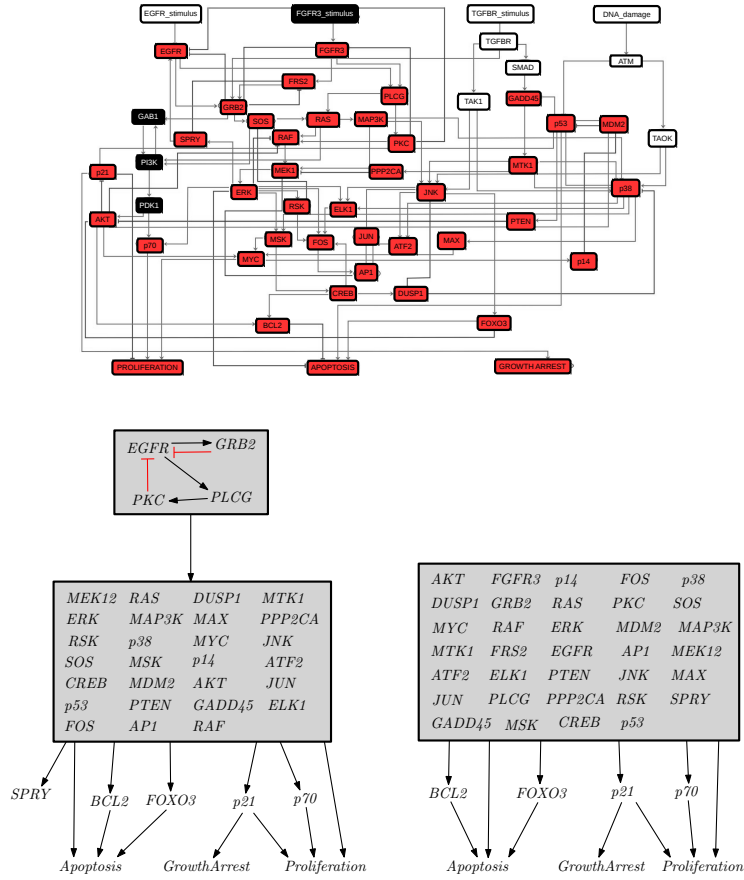


Figure 4.18: The cyclic attractors associated with inputs of type 2 are largely identical and characterized by many oscillating components, including the three outputs. The interaction graph on the top shows the oscillating components in red. At the bottom are the condensation graphs of the corresponding reduced models, the one on the right occurred twice.

Type 4: Internal Decision

This type of stimulus is characterized by the absence of *TGFBRstim*, *EGFRstim* and *FGFRstim* and consists therefore of 2 cases. Each of the 2 reduced networks obtained from percolating the inputs consists of one SCC with several output cascades. We computed the maximal seeds in each case and found that there are 2 steady states in each reduced network accounting for 4 steady states of the unreduced network. The models are just small enough for NUSMV to confirm that there are no cyclic attractors in (S, \leftrightarrow) .

We have grouped these input combinations because in both cases the decision for which of the two steady states is reached is due to the internal state of the network. In other words, the identical signal can result in different steady states. We compared the activities that define the two possible steady states and found that they are identical with respect to the output components.

The Synchronous Update

We have repeated the above analysis for the synchronous transition relation. As mentioned in Fig. 4.13, although the synchronous and asynchronous STGs have the same trap spaces, whether $P \subseteq \text{Seeds}$ is complete and whether $p \in \text{Seeds}$ is univocal may differ between (S, \leftrightarrow) and (S, \Rightarrow) . Indeed, we found that there are 4 input combinations for which the corresponding maximal seeds are not complete. The input combination in which every stimulus and *DNAdamage* are absent (type 4) has, for example, 14 additional cyclic attractors in (S, \Rightarrow) whereas (S, \leftrightarrow) has only the 2 steady states mentioned above. For a table of the number of additional cyclic attractors see Fig. 4.20. Note that the observation that synchronous STGs often have many more cyclic attractor than asynchronous, more realistic STGs is not new (see e.g. [111]). One of the reasons why (S, \Rightarrow) is often said to be less realistic is because it contains so many "spurious limit cycles".

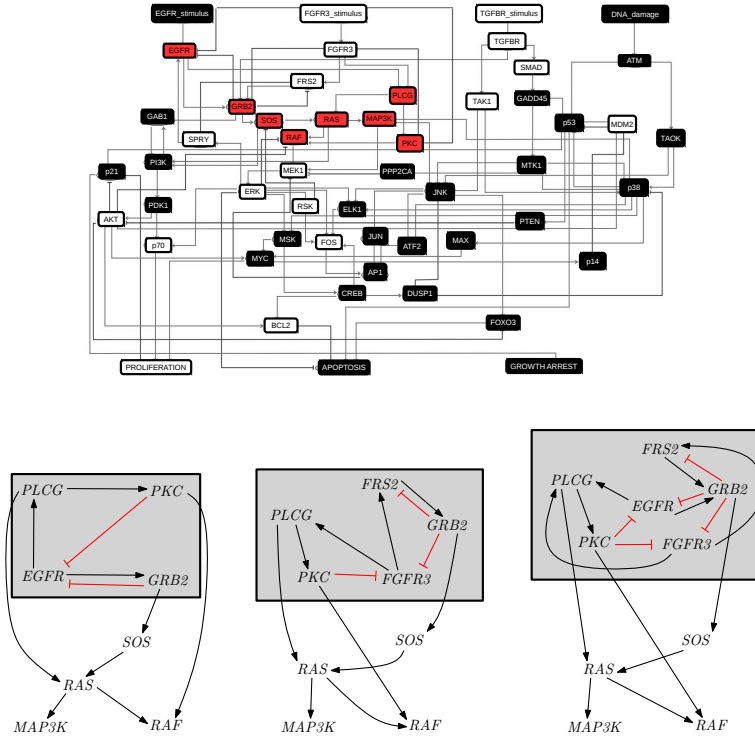


Figure 4.19: The cyclic attractors that exist for input combinations of type 3 are characterized by few oscillating components, illustrated in red in the interaction graph on top. The interaction graphs of the 3 reduced models are given below. Note that they involve mostly the same components and negative feedback circuits.

Input-Output Mapping

The main result of this case study is that we could confirm for a large network that the maximal seeds are a good description of the asymptotic behaviors of asynchronous STGs. We found that $\max(\text{Seeds}(V, F))$ is complete in (S, \hookrightarrow) and each $p \in \max(\text{Seeds}(V, F))$ univocal. Furthermore the unstable components in each $X \in \text{Cyclic}(S, \hookrightarrow)$ are exactly the components $V \setminus V[p]$, i.e., each $p \in \max(\text{Seeds})$ is the smallest possible subspace that contains the cyclic attractor in $S[p]$.

For most input combinations there is exactly 1 corresponding attractor, except when all growth factors are absent. In that case there are two steady states for $DNADamage = 1$ and two for $DNADamage = 0$. Nevertheless, if only the values of the output components are considered, the *MAPK* network behaves like an input-output network in the sense that the values of the input components fully determine the asymptotic values of the output components.

Input combination	Steady states	$ \text{Cyclic}(S, \hookrightarrow) $	$ \text{Cyclic}(S, \twoheadrightarrow) $
0010 (type 2)	0	1	6
1010 (type 3)	0	1	3
1000 (type 4)	2	0	1
0000 (type 4)	2	0	14

Figure 4.20: The 4 input combinations for which there are additional cyclic attractors in the synchronous STG. The inputs are specified in the following order: *DNADamage*, *EGFRstim*, *FGFR3stim* and *TGFBRstim*. The additional limit cycles of (S, \twoheadrightarrow) consist of between 2 and 12 states.

