Program Synthesis for Systems Biology

by

Ali Sinan Köksal

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Rastislav Bodík, Co-chair
Professor Nir Yosef, Co-chair
Professor Ellen Robey
Professor Stuart Russell

Spring 2018

Program Synthesis for Systems Biology

Abstract

Program Synthesis for Systems Biology

by

Ali Sinan Köksal

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Rastislav Bodík, Co-chair

Professor Nir Yosef, Co-chair

Cell signaling controls basic cellular activities and coordinates cell actions, such as cell differentiation, division and growth. Consequently, errors in cellular signaling are responsible for diseases such as cancer, autoimmunity, and diabetes.

Executable biology describes mechanistic models of biological processes in a formal language that is dynamic and executable by a computer. Models in executable biology are able to capture complex behaviors of biological systems, such as time and concurrency. In addition, discrete modeling enables efficient algorithms to exhaustively explore spaces of models.

This thesis introduces tools to automatically infer executable models at different levels of abstraction from varied types of experimental data. In each case, we investigate identifiability of models when the provided experimental evidence and prior knowledge are varied. We make the following individual contributions:

- TPS: A framework for the automated inference of signed directed graphs modeling protein signaling networks, using time series data.
- SBL: A modeling language embedded in Scala for the automated synthesis of concurrent programs modeling cell fate decision using mutation experiments.
- Karme: A framework for investigating identifiability of asynchronous Boolean network models from single-cell gene expression data.

To evaluate our work, we apply our tools to *in vivo*, *in vitro*, and *in silico* data sets on cellular differentiation and protein signaling. We show that, through explicit characterization of ambiguities in input specifications, our approaches make unambiguous predictions supported by experimental evidence, and suggest new experiments that help disambiguate alternative explanations. Applied to epidermal growth factor signaling response data, TPS exhaustively explores all models that are consistent with the input, and makes predictions that are unambiguous across the model space. These predictions are supported by further experimental validation. Using SBL, we synthesize valid models of cell fate decision in *C. elegans* vulval precursor cells, fixing a bug in previous modeling. We show the existence of internally different models that are behaviorally equivalent under all mutation experiments. One of the inferred models expresses a previously unknown biological hypothesis. Finally, we use Karme to synthesize models of myeloid cell differentiation from simulated noisy single-cell data, and demonstrate that experimental design can reveal key genes in the system.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my gratitude to my advisors Ras Bodík and Nir Yosef, whose guidance and optimism helped me greatly throughout my graduate studies. I had the chance to start working with Ras in 2010 as a visiting student, and it has been a great pleasure to learn from him and to be inspired by his encouraging optimism throughout all these years. I am also grateful to Nir for his guidance in the final years of my graduate studies, and I benefited deeply from his insight into biology. Thank you both for your constant support throughout this journey.

I would like to thank my other collaborators, in particular Saurabh Srivastava and Anthony Gitter. Saurabh paved the way for our work on modeling biology by initiating our first collaboration on the subject. Working closely with Saurabh and Anthony throughout these years has been an enriching experience. I would also like to thank all my other collaborators: Kirsten Beck, Nathan D. Camp, Dylan R. Cronin, Jasmin Fisher, Ernest Fraenkel, Matthew E. MacGilvray, Aaron McKenna, Nir Piterman, Yewen Pu, Aditya V. Thakur, and Alejandro Wolf-Yadlin. I thoroughly enjoyed collaborating with you.

I would also like to thank members of my dissertation committee, Ellen Robey and Stuart Russell, for their valuable feedback.

It has been great working alongside other members of Ras' lab, Nir's lab, and the PLSE lab at the University of Washington. Thanks for all the fun conversations we have had.

Finally, I'd like to thank my parents, Aykut and Dilek, my friends, and Keahilani, for their support and encouragement.

Chapter 1

Introduction

Biological systems are extremely complex, and are a combination of tightly-coupled systems at various levels of magnitude. At the molecular level, we have DNA, RNA, proteins, and metabolites. These molecules are parts of a cell, which in turn is a part of a tissue. Different tissues constitute the organs of an organism. Many organisms together form the ecosystem. At all these levels of details, the relationship between the elements are of great interest. The goal of this dissertation is to develop tools and techniques for automatically inferring models of biological processes at the levels of gene regulation as well as protein signaling, and, consequently, raise the level of automation in computational biology.

1.1 Background

Cellular control. In this dissertation, we focus on mechanisms for cell signaling. Cell signaling controls basic cellular activities and coordinates cell actions, such as cell division and growth, and controls cell differentiation. Therefore, errors in cellular signaling are responsible for diseases such as cancer, autoimmunity, and diabetes. We focus our attention on the interactions of proteins and genes, because proteins control and mediate the vast majority of biological processes in a living cell.