4.10 Discussion

The relevance of trap spaces in predicting the asymptotics of a model is that they can be computed efficiently for large networks (similar to the computation of steady states) and have implications on the existence of cyclic attractors independently of the chosen update strategy. In practice, a combination of computing maximal seeds and CTL model checking to confirm that every attractor is contained in one of the respective trap spaces (completeness) and that every trap space contains a unique attractor (univocality) seems promising. To deal with larger networks we described a reduction method by substitution to decide univocality and a reduction method by percolating input values to decide completeness (both in Sec. 4.8). A generalization of the second method to networks that do not have inputs is possible. One way is to exploit the layers in the condensation of the interaction graph of a model (V, F) (see Def. 11) by removing all components $U \in \text{SCC}(V, \rightarrow)$ with $\text{layer}(U) > k \geq 0$ and considering the asymptotics of the remaining network (V^k, F^k) . In this case one can prove that a necessary condition for the completeness of $\max(\text{Seeds}(V, F))$ in $(S(V), \rightarrow)$ is the completeness of $\max(\text{Seeds}(V^k, F^k))$ in $(S(V^k), \rightarrow)$. This reduction by removing "passenger components", i.e., downstream components with no feedback to the upper layers, is of course only possible if there are different layers in the condensation. Networks whose interaction graph consists of a single strongly connected components appear to pose the hardest problem in

terms of asymptotic behaviors. We expect that other reduction methods can be shown to conserve completeness and univocality, for example [60], and that the asymptotics of even larger networks can efficiently be understood.

A topic that we are currently exploring is the structure of maximal seeds in terms of minimal seeds. For every $p \in \max(\text{Seeds}(V, F))$ we can list all $q \in \min(\text{Seeds}(V, F))$ that satisfy $q \leq p$. The symbolic steady state $r \in \text{Seeds}$ that is obtained by percolating the conjunction of all such q must also satisfy $r \leq p$. Depending on whether $r = p$ we repeat this decomposition (for the reduced model (V_r, F_r)) until all components $V[p]$ belong to some minimal seed or are stabilized by percolation. The positive feedback circuits (see Sec. 4.7) that correspond to the resulting sequence of minimal seeds form a hierarchy. This hierarchy might be interpreted as a sequence of stabilization events that lead to the trap space $S[p]$. We believe that this might be relevant to questions regarding the control of regulatory networks. To force the system into $S[p]$ would for example require interfering with the respective minimal positive circuits in the structure of p .

Of related work, one is particularly close to our method of computing seeds by searching for stable and consistent arc sets in the prime implicant graph. In [112] the authors propose a method for computing the *stable dynamic repertoire of a network*, i.e., the smallest subspaces that contain $C \in \text{Attr}(S, \leftrightarrow)$, by an iterative 8 step reduction method. The algorithm is based on computing *stable motifs*, which correspond to our definition of minimal seeds, and *oscillating components* which correspond, essentially, to *maximal stable but inconsistent arc sets* in the prime implicant graph. Both are obtained from the *expanded network*, a simple directed graph that assumes the role of our prime implicant graph. The concept of the expanded network is taken from [113] where *elementary signaling modes* are defined and used to predict the propagation of signals in Boolean networks. In [112] it is explained that the expanded network is derived from F by assuming that all $f \in F$ are given in DNF and that each f satisfies property 3 of Appendix A:

”If, for a state of a subset of the inputs of f , one has $f = 1$ (whatever the states of the remaining inputs), then the disjunctive form of f must have at least one of its conjunctive clauses equal to 1 when evaluated at the state of this subset of nodes.”

In [112] no formal definition of the expanded graph is given and it is not explained how to convert a given DNF into one that satisfies this property. But, it seems that this property requires exactly that each $f \in F$ is given as the (unique) disjunction of all of its prime implicants. Any other DNF will essentially correspond to a subset of arcs in the prime implicant graph and hence a subset of computed seeds.

A formal description of the algorithm that computes the stable motifs (minimal seeds) from the expanded network is missing, but Sec. III of [112] suggests that a generate-and-test approach is taken: all directed cycles in the expanded network are enumerated by *Johnson’s algorithm* [114], each is tested for the desired properties (stability and consistency) and only the inclusion-wise minimal ones are kept. The *quasi-attractors*, which correspond to our maximal seeds, are then enumerated by iteratively (1) computing a minimal seed and finding the associated symbolic steady state by percolation and (2) repeating (1) for the remaining network until no more of minimal seeds exist in it.

A preliminary comparison of running times between our ILP encoding and the *complete reduction method*, see Fig. 9 in [112], suggests that our method is faster and capable of solving harder problems. Note that [112] also generated random Boolean networks, hence our results are comparable, but only for a fixed in-degree of $k = 2$, and only for minimal seeds p of size $|p| < 40$ for networks (V, F) with $|V| > 100$ due to the large number of circuits in the expanded networks.

[112] goes beyond our results in trying to predict, heuristically, which components might stabilize due to *incomplete oscillations* and which cyclic attractors might be missed due to *unstable oscillations*. The predictions are based on observations that are comparable to the ones given in Sec. 4.4. On the other hand, [112] neither mentions the problem of univocality, which we address with CTL model checking in Sec. 4.8, nor the observation that the number of non-regular maximal seeds is a lower bound for the cyclic attractors (Thm. 3).

Overall, [112] and our work discuss very similar concepts and it would be fruitful to unify the ideas in one framework, e.g., by translating the definition of *oscillating components* (Sec. II G in [112]) into our terminology such that ILP or ASP solvers can be used to detect these objects.

But, the principle problem of efficiently characterizing which components oscillate asymptotically and which stabilize, i.e., the problem of computing the smallest subspace that contains X for every $X \in Attr(S, \rightarrow)$, remains open. Our experience with networks, the results for the *MAPK* network and the discussion in [112] suggest that in practice there is not much room for complications. It seems that for (S, \leftrightarrow) , the maximal seeds are usually complete, each one of them univocal and an exact characterization of the oscillating and stable components.

Chapter 5

Construction and Analysis of a Model Pool

”Moreover, when studying a system, the biological knowledge arrives in an incremental manner. It would be appreciable to apprehend the problem in such a way that when new knowledge has to be taken into account, previous work is not put into question. In other words, one would like to handle not only a possible model of the system, but the exhaustive set of models which are, at a given time, acceptable according to the current knowledge.”

– G. Bernot and J.-P. Comet in [72]

5.1 Introduction: Reverse Engineering

In Chap. 3 we discussed how to decide if a given property, in this case a time-series specification, holds for a fully specified model. The motivation for this chapter is the inverse problem where the property specification and whether it holds are given but the model is only partially known. The natural motivation behind this problem is its application to the reverse engineering of models from data. As mentioned in [72], the available data is usually not sufficient to reconstruct a model uniquely. Rather, there is a pool of models where each member satisfies the constraints placed by the data. The procedure of reverse engineering has often been described as an iterative, circular process that generates new constraints by interpreting the current data, while the analysis of the current model pool guides experiments that in turn generate new data and constraints. The procedure is illustrated in Figure 5.1.

At the core of this chapter is a software toolbox, currently called LOGICMODELCLASSIFIER [55], that is based on a software seminar given in 2011 at the FU Berlin. Originally, the goal was to write a software that (1) enumerates all update functions that are consistent with a given signed interaction graph, (2) filters from this initial pool all models whose state transition graph satisfies a given CTL specification, and (3) analyzes the remaining models by returning for each update function and input the set of occurring target values. Towards the end of the seminar we added several functionalities to the software.

In particular, we extended the specification language used in (1) from interaction signs to propositions about *threshold functions*, a certain class of signed input-symmetric functions, and introduced a logic for attractors, that could be used instead of CTL in step (2). In 2012 the software was completely revised and the result is the focus of this chapter.

In the following two sections we will introduce a specification language for logical models and the toolbox LOGICALMODELCLASSIFIER.

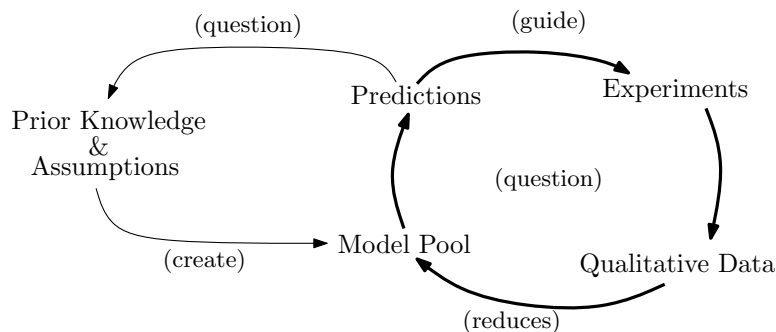


Figure 5.1: The simplified and iterative process of reverse engineering with data. It begins with the translation of prior knowledge into a set of initial constraints. All models that are consistent with them are contained in the initial model pool. The analysis of the pool leads to predictions that guide the design of new experiments. The obtained data is translated into additional constraints that reduce the number of consistent models. Inconsistent constraints lead to an empty model pool that may be used to question the prior knowledge, modeling assumptions or translation of data into constraints.

5.2 The Specification Language

The aim of this section is to propose a specification language for logical models. We will define it in terms of predicates which we group into five different themes that are loosely related to the computational complexity of the corresponding decision problems. A specification is a statement that determines some aspect of (V, F) or (S, \rightarrow) . An example is *Components* ≥ 4 which requires that the model has at least 4 components. Formally, we define the syntax and semantics by statements like

$$(V, F) \models \text{Components} \geq 5 \quad \text{iff} \quad |V| \geq 5.$$

Complex specifications are constructed by the usual logical connectives \wedge, \vee, \neg . An example is *Components* $\geq 5 \wedge \text{Components} \leq 10$.

A. Components

The two fundamental predicates, *Components* and *Max*, concern the number of components and their maximal activities. Since we are interested in cardinalities we require that they are followed by a comparison operator $\diamond \in \Omega = \{=, <, \leq, >, \geq\}$ and a natural number $k \in \mathbb{N}_1$.

$$\begin{aligned}
(V, F) \models \text{Components} \diamond k & \quad \text{iff } |V| \diamond k \\
(V, F) \models \text{Max}(v_i) \diamond k & \quad \text{iff } \text{Max}(v_i) \diamond k.
\end{aligned}$$

Throughout this chapter we assume that the number of possible components and their maximal activities are finite.

B. Interactions

Interaction labels are closely related to biological assumptions regarding the character of an interaction between two components. Typically, the regulators of a component are divided into activators and inhibitors but formally more variations exist, see for example [85]. The basic interaction predicates are *Activating* and *Inhibiting*. For two components $u, v \in V$ and $t \in \mathbb{N}$ we define

$$\begin{aligned}
(V, F) \models \text{Activating}(u, t, v) & \quad \text{iff } u \xrightarrow{+t} v \\
(V, F) \models \text{Inhibiting}(u, t, v) & \quad \text{iff } u \xrightarrow{-t} v.
\end{aligned}$$

A lot of "syntactic sugar" can be added to them, notably predicates for observability and sign-definiteness.

$$\begin{aligned}
\text{Observable}(u, t, v) & \quad :\Leftrightarrow (\text{Activating}(u, t, v) \vee \text{Inhibiting}(u, t, v)) \\
\text{ActivatingOnly}(u, t, v) & \quad :\Leftrightarrow (\text{Activating}(u, t, v) \wedge \overline{\text{Inhibiting}(u, t, v)})
\end{aligned}$$

A true increase in expressiveness is obtained by adding an additional parameter that specifies where in state space the interaction should be effective. The more expressive version of activation is, for example, *Activating*(u, t, v, d) where $d \in \text{StateDesc}$ (see Def. 2) restricts the state $x \in S$ that proves that $u \rightarrow v$ is observable (see Def. 9).

$$(V, F) \models \text{Activating}(u, t, v, d) \quad \text{iff } \exists x \in S[d \cdot (u = t - 1)] : f_v(x) < f_v(x \oplus e_u).$$

Additionally, "threshold-free" predicates are also useful for non-Boolean components.

$$\begin{aligned}
\text{Activating}(u, v) & \quad \text{iff } \exists t \in \text{Dom}(u) : u \xrightarrow{+t} v \\
\text{Observable}(u, v) & \quad \text{iff } u \rightarrow v
\end{aligned}$$

C. Update Functions

The basic predicate to affect the target values of an update function directly is to require that somewhere in state space they are all below, above or equal to a certain value. Hence, we require that the following predicates always appear together with a comparison operator $\diamond \in \Omega$ and a constant $k \in \mathbb{N}_0$.

$$\begin{aligned}
(V, F) \models \text{AllTargetValues}(v, d) \diamond k & \quad \text{iff } \forall x \in S[d] : f_v(x) \diamond k \\
(V, F) \models \text{SomeTargetValues}(v, d) \diamond k & \quad \text{iff } \exists x \in S[d] : f_v(x) \diamond k
\end{aligned}$$

Additionally, it might be useful to allow relative comparisons, i.e., comparisons among target values rather than between target values and constants. In [55] we have experimented with the following semantics

$$(V, F) \models \text{AllTargetValues}(u, d_1) \diamond \text{AllTargetValues}(v, d_2) \\ \text{iff } \forall x \in S[d_1] : \forall y \in S[d_2] : f_u(x) \diamond f_v(y)$$

and the three other cases involving *SomeTargetValues*. But, relative comparisons have so far not been necessary in practice.

Several useful shorthands are derived from the above predicates. The first one requires that all target values are equal in $S[d]$, without specifying a value.

$$(V, F) \models \text{AllEqual}(v, d) \quad \text{iff } \forall x, y \in S[d] : f_v(x) = f_v(y)$$

The second one is a shorthand for defining the update function of Boolean components.

$$\text{Boolean}(v, d) :\Leftrightarrow \text{AllTargetValues}(v, d) = 1 \wedge \text{AllTargetValues}(v, \bar{d}) = 0$$

The third one allows the use of *multiplexes* as introduced in [14]. Multiplexes can be thought of as enforcing assumptions regarding the formation of complexes among the regulators of a component. Formally they enforce several *AllEqual* constraints. Suppose that a component v has $2 \leq m$ regulators and that they can form $1 \leq k \leq 2^m$ regulatory complexes (e.g. protein complexes). The requirement for the formation of a complex can be described in terms of the states $S[d_i]$ in which it forms. That is, we assume that in $S[d_i]$ the building blocks of the i^{th} complex are present and it forms spontaneously without a state transition. Suppose further that the activity of v depends only on the presence or absence of the complexes (and not the individual regulators). Then f_v is structured into 2^k different input combination rather than 2^m . Denote by $C(d_1, \dots, d_k) \subseteq \text{StateDesc}$ the 2^k combinations of complexes being present or absent.

$$\text{Multiplex}(v, d_1, \dots, d_k) :\Leftrightarrow \forall d \in C(d_1, \dots, d_k) : \text{AllEqual}(v, d)$$

As an example consider a component $v \in V$ that has three Boolean regulators $a, b, c \in V$. Suppose a and b regulate v by forming a protein complex which is present in all $x \in S$ such that $x(a) = 1$ and $x(b) = 1$. The constraint $\text{Multiplex}(v_1, d_1, d_2)$ with $d_1 := (a = 1) \cdot (b = 1)$ and $d_2 := (c = 1)$ enforces that f_v behaves as if it had only two regulators, the protein complex and c . In this case $C = \{d_1 \cdot d_2, \bar{d}_1 \cdot d_2, d_1 \cdot \bar{d}_2, \bar{d}_1 \cdot \bar{d}_2\}$. For more details and examples see [14].

The fourth shorthand enforces the presence of subgraphs in the state transition graph. The simplest one is to enforce a transition between two states $x_1, x_2 \in S$ and a given transition relation \rightarrow .

$$(V, F) \models \text{Transition}(x_1, \rightarrow, x_2) \quad \text{iff } x_1 \rightarrow x_2$$

Useful derived predicates are

$$\begin{aligned} \text{Path}(\rightarrow, x_1, \dots, x_k) & :\Leftrightarrow \text{Transition}(x_1, \rightarrow, x_2) \wedge \dots \wedge \text{Transition}(x_{k-1}, \rightarrow, x_k) \\ \text{SteadyState}(x, \rightarrow) & :\Leftrightarrow \text{Transition}(x, \rightarrow, x) \end{aligned}$$

D. Temporal Logic

The predicates that enforce branching and linear time properties come in two flavours each, depending on which satisfaction relation, \models^\exists or \models^\forall , is chosen. Hence, for a quantifier $Q \in \{\exists, \forall\}$, a description of the initial states $d \in StateDesc$, a transition relation \rightarrow and a LTL or CTL formula ϕ resp. ψ , we define the following predicates.

$$\begin{aligned} (V, F) \models CTL(Q, d, \rightarrow, \psi) & \quad \text{iff } TS \stackrel{Q}{\models} \psi \text{ with } TS = (S, \rightarrow, S[d]) \\ (V, F) \models LTL(Q, d, \rightarrow, \phi) & \quad \text{iff } TS \stackrel{Q}{\models} \phi \text{ with } TS = (S, \rightarrow, S[d]) \end{aligned}$$

All the reachability queries that we introduced in Chap. 3, for example "nested reachability", "branching time reachability" or "best fit", can be turned into shorthand predicates based on *CTL* and *LTL*.