There are at least two practical reasons for studying cellular control networks:

- *Drug design:* Interactions between specific proteins are essential to all stages of development and homeostasis. Not surprisingly, many human diseases can be traced to aberrant protein-protein interactions, either through the loss of an essential interaction or through the formation of a protein complex at an inappropriate time or location [127]. Hence, a potential therapeutic strategy for treating a disease involves the drug-induced manipulation of protein-protein interactions whose malfunctioning contributes to that disease [158].

  A recent example of how controlling cell development can lead to a therapeutic cure for a disease is found in the work by Solt et al. [142]. Solt et al. discovered a way to significantly reduce diabetes in mice. Type I diabetes is a chronic autoimmune disease that occurs when the body's immune system destroys insulin producing pancreatic β cells, resulting in insulin deficiency and hyperglycemia. Current treatments for type I diabetes focus on controlling blood sugar with insulin therapy and must continue throughout a person's life. The method discovered by Solt et al. prevents the illness from developing rather than treating its symptoms. Their research built on the observation that T_H17 cells have been linked to the autoimmune process that destroys β cells. Solt et al. [142] discovered two nuclear receptor transcription factors (ROR_α and ROR_γ) that play a critical role in the development of T_H17 cells. Blocking these ROR receptors using SR1001 (a selective agonist developed by Solt et al.) significantly reduced diabetes in mice that were treated with it. As a result, the use of drugs that target the ROR receptors of the T_H17 cells may offer a new treatment for the illness.

- *Induced pluripotency:* Pluripotency is a property of stem cells that refers to the potential to differentiate into any of the multiple possible germ layers. During mammalian development, cells gradually lose potential and become progressively differentiated to fulfill the specialized functions of somatic tissues. Induced pluripotency [144] artificially derives a pluripotent stem cell from a non-pluripotent cell by overexpressing a small set of proteins. Such induced pluripotent stem cells (iPSCs) can then be redifferentiated into various cell types. Thus, iPSCs find uses in cell and tissue transplants without the risk of rejection that is commonly encountered. iPSCs also enable producing custom-tailored cells for the study and treatment of numerous human diseases, and replacing unsuitable animal and in-vitro models used previously [110, 160].

Mechanistic models. A mechanistic model is one where the basic elements of the model have a direct correspondence to the underlying mechanisms in the system being modeled. In contrast, empirical models are solely based on direct observation and measurement, with no understanding of the underlying system. Consequently, mechanistic models have greater predictive power.

One of the simplest and most widely-used example of a mechanistic model used in computational biology is protein-protein interaction (PPI) networks. A PPI network identifies which proteins interact with each other. PPI can be visualized as an undirected or directed graph in which nodes represent proteins and an edge represents whether two proteins interact [161, 125]. The topology of such a graph abstraction can provide insights into the importance of a particular protein [72].

A signaling pathway model may represent each protein type as a single node, where the state of the node is the discretized concentration of the protein, and represent interactions between proteins as edges. Gene regulatory networks (GRNs) are represented similarly, where each node represents one protein, and an edge from one protein to another means that the first protein (a transcription factor) binds the DNA that encodes the second protein.

Mechanistic models are especially useful in understanding cell signaling, as illustrated below:

- *Drug design:* A purely empirical approach to identifying which genes and proteins are involved in a disease is to look for genes whose expression differs significantly in the healthy and diseased cell. Having narrowed down such differentially expressed genes, we can try out different mechanisms to activate or inhibit corresponding proteins. However, such an empirical approach does not lead to a proper understanding of the side effects of the treatment. To be able to forecast such side effects, it is critical to understand the *system-wide effect* of activating or inhibiting a particular protein-protein interaction [66, 11]. For instance, questions about the other tissue-specific effects of ROR inhibition in T_H17 cells remain in the work by Solt et al. [142]. Thus, identifying new therapeutic methods and understanding their side effects requires synthesizing a mechanistic model of cellular control.

- *Induced pluripotency:* The main observation behind induced pluripotency was that transcription factors are key determinants of cell fate. Thus, by forcing the expression of certain transcription factors we can reprogram adult cells into pluripotent cells. Recently, Dunn et al. [41] synthesized a mechanistic model that explains observed mouse embryonic stem (ES) cell behavior. Their model revealed the essential program controlling cell pluripotency. This model also accurately predicted responses to genetic perturbation, which illustrates the extrapolatory power of mechanistic models. Furthermore, more general forms of cellular reprogramming is also possible that converts one cell type into another directly, without the need to first revert the cell to an undifferentiated pluripotent state. For instance, researchers have identified sets of transcription factors that induce the conversion of pancreatic acinar cells into insulin-producing β cells [175]. Identifying the mechanism for such cellular reprogramming entails building a mechanistic model of cellular control.