E. Stability and Oscillations

These predicates facilitate statements about the asymptotic stability or instability of components as defined in Sec. 4.1. First, we would like to be able to specify where in state space an attractor is. We suggest to do so by requiring the existence of a trap set or a trap space. The distinction is made mainly because the second one is potentially easier to decide, even for large networks, using our results on computing seeds in Sec. 4.6. Let $d \in StateDesc$.

$$\begin{aligned} (V, F) \models ExistsTrapSet(d) & \quad \text{iff } \exists X \in Attr : X \subseteq S[d] \\ (V, F) \models ExistsTrapSpace(d) & \quad \text{iff } \exists p \in Seeds : S[p] \subseteq S[d] \end{aligned}$$

In addition, it is useful to introduce predicates that relate directly to the stable and oscillating components of an attractor (see Sec. 4.1). Let $U, U_1, U_2 \subseteq V$.

$$\begin{aligned} (V, F) \models ExistsStab(U) & \quad \text{iff } \exists X \in Attr \text{ s.t. } u \text{ is stable in } X \forall u \in U \\ (V, F) \models ExistsOsc(U) & \quad \text{iff } \exists X \in Attr \text{ s.t. } u \text{ is unstable in } X \forall u \in U \\ (V, F) \models ExistsStabOsc(U_1, U_2) & \quad \text{iff } \exists X \in Attr \text{ s.t. } u \text{ is stable in } X \forall u \in U_1 \\ & \quad \text{and } v \text{ is unstable in } X \forall v \in U_2 \end{aligned}$$

Note that these predicates can be formulated in terms of $CTL(Q, d, \rightarrow, \psi)$ using the queries discussed in Sec. 4.2.

F. Derived Networks

We can also require that a predicate is evaluated on a network (V', F') that is obtained from (V, F) by some modification of F or V . An example are knock-in or knock-out experiments. In the basic setting, a perturbation of (V, F) is specified by $p \in Sym$ where each $(v, c) \in Set_p$ indicates that $v \in V$ is fixed at $c \in Dom(v)$. A knock-out of two components $v_1, v_2 \in V$ may for example be specified by $p \in Sym$ with $Set_p := \{(v_1, 0), (v_2, 0)\}$. For a perturbation $p \in Sym$ of (V, F) we define the update functions $F^p := \{g_v \mid v \in V\}$ by $g_v := c$ (constant) if $(v, c) \in Set_p$ and $g_v := f_v \in F$ otherwise. The predicate *Perturb* is then defined in terms of a perturbation $p \in Sym$ and an arbitrary expression ω of our specification language:

$$(V, F) \models \text{Perturb}(q, \omega) \quad \text{iff} \quad (V, F^q) \models \omega$$

Note that an alternative to using the predicate *Perturb* is to explicitly add the perturbed models (V', F') to the model pool, annotate them by whether they satisfy ω , and (somehow) consider the connection between (V, F) and (V', F') during the analysis (see Sec. 5.3).

5.3 Description of a Prototype Software

In this section we describe the software LOGICMODELCLASSIFIER [55]. It can be thought of as consisting of three parts: (1) model instantiation, (2) model annotation and (3) model analysis. There is no clear conceptual distinction between steps one and two and we distinguish them merely for computational reasons.

We use the term *model pool* as essentially meaning a set of models. A key feature of a model pool is that individual models may be equipped with additional information in the form of *labels*. In the prototype software [55] model pools are implemented by the database engine SQLite. Although a *structured query language* (SQL) database is relational we do not use this feature explicitly, i.e., the database consists of a single table and each model of the pool is stored in a single row. A model is stored as an integer vector that encodes a rule-based representation of the update functions F (see Def. 8). Additional information, knowledge or assumptions are stored in dedicated columns. Each column represents a single property, for example whether a model is compatible with a time series called "A". The *property name* is then the identifier of the column, it might for example be *TimeSeriesA*. If a model (V, F) is compatible with time series "A" we will *annotate* the model by " $(\text{TimeSeriesA} = 1)$ " and say that "1" is the label of (V, F) and the property *TimeSeriesA*.

In the following sections we describe the currently implemented methods for instantiation, annotation and analysis.

1. Model Instantiation

The user can choose between three means of model instantiation: complete enumeration, sampling and perturbation.

Name	Complete Enumeration.
Input	An expression ϕ of the specification language with predicates from Sec. 5.2 (B-C).
Output	The model pool that contains all models that satisfy ϕ .

The first approach, *complete enumeration*, requires the components V , including their maximal activities and names, and an interaction graph (V, \rightarrow) together with a threshold function T that assigns $T(u, v) \subseteq [1..Max(u)]$ to each interaction $u \rightarrow v$ in (V, \rightarrow) . Note that V and their maximal activities ensure that the number of feasible models is finite. (V, \rightarrow) and T can be thought of as a "super-structure" that is the union of all potentially existing interactions and thresholds as well as all prior knowledge interactions and thresholds. It is therefore not restrictive.

The second input is an expression ϕ of the specification language that consists exclusively of predicates mentioned in Sec. 5.2 (B-C), i.e., refers to the interactions and update functions. The expression is parsed and turned into a *constraint satisfaction problem* (CSP, see e.g. [115]) that consists of an integer variable for every *logical parameter* of every update function. In this setting, a logical parameter is essentially a rule in the representation of an update function, see [116]. In practice the CSP often splits into several independent sub-problems. For efficiency the script will partition the logical parameters into independent sub-problems that are solved separately and whose solutions are combined, by taking the Cartesian product, to yield all models that satisfy the constraint ϕ . We use the CSP solver PYTHON-CONSTRAINT, see [117].

Name	Sampling.
Input	An expression ϕ of the specification language with predicates from Sec. 5.2 (B-C) and the sample size $k \in \mathbb{N}$.
Output	A model pool that contains k models that are chosen uniformly from the set of all feasible models.

The instantiation by *sampling* is based on the previous observation that, in practice, the CSP that is created for the complete enumeration often consists of several independent sub-problems. If the feasible models can not be enumerated and stored (Cartesian product too large) we may instead choose to sample the feasible models by drawing solutions from the sub-problems. Note that we can sample uniformly from this space as it is a Cartesian product.

A typical example is the case when ϕ is a conjunction of interaction labels. The CSP for complete enumeration then splits into $|V|$ sub-problems, one for each $f \in F$. If the space of feasible models is too large to be stored explicitly but the set of feasible functions f_v for each $v \in V$ can be enumerated, the user may choose to draw a uniform sample from feasible models instead.

Name	Perturbation.
Input	A model (V, F) and a parameter $d_i \in \mathbb{N}$ for each $v_i \in V$.
Output	The model pool that contains all (V, F') that satisfy $\text{dist}(f_i, f'_i) \leq d_i$ for all $v_i \in V$.

The instantiation by a *perturbation* takes an initial model (V, F) and creates all models that are similar with respect to the logical parameters. More precisely, the distance $\text{dist}(f_i, f'_i) \in \mathbb{N}$ between two update functions f_i, f'_i of a component v_i is defined to be the sum of differences between the logical parameters of f_i and f'_i . The idea is identical to the Manhattan distance between states, see Def. 1. The numbers $d_i \in \mathbb{N}$ for each $v_i \in V$ specify the admissible distance to $f_i \in F$.

The script also accepts a model pool, rather than a single model, as an input. In that case all perturbations of all members are combined.

2. Model Annotation

During the annotation phase models are drawn from the pool and subjected to tests, typically model checking or other properties that require exploring the state transition graph. But, conceptually a test is any algorithm that computes

a label for a model. It could, for example, consist of running a certain number of stochastic simulations and returning some statistics. The result of the test is attached to the model in the form of a label. The parameters that are common to all annotation scripts are (1) a property name under which the annotation is stored (column in the database), (2) a model pool, and (3) an optional SQL expression that defines a subset of models. The *property name* specifies the name of the column that will be created to store the computed labels. Usually the whole pool is annotated, but if the annotation algorithm is computationally expensive or the pool large, we may want to annotate only a subset of models. To select subsets of models we use statements of the form

SELECT * FROM models WHERE *expr*

where * indicates that the complete row in the database is returned, **models** is the table that contains the models and *expr* is a Boolean expression over existing labels in the pool that is specified by the user.

In the following we describe the annotation scripts that are currently available.

Name	CTL Model Checking.
Parameters	A CTL formula ψ , transition relation \rightarrow , $d \in StateDesc$ and $Q \in \{\exists, \forall\}$.
Label	(<i>Boolean</i>) The truth value of $CTL(Q, d, \rightarrow, \psi)$.

The CTL model checking script computes a Boolean label: either a model satisfies $CTL(Q, d, \rightarrow, \psi)$ or it does not. There is a corresponding script for LTL.

Name	Update Functions.
Parameters	Subset of components $U \subseteq V$.
Labels	(<i>Integer</i>) Creates one label for every $v \in U$ that records the ID of the update function $f_v \in F$

The update functions script creates $|U|$ integer valued labels for each model in the pool, one for each $v \in U$, such that two models have identical labels for $v \in U$ iff they have identical update functions $f_v \in F$. The motivation for associating models with identical update functions for certain components, with each other, is that this information can be used during the analysis described below.

Name	Attractor Detection.
Parameters	A transition relation \rightarrow and $d \in StateDesc$.
Label	(<i>Text</i>) A comma-separated list of the smallest subspaces containing an attractor and the number of attractors contained.

The *attractor detection* script creates a string valued label which stores the smallest subspaces $S[p]$ for $p \in Sym$ that contain an attractor that is reachable from the initial states $S[d]$. The symbolic state p of each subspace $S[p]$ is converted into a string of the form $(v1=c1)(v2=c2) \dots (vk=ck)$ where each $vi=ci$ corresponds to some $(v_i, c_i) \in Set_p$. The label is then a comma-separated list of all such strings together with the number of attractors $X \in Attr(S, \rightarrow)$ in each subspace $S[p]$. The attractors are computed using *Tarjan's algorithm* [118]

with an explicit unfolding of the state transition graph. This implementation is therefore intended for small models with about 15 components.

Name	Seeds.
Parameters	The objective $O \in \{min, max\}$.
Label	(<i>text</i>) A comma-separated list that represents $\max(Seeds)$ or $\min(Seeds)$.

The *seeds* script creates a string valued label that represents the set $\min(Seeds)$ or $\max(Seeds)$, depending on the choice of O . Each seed p is represented in the same way as for the attractor detection script. The seeds are computed using the ILP encoding and the solver GUROBI, see Sec 4.6. Note that the number of maximal seeds is exponential in the number of stable input components and so the labels may become too long for storage very quickly.

There is also a script that contains the code of [94], i.e., an A^* search algorithm for everywhere precise time-series data and monotony specifications. Finally, there are two convenience scripts. The first one annotates models by the truth value of a single predicate of Sec. 5.2 (B-C) and the second one by a SQL selection query in terms of labels already in the database. These scripts are useful if the user wants to create custom partitions of the model pool, based on expressions over existing labels. The resulting classes can then be analyzed during subsequent analysis steps.

3. Model Analysis

The third phase analyzes the models and their labels in a given database. Whereas for a single model a property either holds or does not hold, for sets of models there are three cases. Either the property holds for all, for some or for no models. Mathematically, the analysis of a model pool is about testing whether subsets have a non-empty intersection, whether one set is a subset of another and about computing the cardinalities of sets. The analysis scripts take advantage of SQL by repeatedly executing `SELECT * WHERE expr` and `COUNT(expr)` queries. In the following we discuss the currently available scripts. As in the previous section, the user can always restrict the analysis to a subset of models by a corresponding SQL expression.

Name	Class Detection.
Parameters	A set of property names $N = \{Name_1, \dots, Name_k\}$.
Output	A table of all non-empty classes with respect to N .

The *class detection* script computes all combinations of labels of the properties N (among the selected models). Suppose, for example, that we are interested in two properties $N := \{TimeSeriesA, TimeSeriesB\}$ with Boolean labels, where each label indicates whether a model is compatible with the some expression data A , resp. B . Since there are 4 label combinations, a pool may contain up to 4 different classes with respect to N . It is often the case, however, that classes are empty, i.e., that there is no model in the pool that is annotated by that particular label combination. The script prints all non-empty classes and their cardinalities to the screen or to a CSV file.

Signs in the pool				Strictest Sign
$u \not\rightarrow v$	+	-	$+\wedge-$	
✓				<i>not observable</i>
	✓			<i>activating only</i>
		✓		<i>inhibiting only</i>
			✓	<i>dual</i>
✓	✓			<i>not inhibiting</i>
✓		✓		<i>not activating</i>
✓			✓	<i>either dual or not observable</i>
	✓	✓		<i>monotonous</i>
	✓		✓	<i>activating</i>
		✓	✓	<i>inhibiting</i>
✓	✓	✓		<i>not dual</i>
✓	✓		✓	<i>not inhibiting only</i>
✓		✓	✓	<i>not activating only</i>
	✓	✓	✓	<i>observable</i>
✓	✓	✓	✓	<i>free</i>

Table 5.1: The mapping between the interaction signs of $u \xrightarrow{t} v$ as they occur in a model pool and the strictest sign that will be returned by the script.

Name	Strictest Interaction Sign.
Parameters	An interaction $u \xrightarrow{t} v$.
Output	The strictest interaction sign of $u \xrightarrow{t} v$ among the models in the pool.

This script computes the *strictest interaction sign* for a given interaction $u \xrightarrow{t} v$. The notion is taken from [85]. It is motivated by the need to determine the character of an interaction for a set of models rather than an individual model. For individual models only four cases arise: either $u \not\rightarrow v$ or $Sign(u \xrightarrow{t} v) = \{+\}$ or $Sign(u \xrightarrow{t} v) = \{-\}$ or $Sign(u \xrightarrow{t} v) = \{+, -\}$, see Sec. 2.2. More cases arise for sets of models because now the interaction signs of individual models may disagree with each other. An interaction may, for example, be activating in some models and not observable in the other models. Hence, a set of models may have up to $2^4 = 16$ different combinations of occurring signs for $u \xrightarrow{t} v$. In [85] we proposed to compute the strictest interaction sign by mapping each of the 16 different combinations of occurring signs to the strictest sign, with respect to logical implication, that is true for each model in the set. The script does essentially the same and returns a verbal description of the strictest sign as defined in Tab. 5.1.

Name	Implication and Independence.
Parameters	Two property names $Name_1, Name_2$.
Output	All implications between $Name_1$ and $Name_2$.

The *implication and independence* script requires two property names. Let $L_1 = \{a_1, \dots, a_m\}$ and $L_2 = \{b_1, \dots, b_n\}$ be the labels that occur for them in

the pool. For each $(a, b) \in L_1 \times L_2$ the script checks whether every model that is annotated by $(Name_1 = a)$ is also annotated by $(Name_2 = b)$. Each such (a, b) is called an *implication* and we write " $(Name_1 = a) \Rightarrow (Name_2 = b)$ ". Each implication is printed to the screen. Implications that are true in both directions are called *equivalences* and marked as such. If there are no implications, it is checked whether the properties are *independent* in the sense that for every label combination $(a, b) \in L_1 \times L_2$ there is a model that is annotated by it. Two properties are therefore independent if the number of classes of $N := \{Name_1, Name_2\}$ is equal to $|L_1| \cdot |L_2|$. Note that the existence of an implication and therefore the question of independence is decided with respect to the selected models.

Chapter 6

Conclusion

Our aim was to contribute to the analysis and understanding of logical models of regulatory networks. To do so we decided to focus on the following three questions:

- (1) *How can time-series data be used for the validation of asynchronous, non-deterministic models?*
- (2) *How can the asymptotics of a specific model be predicted?*
- (3) *What is a good reverse-engineering framework given that knowledge arrives in an incremental manner?*

The starting point for the questions was the notion of compatibility between data and a given model. We suggested various extensions involving assumptions about the stability, robustness and monotony of the system. A toolbox, which is called `TEMPORALLOGICTIME SERIES`, for the conversion of data in CTL and LTL queries is available [54].