Executable biology. Executable biology describes mechanistic models of biological processes in a formal language that is dynamic and executable by a computer [44]. Thus, using such models biologists can simulate the underlying biological process on different input conditions. Furthermore, models in executable biology are able to capture complex behaviors of biological systems, such as time and concurrency. Such capabilities are required if we wish to capture the temporal and contextual signals underlying the protein interactions [117]. Apart from their explanatory and predictive uses, models for executable biology can be formally verified to ensure that they satisfy properties such as bi-stability, which implies that they are faithful models [20, 34, 32]. The simplest model for executable biology that models protein interactions is a directed graph in which each node n represent protein p, and an edge directed from node n_i to n_j indicates that change in the activity of protein p_i affects the activity of protein p_j. Thus, while the (undirected) PPI network merely represents a physical binding relationship between proteins, the directed-graph model expresses a *causation* relationship. The work by Dunn et al. [41] described earlier is one recent instance illustrating the advantage of synthesizing models for executable biology. Dunn et al. synthesized a directed-graph model to explain observed mouse ES cell behavior. Though the gene regulatory circuit controlling whether ES cells self-renew or differentiate appears vast, the simplest model synthesized by Dunn et al., which explains observed behavior, consists of only 16 interactions, 12 components, and three inputs. In our work on modeling the epidermal growth factor receptor signaling mechanism, we use signed directed graphs to model protein interactions (Chapter 2).

Models in executable biology, however, go beyond simple signed directed networks. For instance, concurrent finite state machines capture the synchronization or the more complex logical relation among proteins. Various other types of models for executable biology exist, which capture different characteristics of the interaction between the nodes [78, 171, 36, 136, 131, 155, 134, 33]. For instance, in our work, we used an asynchronous Boolean network to model the developmental progress of vulval precursor cells in *C. elegans* (Chapter 3).

Challenges in Executable Biology. It is tempting to equate inferring executable models of biological processes to reverse engineering a high-level program from machine code [149, 121]. This analogy breaks down for the following reasons [29]. First, the granularity and precision of the data available when trying to reverse engineer a program is significantly better than for a biological system. It would be every biologist's dream to insert a break-point into a cell and non-destructively analyze the state of the system! In contrast, biologists have access to sparse information of multiple types. Hence, to infer an executable model we have to rely on indirect inference and collate multiple sources of data. Even then some level of ambiguity must be accepted. Second, biological systems tend to be very noisy with multiple side conditions and experimental anomalies. Biologists often develop intuitions to weed out unimportant phenomena. These biological intuitions and assumptions need to be incorporated when inferring an executable model.

1.2 Existing Tools for Modeling Biological Systems as Boolean Networks

We now briefly describe other tools available to biologists who are studying cell signaling networks, focusing on the modeling of systems as *Boolean networks*. We categorize such tools into three types as follows:

- *Visualization tools:* Tools such as Cytoscape [132], PathVisio [69], BioSPICE [54], and Pathway Studio [106] are platforms for visualizing biological networks. Cytoscape, for instance, allows networks to be integrated with annotations, gene expression profiles and other data. Limited forms of network analysis, such as identifying motifs, is possible in Cytoscape via the use of plug-ins. However, none of these tools are capable of synthesizing the network model from experimental data. BioTapestry [92] is a tool for visualization and simulation gene regulatory networks (GRNs).

- *Analysis tools:* BoolNet [104] is an R package that computes attractors for synchronous and asynchronous Boolean networks. Antelope [6] is a model checker for analyzing asynchronous Boolean networks that describe GRNs. BioModel Analyzer (BMA) [19] allows the user to create a model of the biological network using a graphical interface. The tool then translates this graphical model into a Quantitative Network [131] to prove stabilization of the model [34]. Z34Bio [168, 167] is an SMT-based analysis tool for gene regulatory and chemical reaction networks. It encodes the verification problem into a formula in bit-vector logic, and uses bounded model checking to verify properties of biological networks.

- *Synthesis tools:* Recently, the Z34Bio tool has been extended to perform synthesis of GRNs using gene expression data [109]. This work uses a synchronous execution model based on semantics proposed in prior work [116]. CellNOptR [148] derives a Boolean logic model from a prior knowledge network (PKN, i.e., a network obtained from literature or expert knowledge) and trains it against perturbation data.

  GenePath [177] derives a signed directed graph representing a GRN using gene expression data. Thus, this tool does not support synthesizing richer logical relations between genes.