To answer the second question we studied subspaces in state space that are stable in the sense that trajectories that enter into them can not escape anymore. These trap spaces exist independently of and are identical for every transition relation. They can be seen as a generalization of steady states to steady subspaces. We proposed a method for computing the smallest and largest trap spaces of a given networks that scales well to networks with hundreds of components. To draw conclusive statements about the number of cyclic attractors we suggest to use CTL model checking combined with model reduction techniques to increase the efficiency. Although cyclic attractors may be outside of trap spaces, we believe that in practice and for the asynchronous transition graph, the smallest subspaces that contain cyclic attractors are trap spaces. A toolbox, which is called `BOOLNETFIXPOINTS`, for the computation of all maximal and minimal trap spaces is available [109].

It seems to us that the main challenge for the third question is to design a framework that is capable of employing arbitrary methods of analysis. Our suggestion is to explicitly enumerate the set of feasible models and then annotate them with additional information. A toolbox, which is called `LOGICMODEL-CLASSIFIER`, for the specification and annotation of model pools is available [55].

Bibliography

- [1] Leon Glass and Stuart A Kauffman. The logical analysis of continuous, non-linear biochemical control networks. *Journal of Theoretical Biology*, 39(1):103–129, 1973.
- [2] El Houssine Snoussi. Qualitative dynamics of piecewise-linear differential equations: a discrete mapping approach. *Dynamics and stability of Systems*, 4(3-4):565–583, 1989.
- [3] Motoyosi Sugita. Functional analysis of chemical systems in vivo using a logical circuit equivalent. ii. the idea of a molecular automaton. *Journal of Theoretical Biology*, 4(2):179–192, 1963.
- [4] René Thomas. Regulatory networks seen as asynchronous automata: A logical description. *Journal of Theoretical Biology*, 153(1):1 – 23, 1991.
- [5] René Thomas, Denis Thieffry, and Marcelle Kaufman. Dynamical behaviour of biological regulatory networks i. biological role of feedback loops and practical use of the concept of the loop-characteristic state. *Bulletin of mathematical biology*, 57(2):247, 1995.
- [6] Denis Thieffry and René Thomas. Dynamical behaviour of biological regulatory networks ii. immunity control in bacteriophage lambda. *Bulletin of Mathematical Biology*, 57(2):277–297, 1995.
- [7] Luis Mendoza, Denis Thieffry, and Elena R. Alvarez-Buylla. Genetic control of flower morphogenesis in *Arabidopsis thaliana*: a logical analysis. *Bioinformatics*, 15(7):593–606, 1999.
- [8] Adrien Fauré and Denis Thieffry. Logical modelling of cell cycle control in eukaryotes: a comparative study. *Molecular BioSystems*, 5(12):1569–1581, 2009.
- [9] Lucas Sánchez and Denis Thieffry. A logical analysis of the drosophila gap-gene system. *Journal of Theoretical Biology*, 211(2):115 – 141, 2001.
- [10] Réka Albert and Hans G. Othmer. The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in *Drosophila melanogaster*. *arXiv preprint q-bio/0311019*, 2003.
- [11] Luca Grieco, Laurence Calzone, Isabelle Bernard-Pierrot, François Radvanyi, Brigitte Kahn-Perlès, and Denis Thieffry. Integrative modelling of the influence of mapk network on cancer cell fate decision. *PLoS computational biology*, 9(10):e1003286, 2013.

- [12] Stuart A Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of theoretical biology*, 22(3):437–467, 1969.
- [13] Gilles Bernot, Jean-Paul Comet, Adrian Richard, and Janine Guespin. Application of formal methods to biological regulatory networks: Extending Thomas’ asynchronous logical approach with temporal logic. *Journal of Theoretical Biology*, 229(3):339–347, 2004.
- [14] Zohra Khalis, Jean-Paul Comet, Adrien Richard, and Gilles Bernot. The SMBioNet method for discovering models of gene regulatory networks. *Genes, Genomes and Genomics*, 3(special issue 1):15–22, 2009.
- [15] François Fages and Sylvain Soliman. Formal cell biology in biocham. In *Formal Methods for Computational Systems Biology*, pages 54–80. Springer, 2008.
- [16] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.
- [17] Edmund M Clarke and IA Draghicescu. Expressibility results for linear-time and branching-time logics. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 428–437. Springer, 1989.
- [18] Pedro T. Monteiro, Delphine Ropers, Radu Mateescu, Ana T. Freitas, and Hidde de Jong. Temporal logic patterns for querying dynamic models of cellular interaction networks. *Bioinformatics*, 24(16):i227–i233, 2008.
- [19] Jack Heidel, John Maloney, Christopher Farrow, and JA Rogers. Finding cycles in synchronous boolean networks with applications to biochemical systems. *International Journal of Bifurcation and Chaos*, 13(03):535–552, 2003.
- [20] Vincent Devloo, Pierre Hansen, and Martine Labbé. Identification of all steady states in large networks by logical analysis. *Bulletin of mathematical biology*, 65(6):1025–1051, 2003.
- [21] David James Irons. Improving the efficiency of attractor cycle identification in boolean networks. *Physica D: Nonlinear Phenomena*, 217(1):7–21, 2006.
- [22] Shu-Qin Zhang, Morihiro Hayashida, Tatsuya Akutsu, Wai-Ki Ching, and Michael K Ng. Algorithms for finding small attractors in boolean networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2007:4–4, 2007.
- [23] Abhishek Garg, Alessandro Di Cara, Ioannis Xenarios, Luis Mendoza, and Giovanni De Micheli. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics*, 24(17):1917–1925, 2008.
- [24] Tatsuya Akutsu, Morihiro Hayashida, and Takeyuki Tamura. Integer programming-based methods for attractor detection and control of boolean networks. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 5610–5617. IEEE, 2009.

- [25] Thomas Skodawessely and Konstantin Klemm. Finding attractors in asynchronous boolean dynamics. *Advances in Complex Systems*, 14(03):439–449, 2011.
- [26] Elena Dubrova and Maxim Teslenko. A SAT-based algorithm for finding attractors in synchronous boolean networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 8(5):1393–1399, 2011.
- [27] Nikolaos Berntenis and Martin Ebeling. Detection of attractors of large boolean networks via exhaustive enumeration of appropriate subspaces of the state space. *BMC bioinformatics*, 14(1):361, 2013.
- [28] Martin Hopfensitz, Christoph Müssel, Markus Maucher, and Hans A Kestler. Attractors in boolean networks: a tutorial. *Computational Statistics*, 28(1):19–36, 2013.
- [29] Alan Veliz-Cuba, Boris Aguilar, Franziska Hinkelmann, and Reinhard Laubenbacher. Steady state analysis of boolean molecular network models via model reduction and computational algebra. *BMC bioinformatics*, 15(1):221, 2014.
- [30] Aurélien Naldi, Elisabeth Remy, Denis Thieffry, and Claudine Chaouiya. Dynamically consistent reduction of logical regulatory graphs. *Theoretical Computer Science*, 412(21):2207–2218, 2011.
- [31] Assieh Saadatpour, Reka Albert, and Timothy Reluga. A reduction method for boolean networks proven to conserve attractors. 2013.
- [32] René Thomas. On the relation between the logical structure of systems and their ability to generate multiple steady states or sustained oscillations. In *Numerical methods in the study of critical phenomena*, pages 180–193. Springer, 1981.
- [33] Adrien Richard and Jean-Paul Comet. Necessary conditions for multistationarity in discrete dynamical systems. *Discrete Appl. Math.*, 155:2403–2413, November 2007.
- [34] Adrien Richard. Negative circuits and sustained oscillations in asynchronous automata networks. *Advances in Applied Mathematics*, 44(4):378–392, 2010.
- [35] Elisabeth Remy and Paul Ruet. From minimal signed circuits to the dynamics of boolean regulatory networks. 24(16):i220–i226, 2008.
- [36] J. Demongeot, A. Elena, M. Noual, S. Sené, and F. Thuderoz. "immunetworks", intersecting circuits and dynamics. *Journal of Theoretical Biology*, 280(1):19–33, 2011.
- [37] Jean-Paul Comet, Mathilde Noual, Adrien Richard, Julio Aracena, Laurence Calzone, Jacques Demongeot, Marcelle Kaufman, Aurélien Naldi, Denis Thieffry, et al. On circuit functionality in boolean networks. *Bulletin of mathematical biology*, pages 1–14, 2012.