1.3 Overview

The work presented in this dissertation tackles the following challenges in synthesis of models for executable biology:

- **Choosing the level of modeling abstraction**: The type of model depends on the nature of the experimental data available and the eventual use of the model. As more data and constraints are added, we may wish to increase the level of detail of the model. On the other hand, decreasing the level of detail may help summarize alternative models in the (common) case when data are ambiguous.

- **Incorporating prior knowledge and assumptions**: We allow biologists to specify prior knowledge or assumptions in the form of partial models (or *sketches*) and constraints.

- **Joint inference from diverse experimental evidence**. Models are synthesized using different types of experimental data such as mutation experiments, time series data, or single-cell gene expression profile. Our methods are designed to handle measurement errors and sparsity in experimental data.

- **Summarizing ambiguity:** It is possible that even with the data and assumptions, the model is underdetermined. Thus, it is imperative in some cases that we succinctly summarize all possible valid models.

- **Suggesting disambiguating experiments:** When multiple possible models exist, our approaches suggest experiments whose outcomes would reduce ambiguity.

We develop computational program synthesis techniques [94, 141] to produce established models of cell signaling, such as gene regulatory networks (GRNs) and signaling pathways [78, 27, 36, 42, 59, 129, 173, 130]. These models operate at the level of concentrations of entities (such as proteins and RNA), rather than at the level of individual molecules. These models are a good match for data-driven inference because experimental methods often capture concentrations of components (e.g., RNA sequencing reveals the presence of RNA; and mass spectrometry reveals protein abundance).

Our methods synthesize models from experimental data, which we view as execution traces of the program that models the system. Experiments are *abstract* traces, in that the measurement does not capture the concentration of all proteins at all times points of the execution:

1. *Mutation experiments* can be viewed as input–output examples, where the mutation of genes provide an initial configuration as the system input, and the observed phenotype after the experiment is performed is the output computed in terms of the input [85].

2. *Time series data* obtained by phosphoproteomics mass spectrometry analyses can be abstracted into timing events corresponding to significant changes in the phosphorylation levels of proteins. These timing events can then be used as a partial order on when proteins are activated, leading to inference of how they interact with each other over time [86].

3. *Single-cell gene expression profiles* can be viewed as the states of a program, where each profile is a vector of binary (or richer) values indicating whether individual genes are expressed or not. These vectors can be viewed as the state space of the execution of a program; a model is a program that explains the transitions made between the experimentally observed states [152].

We describe our models as Boolean networks, which appear sufficiently powerful to describe these abstract traces. A **Boolean network** with binary state values can be defined by a finite set of variable states $x \in \{0,1\}^n$, a global *activation function* $F : \{0,1\}^n \rightarrow \{0,1\}^n$, where $F(x) = (f_1(x), \ldots f_n(x))$, and an *update schedule* s. The update schedule specifies in which order the local activation functions f_i are applied in each iteration of the evaluation [5].

Boolean networks constitute a flexible basis for a range of biological models. Using the same update schedule at each iteration of the evaluation leads to a deterministic execution of the BN, while non-deterministic execution can be modeled by allowing different update schedules at each iteration. The degree of non-determinism can range from full asynchrony, where the local functions can be applied in any arbitrary order throughout execution, to a fully synchronous system where each state is updated simultaneously and atomically. Other execution semantics exist in between, where nodes are partitioned into blocks that update synchronously, but blocks can be scheduled in arbitrary order. The number of states of each node can also show variation, in that the states can be binary or multi-valued. Finally, there can also be syntactic restrictions on the shape of the Boolean functions: They can range from finite state machines, to functions with 1-arity that appear in tree-shaped graph models.

Next, we give an overview of our main contributions in Boolean modeling of biological systems. For each contribution, we review a case study that motivates the choice of modeling language, in terms of the challenges posed by existing experimental data and prior knowledge. In each context, we discuss how model spaces are summarized, and how our approaches help suggest experiments to reduce ambiguity in the model space.

Chapter 2: Synthesizing Signaling Pathways from Temporal Phosphoproteomic Data

Epidermal growth factor receptor (EGFR) signaling is a well-studied mechanism, and malfunction of the EGFR pathway is linked to many forms of human cancer [172]. Emerging experimental methods provide novel insight into the EGFR mechanism in the context of different organisms or environmental conditions. New data comes in the form of high-throughput, temporal phosphorylation experiments, measuring protein phosphorylation at given discrete time points after stimulating the receptor. Phosphorylation is used as a proxy of protein activity, and is therefore used to infer how proteins behave as a response to the stimulus. Our goal is to reconstruct pathway models from experimental data and produce novel protein interaction predictions that are candidates for experimental validation.

Experimental data. We tackle this network inference problem using multiple experimental data sources. First, a large database of protein-protein interactions (the PPI network) in the form of a weighted undirected graph gives insight into the likelihood of physical, pairwise protein interactions. Second, temporal phosphoproteomics data is used to identify proteins relevant to the system being studied, and to impose a partial order on the dynamic behavior of proteins. Finally, a database of kinase-substrate interactions describes the interaction direction for some of the edges in the PPI network (Figure 2.1).

Prior knowledge and assumptions Our approach obtains a partial model for the synthesis task by applying a graph-theoretic algorithm to prune down the PPI network into a smaller, undirected network that reveals undirected interactions that are relevant to the modeled system [67]. In addition, we make the (biologist-supplied) assumption of a single, monotonic change in the phosphorylation level of a protein when interpreting the time series data. Further assumptions can be made on the network structure, such as favoring shorter path lengths, and prioritizing paths that contain certain proteins.

Modeling language. As described in Chapter 2, we infer partial orders of activity between proteins from the time series data. Partial orders, joined with the assumption that a single state change during the observation occurs per protein, imply that the semantics of signed directed graphs is a sufficient language for expressing models of protein-protein interaction. In other words, more accurate models (such as general BNs where functions show how proteins depend on each other) cannot be inferred from the experimental data and the assumption-induced constraints. These graphs are a restricted form of Boolean networks, where each node in the network (except for the stimulated source nodes) has one predecessor. The binary state of a node changes after its predecessor shows activity, in terms of the sign of the directed edge between them. The execution is deterministic, and nodes are updated in topological order.

Summarization. Due to the data being underconstrained, the network inference task typically admits a considerable number of network models, making exhaustive enumeration of solutions intractable. The solution space is summarized into the union of all signed directed trees that agree with the experimental data and the assumptions. The resulting aggregate structure shows which edges have an unambiguous direction or sign across all valid models. These high-confidence predictions are candidates for experimental validation through more specific methods that target individual interactions. We have built a tool, the Temporal Pathway Synthesizer (TPS), and applied to EGFR Flp-In HEK-293 cells stimulated with and EGF ligand, inferring a summary network of 413 edges, in which 202 edges are assigned a unique direction, and 38 edges are assigned a unique direction and sign. The directed interactions predicted in the EGFR model are supported by prior literature, and interactions that are not present in literature are supported by experimental validation through the inhibition of upstream nodes [86].

Disambiguating experiments. TPS's ability to show that an edge has the same orientation and sign in all valid models provides a means to focus on predictions that are strongly supported by experimental data. In our work, we focused on validation of high-confidence predictions for which only one of the two proteins appeared in EGF reference pathways.

Chapter 3: Synthesis of Biological Models from Mutation Experiments

The *C. elegans* vulval precursor cells (VPC) system provides an important paradigm for studying animal development, and its malfunction is linked with cancer [49, 146]. The biological problem is to understand how VPCs coordinate to determine their fate. Specifically, we want to understand which pairwise protein interactions are involved in cell fate decision, and how the same fate decision can robustly be made in a time-sensitive manner between the cells.

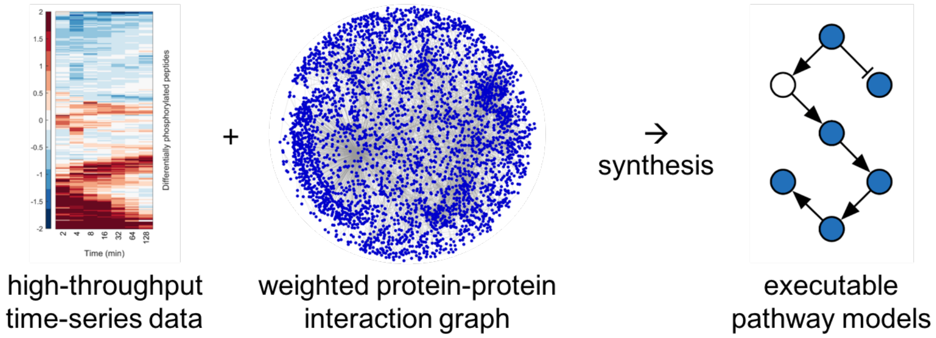


Figure 1.1. TPS takes as input multiple datasets, in the form of time series data, topological information, and additional prior knowledge about the system in the form of edge orientations. It produces all Boolean networks that satisfy all constraints imposed by input components, in the form of a summary graph.