- [38] Stuart A. Kauffman. *The origins of order: Self organization and selection in evolution*. Oxford University Press, USA, 1993.
- [39] Francoise Fogelman-Soulie. Parallel and sequential computation on boolean networks. *Theoretical computer science*, 40:275–300, 1985.
- [40] Heike Siebert. Analysis of discrete bioregulatory networks using symbolic steady states. *Bulletin of Mathematical Biology*, 73:873–898, 2011.
- [41] Tatsuya Akutsu, Satoru Kuhara, Osamu Maruyama, and Satoru Miyano. Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 695–702. Society for Industrial and Applied Mathematics, 1998.
- [42] Steffen Klamt, Julio Saez-Rodriguez, Jonathan A Lindquist, Luca Simeoni, and Ernst D Gilles. A methodology for the structural and functional analysis of signaling and regulatory networks. *BMC bioinformatics*, 7(1):56, 2006.
- [43] Aurélien Naldi, Duncan Berenguier, Adrien Fauré, Fabrice Lopez, Denis Thieffry, and Claudine Chaouiya. Logical modelling of regulatory networks with ginsim 2.3. *Bio Systems*, 97(2):134–139, 2009.
- [44] Christoph Müssel, Martin Hopfensitz, and Hans A. Kestler. Boolnet - an R package for generation, reconstruction, and analysis of boolean networks. *Bioinformatics*, (10):1378–1380, 2010.
- [45] Alessandro Di Cara, Abhishek Garg, Giovanni De Micheli, Ioannis Xenarios, and Luis Mendoza. Dynamic simulation of regulatory networks using squad. *BMC bioinformatics*, 8(1):462, 2007.
- [46] Fabien Corblin, Sébastien Tripodi, Eric Fanchon, Delphine Ropers, and Laurent Trilling. A declarative constraint-based method for analyzing discrete genetic regulatory networks. *Biosystems*, 98(2):91 – 104, 2009.
- [47] Fabien Corblin, Eric Fanchon, and Laurent Trilling. Applications of a formal approach to decipher discrete genetic networks. *BMC Bioinformatics*, 11(1), July 2010.
- [48] Fabien Corblin, Eric Fanchon, Laurent Trilling, Claudine Chaouiya, and Denis Thieffry. Automatic inference of regulatory and dynamical properties from incomplete gene interaction and expression data. In *Proceedings of the 9th international conference on Information Processing in Cells and Tissues*, IPCAT’12, pages 25–30, Berlin, Heidelberg, 2012. Springer-Verlag.
- [49] Mats Carlsson and Per Mildner. Sicstus prolog—the first 25 years. *Theory and Practice of Logic Programming*, 12(1-2):35–66, 2012.
- [50] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The potsdam answer set solving collection. *Ai Communications*, 24(2):107–124, 2011.

- [51] Julio Saez-Rodriguez, Leonidas G Alexopoulos, Jonathan Epperlein, Regina Samaga, Douglas A Lauffenburger, Steffen Klamt, and Peter K Sorger. Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. *Molecular systems biology*, 5(1), 2009.
- [52] Carito Guziolowski, Santiago Videla, Federica Eduati, Sven Thiele, Thomas Cokelaer, Anne Siegel, and Julio Saez-Rodriguez. Exhaustively characterizing feasible logic models of a signaling network using answer set programming. *Bioinformatics*, 30(13):1942–1942, 2014.
- [53] Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. Hypergraphs and cellular networks. *PLoS computational biology*, 5(5):e1000385, 2009.
- [54] Hannes Klärner. sourceforge.net/projects/temporallogictimeseries, 2014. Online, accessed in November 2014.
- [55] Hannes Klärner. sourceforge.net/projects/logicmodelclassifier, 2014. Online, accessed in November 2014.
- [56] Yaman Barlas and Stanley Carpenter. Philosophical roots of model validation: two paradigms. *System Dynamics Review*, 6(2):148–166, 1990.
- [57] Adrien Richard, Jean-Paul Comet, and Gilles Bernot. R. thomas’ modeling of biological regulatory networks: Introduction of singular states in the qualitative dynamics. *Fundamenta Informaticae*, 65(4):373–392, 2005.
- [58] Heike Siebert and Alexander Bockmayr. Relating attractors and singular steady states in the logical analysis of bioregulatory networks. In *Algebraic Biology*, pages 36–50. Springer, 2007.
- [59] François Robert and Jon Rokne. *Discrete iterations: a metric study*. Springer-Verlag New York, 1986.
- [60] Duncan Bérenguier, Claudine Chaouiya, Pedro T. Monteiro, Aurélien Naldi, Elisabeth Remy, Denis Thieffry, and Line Tichit. Dynamical modeling and analysis of large cellular regulatory networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 23(2):–, 2013.
- [61] Mathilde Noual. *Updating Automata Networks*. PhD thesis, École normale supérieure de Lyon, 2012.
- [62] Mathilde Noual. Synchronism vs asynchronism in boolean networks. *arXiv preprint arXiv:1104.4039*, 2011.
- [63] Carlos Gershenson. Classification of random boolean networks. *Artificial Life*, 8:1–8, 2003.
- [64] Carlos Gershenson. Updating schemes in random boolean networks: Do they really matter. In *Artificial Life IX Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems*, pages 238–243. MIT Press, 2004.

- [65] René Thomas. *Kinetic Logic: A Boolean Approach to the Analysis of Complex Regulatory Systems*. Springer-Verlag, 1979. Lecture notes in Biomathematics, vol. 29.
- [66] Shahrads Jamshidi. *Comparing discrete, continuous and hybrid modelling approaches of gene regulatory networks*. PhD thesis, Freie Universität Berlin, 2013.
- [67] Shahrads Jamshidi, Heike Siebert, and Alexander Bockmayr. Comparing discrete and piecewise affine differential equation models of gene regulatory networks. In *Information Processign in Cells and Tissues*, pages 17–24. Springer, 2012.
- [68] Therese Lorenz, Heike Siebert, and Alexander Bockmayr. Analysis and characterization of asynchronous state transition graphs using extremal states. *Bulletin of mathematical biology*, 75(6):920–938, 2013.
- [69] Adam Streck and Heike Siebert. Equivalences in multi-valued asynchronous models of regulatory networks. In Jaroslaw Was, Georgios Ch. Sirakoulis, and Stefania Bandini, editors, *Cellular Automata*, volume 8751 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014.
- [70] Homepage of the association for computing machinery. amturing.acm.org. Online, accessed in November 2014.
- [71] Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos, François Fages, and Vincent Schächter. Modeling and querying biomolecular interaction networks. *Theoretical Computer Science*, 325(1):25–44, 2004.
- [72] Gilles Bernot and Jean-Paul Comet. On the use of temporal formal logic to model gene regulatory networks. In *Computational Intelligence Methods for Bioinformatics and Biostatistics*, pages 112–138. Springer, 2010.
- [73] Jonathan Fromentin, Jean-Paul Comet, Pascale Le Gall, and Olivier Roux. Analysing gene regulatory networks by both constraint programming and model-checking. In *EMBC07, 29th IEEE EMBS Annual International Conference*, pages 4595–4598, August 23–26 2007.
- [74] Gustavo Arellano, Julián Argil, Eugenio Azpeitia, Mariana Benítez, Miguel A Carrillo, Pedro Arturo Góngora, David A Rosenblueth, Elena R Alvarez-Buylla, et al. "Antelope": a hybrid-logic model checker for branching-time boolean GRN analysis. *BMC bioinformatics*, 12(1):490, 2011.
- [75] Miguel Carrillo, Pedro A Góngora, and David A Rosenblueth. An overview of existing modeling tools making use of model checking in the analysis of biochemical networks. *Frontiers in plant science*, 3, 2012.
- [76] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification, CAV '02*, pages 359–364, London, UK, UK, 2002. Springer-Verlag.