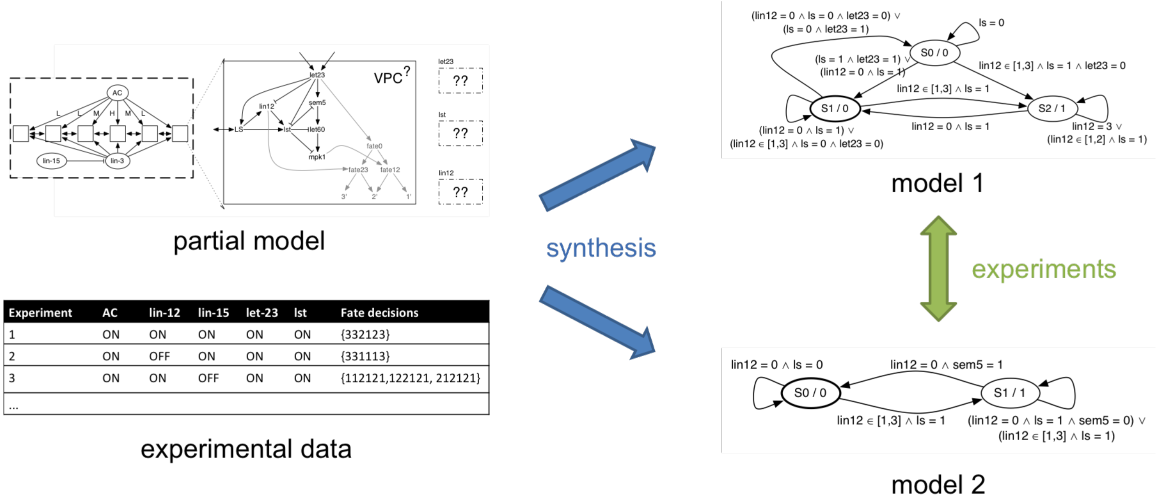
Experimental data. Over many decades, multiple mutation experiments have been performed by many labs to help uncover the signaling mechanism in the VPC system. These experiments are performed by systematically mutating combinations of genes, known to transcribe specific proteins, and then observing the fate decision made by all VPC cells. While the mutation-free, wild-type scenario always leads to the same fate decision pattern, some mutations introduce non-deterministic outcomes by leading the system to produce multiple cell fate patterns in different runs of the same initial configuration.

Prior knowledge. Some of the cell signaling events between proteins are well-established, and are part of existing literature. Meanwhile, some other proteins exhibit complex behavior, responding to the abundance of other proteins in a time-sensitive manner. Prior knowledge about the system can be modeled in terms of syntactic restrictions on the functions (e.g. which protein may influence another), as well as semantic ones (when one protein is known to activate or inhibit another). In our work, we integrate both kinds of prior knowledge.

Modeling language. The formalism for modeling the VPC system needs to be expressive enough for reproducing the non-deterministic output of the system and its time-sensitive behavior. Protein abundance is discretized into multiple levels, and each protein is represented as a finite-state machine. Activation or inhibition of a protein by another one is modeled by allowing the finite state machine to read the state of other finite state machines. Each cell is modeled as a network of proteins, which are updated synchronously. To model different rates of progress for the cells, which leads to non-deterministic outcomes, cells are scheduled to update asynchronously by a non-deterministic scheduler. We have designed SBL, a domain-specific language for expressing such models, and built SBLSynth, a tool for synthesizing models in this language from mutation experiment data and the prior knowledge given as a partial model (Figure 1.2).

Summarization. Synthesis enables a succinct characterization of spaces of valid VPC models in terms of the outcome they produce. In our work, we designed queries built on top of our synthesis approach to find out whether a given biological hypothesis (given as prior knowledge about the network topology) is consistent with current experimental knowledge (i.e. whether there exists a valid model that is a completion of the program sketch describing the hypothesis). We used SBLSynth to explore spaces of models that correspond to different biological hypotheses, and found distinct hypotheses that are consistent with existing experimental knowledge. In addition, we developed a specification minimization query to identify a minimal set of experiments that are nonredundant with respect to existing knowledge about the system. If one wants to validate experiments about the system, it is sufficient to perform the minimal set of experiments because it defines an unambiguous specification.

Disambiguating experiments. In SBLSynth, we extended model synthesis to answer additional queries to help guide the experimental procedure. Our tool searches the space of models to find alternative explanations that agree on all known experiments, but differ on a future experiment. The tool also computes disambiguating experiments, which, when performed, can help rule out alternative models (Figure 1.2).



Experiment	AC	lin-12	lin-15	let-23	lst	Fate decisions
1	ON	ON	ON	ON	ON	{332123}
2	ON	OFF	ON	ON	ON	{331113}
3	ON	ON	OFF	ON	ON	{112121,122121, 212121}
...						

Figure 1.2. SBLSynth takes a partial model and experimental data as input. It produces a completion of the partial model in the form of finite state machines. In case there are multiple models that satisfy the specification, SBLSynth can suggest a novel experiment that will distinguish between the two models.

Chapter 4: Investigating the Identifiability of Boolean Network Models from Single-Cell Data

Gene regulatory networks (GRNs) are principal mechanisms governing stem cell behavior, controlling self-renewal and differentiation of cells. They also play a crucial role in embryo development. Understanding GRNs is important for manipulating them, to generate specific cell types in the context of regenerative medicine, and also for developing targeted therapies against cancer. Single-cell gene expression measurements, coupled with algorithms to infer their temporal order, offer a high-resolution view of the temporal behavior of GRNs. We aim to use single-cell data to infer Boolean network models that capture the combinatorial relationships between genes in regulatory networks.

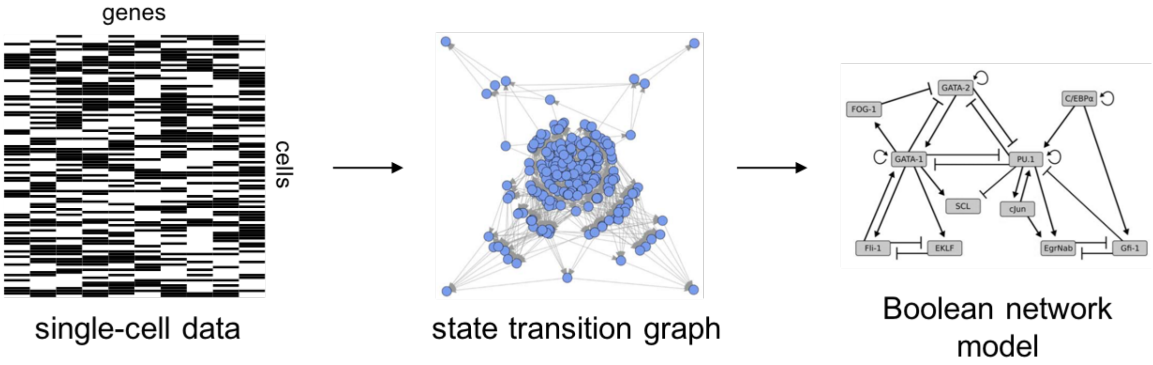


Figure 1.3. KARME takes single-cell gene expression profiles as input. It produces a state transition graph that is then used to synthesize Boolean network models of gene regulatory networks. The state transition graph and Boolean network model shown in this figure were introduced in [88].

Experimental data. There exists a wealth of single-cell experimental data sets for studying GRNs during cellular differentiation, sampling different developmental stages [169, 152, 51]. Single-cell assays profile hundreds to thousands of cells, and provides gene expression levels of individual cells, revealing heterogeneous cell subpopulations. In our work, we use *in silico* single-cell measurements obtained by the simulation of a well-established model of myeloid differentiation [88] (Figure 1.3).

Prior knowledge and assumptions. Motivated by prior examples of GRN models, Boolean functions can be restricted to a specific syntactic form. Additionally, assumptions are made on the developmental trajectories of cells through the state space: A vast array of methods from recent literature aim to reveal cell lineages by ordering individual cells along inferred trajectories [150, 145]. When viewing the single-cell measurements as a discrete state space graph where adjacency is defined by having a Hamming distance of 1, shortest paths leading to the desired final states may be preferred execution traces [159].

Modeling language. GRNs are modeled as asynchronous Boolean networks, where each single-cell measurement is viewed as one state of the network. The network is executed by non-deterministically choosing one gene update function and applying it to the current state. Each state differs from the previous one by the expression level of exactly one gene. This formalism allows capturing branching non-deterministic executions from the same initial stem cell state to distinct final stable attractor states.

Summarization. The space of all valid GRN models can be explored to find gene update functions that can only take a unique value given the experimental data and observations. Update functions that can be unambiguously inferred are high-confidence predictions that are prioritized for validation.

Disambiguating experiments. In our work, we developed an experimental design approach that identifies perturbation experiments that lead to the greatest difference between alternative models [9]. Our approach allows the customization of the spaces of candidate experiments and of the metric used to differentiate models. We demonstrate that the particular choice of the model difference metric affects the ability to identify genes that play a key role in the differentiation of myeloid cells.