- [77] Alessandro Cimatti, Edmund Clarke, Fausto Giunchiglia, and Marco Roveri. Nusmv: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.
- [78] E Clarke, K McMillan, S Campos, and V Hartonas-Garmhausen. Symbolic model checking. In *Computer Aided Verification*, pages 419–422. Springer, 1996.
- [79] Christopher James Langmead and Sumit Kumar Jha. Symbolic approaches for finding control strategies in boolean networks. *Journal of Bioinformatics and Computational Biology*, 7(02):323–338, 2009.
- [80] Vepkhia Pilauri, Maria Bewley, Cuong Diep, and James Hopper. Gal80 dimerization and the yeast gal gene switch. *Genetics*, 169(4):1903–1914, 2005.
- [81] Irene Cantone, Lucia Marucci, Francesco Iorio, Maria Aurelia Ricci, Vincenzo Belcastro, Mukesh Bansal, Stefania Santini, Mario Di Bernardo, Diego Di Bernardo, and Maria Pia Cosma. A yeast synthetic network for in vivo assessment of reverse-engineering and modeling approaches. *Cell*, (1):172–181, 2009.
- [82] Gregory Batt, Michel Page, Irene Cantone, Gregor Goessler, Pedro Monteiro, and Hidde de Jong. Efficient parameter search for qualitative models of regulatory networks using symbolic model checking. *Bioinformatics*, 26:i603–i610, September 2010.
- [83] Grégory Batt, Delphine Ropers, Hidde De Jong, Johannes Geiselman, Radu Mateescu, Michel Page, and Dominique Schneider. Validation of qualitative models of genetic regulatory networks by model checking: Analysis of the nutritional stress response in escherichia coli. *Bioinformatics*, 21(suppl 1):i19–i28, 2005.
- [84] Hannes Klarner, Heike Siebert, and Alexander Bockmayr. Parameter inference for asynchronous logical networks using discrete time series. In *Proceedings of the 9th International Conference on Computational Methods in Systems Biology*, CMSB ’11, pages 121–130, New York, NY, USA, 2011. ACM.
- [85] Hannes Klarner, Heike Siebert, and Alexander Bockmayr. Time series dependent analysis of unparametrized Thomas networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9:1338–1351, 2012.
- [86] Hannes Klarner, Adam Streck, David Šafránek, Juraj Kolčák, and Heike Siebert. Parameter identification and model ranking of Thomas networks. In David Gilbert and Monika Heiner, editors, *Computational Methods in Systems Biology*, Lecture Notes in Computer Science, pages 207–226. Springer Berlin / Heidelberg, 2012.
- [87] Ziv Bar-Joseph. Analyzing time series gene expression data. *Bioinformatics*, 20(16):2493–2503, 2004.

- [88] Ziv Bar-Joseph, Anthony Gitter, and Itamar Simon. Studying and modelling dynamic biological processes using time-series gene expression data. *Nature Reviews Genetics*, 13(8):552–564, 2012.
- [89] Elena S. Dimitrova, John J. McGee, and Reinhard C. Laubenbacher. Discretization of time series data. *Journal of Computational Biology*, 17(6):853–868, 2010.
- [90] Tatsuya Akutsu, Satoru Miyano, Satoru Kuhara, et al. Identification of genetic networks from a small number of gene expression patterns under the boolean network model. In *Pacific Symposium on Biocomputing*, volume 4, pages 17–28. World Scientific Maui, Hawaii, 1999.
- [91] Shoudan Liang, Stefanie Fuhrman, Roland Somogyi, et al. Reveal, a general reverse engineering algorithm for inference of genetic network architectures. In *Pacific symposium on biocomputing*, volume 3, page 2, 1998.
- [92] Reinhard Laubenbacher and Brandilyn Stigler. A computational algebra approach to the reverse engineering of gene regulatory networks. *Journal of theoretical biology*, 229(4):523, 2004.
- [93] Elena Dimitrova, Luis David García-Puente, Franziska Hinkelmann, Abdul S Jarrah, Reinhard Laubenbacher, Brandilyn Stigler, Michael Stillman, and Paola Vera-Licona. Parameter estimation for boolean models of biological networks. *Theoretical Computer Science*, 412(26):2816–2826, 2011.
- [94] Sascha Meiers. Graph traversal versus model checking in deciding compatibility of time series with logical networks. Bachelor thesis, Freie Universität Berlin, 2011.
- [95] Adrien Fauré, Aurélien Naldi, Claudine Chaouiya, and Denis Thieffry. Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. In *ISMB (Supplement of Bioinformatics)'06*, pages 124–131, 2006.
- [96] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman New York, 1979.
- [97] Leon Glass and Hava T Siegelmann. Logical and symbolic analysis of robust biological dynamics. *Current opinion in genetics & development*, 20(6):644–649, 2010.
- [98] Tatsuya Akutsu, Zhao Yang, Morihiro Hayashida, and Takeyuki Tamura. Integer programming-based approach to attractor detection and control of boolean networks. *IEICE Transactions on Information and Systems*, 95(12):2960–2970, 2012.
- [99] Ilya Shmulevich and Wei Zhang. Binary analysis and optimization-based normalization of gene expression data. *Bioinformatics*, (4):555–565, 2002.
- [100] Adam Streck, David Šafránek, and Luboš Brim. Parsybone: Parameter synthesizer for boolean networks, 2012.

- [101] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic model checking using sat procedures instead of bdds. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 317–320. ACM, 1999.
- [102] Edmund Clarke, Somesh Jha, Yuan Lu, and Helmut Veith. Tree-like counterexamples in model checking. In *Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on*, pages 19–29. IEEE, 2002.
- [103] Hannes Klarner, Alexander Bockmayr, and Heike Siebert. Computing symbolic steady states of boolean networks. In Jaroslaw Was, Georgios Ch. Sirakoulis, and Stefania Bandini, editors, *Cellular Automata*, volume 8751 of *Lecture Notes in Computer Science*, pages 561–570. Springer International Publishing, 2014.
- [104] Qianchuan Zhao. A remark on “scalar equations for synchronous boolean networks with biological applications”. *Neural Networks, IEEE Transactions on*, 16(6):1715–1716, 2005.
- [105] Willard V. Quine. The problem of simplifying truth functions. *The American Mathematical Monthly*, 59(8):521–531, 1952.
- [106] Yves Crama and Peter L Hammer. *Boolean functions: theory, algorithms, and applications*, volume 142. Cambridge University Press, 2011.
- [107] Robert Dick. Quine-McCluskey two-level logic minimization method. pypi.python.org/pypi/qm/0.2, 2008. Online, accessed in November 2014.
- [108] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2014.
- [109] Hannes Klarner. [Sourceforge.net/Projects/BoolNetFixpoints](https://sourceforge.net/projects/BoolNetFixpoints), 2014. Online, accessed in November 2014.
- [110] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. *Answer set programming: A primer*. Springer, 2009.
- [111] Elisabeth Remy, Brigitte Mossé, Claudine Chaouiya, and Denis Thieffry. A description of dynamical graphs associated to elementary regulatory circuits. *Bioinformatics*, 19(suppl 2):ii172–ii178, 2003.
- [112] Jorge GT Zañudo and Réka Albert. An effective network reduction approach to find the dynamical repertoire of discrete dynamic networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 23(2):025111, 2013.
- [113] Rui-Sheng Wang and Réka Albert. Elementary signaling modes predict the essentiality of signal transduction network components. *BMC systems biology*, 5(1):44, 2011.
- [114] Donald B Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

- [115] Alexander Bockmayr and John N Hooker. Constraint programming. In K. Aardal, G. Nemhauser, and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, chapter 10, pages 559 – 600. Elsevier, 2005.
- [116] René Thomas and Richard d’Ari. *Biological feedback*. CRC press, 1990.
- [117] Gustavo Niemeyer. Constraint solving problem resolver for python. labix.org/python-constraint, 2011. Online, accessed in November 2014.
- [118] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

Declaration

I assert to have written this PhD thesis on my own, that I cited all the sources I used, and that I did not use any illegitimate tools.

Ich versichere diese Arbeit selbständig, ohne unerlaubte Hilfsmittel verfasst und keine außer den angeführten Quellen verwendet zu haben.

Berlin, den 12. November 2014

Hannes Klarner