1.4 Collaborators and Publications

Work described in this thesis was introduced in prior publications [85, 86], and is joint work with the following collaborators: Kirsten Beck, Rastislav Bodík, Nathan D. Camp, Dylan R. Cronin, Jasmin Fisher, Ernest Fraenkel, Anthony Gitter, Matthew E. MacGilvray, Aaron McKenna, Nir Piterman, Yewen Pu, Saurabh Srivastava, Aditya V. Thakur, Alejandro Wolf-Yadlin, Nir Yosef.

Chapter 2

Synthesizing Signaling Pathways from Temporal Phosphoproteomic Data

Advances in proteomics reveal that pathway databases fail to capture the majority of cellular signaling activity. Our mass spectrometry study of the dynamic epidermal growth factor (EGF) response demonstrates that over 89% of significantly (de)phosphorylated proteins are excluded from individual EGF signaling maps, and 63% are absent from all annotated pathways. We present a computational method, the Temporal Pathway Synthesizer (TPS), to discover missing pathway elements by modeling temporal phosphoproteomic data. TPS uses constraint solving to exhaustively explore all possible structures for a signaling pathway, eliminating structures that are inconsistent with protein-protein interactions or the observed phosphorylation event timing. Applied to our EGF response data, TPS connects 83% of the responding proteins to receptors and signaling proteins in EGF pathway maps. Inhibiting predicted active kinases supports the TPS pathway model. The TPS algorithm is broadly applicable and also recovers an accurate model of the yeast osmotic stress response.

2.1 Introduction

High-throughput proteomic assays have illuminated the amazing breadth and complexity of the signal transduction pathways that cells employ to respond to extracellular cues. In addition to quantifying protein abundance, these technologies are now routinely used to quantify protein post-translational modifications (PTMs). Mass spectrometry, in particular, offers a broad view of PTMs, quantifying various modifications such as phosphorylation, ubiquitination, acetylation, and methylation [28]. In contrast to microwestern arrays [30], reverse phase protein arrays [114], mass cytometry [17], and other high-throughput antibody-based assays, mass spectrometry is not restricted to a predefined list of proteins and can detect tens of thousands of phosphopeptides [135]. Here we show how to discover new facets of signaling cascades from complex proteomic data by integrating observed PTMs with existing knowledge of protein interactions.

Many gaps persist in our understanding of phosphorylation signaling cascades. For example, our mass spectrometry experiments show that nearly all proteins that are significantly (de)phosphorylated when the epidermal growth factor receptor (EGFR) is stimulated are absent from EGFR pathway maps. The low overlap is consistent with previous temporal phosphoproteomic studies of mammalian signaling [22, 40, 68]. Discordance between mass spectrometry studies and pathway databases is partly caused by extensive crosstalk among pathways [16] and context-specific interactions [63]. In addition, protein abundance varies greatly among human cells and tissues [81], and interactions from a pathway database are irrelevant when the proteins involved are not expressed. Moreover, perturbations and disease can rewire signaling pathways [115].

Network inference algorithms can explain the phosphorylation events that lie outside of canonical pathways and complement existing manually curated pathway maps. Specialized algorithms model time series data, which contain information about the ordering of phosphorylation changes and can support causal instead of correlative modeling [12]. Temporal protein signaling information can be used to reconstruct more accurate and complete networks than a single static snapshot of the phosphoproteome.

A complementary challenge to interpreting off-pathway phosphorylation is that the cellular stimulus response includes mechanisms that are not captured in phosphoproteomic datasets. There is an interplay between phosphorylation changes and other integral parts of signaling cascades because phosphorylation can affect protein stability, subcellular localization, and recognition of interaction partners [105]. Ubiquitination and other PTMs are not measured in phosphoproteomic studies, and not all phosphorylated proteins are detected by mass spectrometry. Additional information is required to infer comprehensive signaling cascades that include non-differentially phosphorylated proteins.

Protein-protein interaction (PPI) networks can be used for this purpose by identifying the chain of interactions that connect observed phosphorylation events. For example, MAP2K1 phosphorylation is not detected in our EGF response data, but our approach uses PPI to correctly determine that it is the kinase that controls MAPK1 and MAPK3 phosphorylation.

We present the Temporal Pathway Synthesizer, a method to assemble temporal phosphoproteomic data into signaling pathways that extend far beyond existing canonical maps. "Synthesizer" refers to our application of computational program synthesis techniques [94, 141] to produce pathway models from experimental data [47], not synthetic biology [18]. TPS overcomes both of the aforementioned challenges in interpreting phosphoproteomic data: modeling signaling events that are not captured by pathway databases and including non-phosphorylated proteins in the predicted pathway structures. The TPS workflow consists of multiple steps (Figure 2.1).

In the first step, TPS transforms a PPI graph into a condition-specific network by using mass spectrometry data to filter out irrelevant interactions. We adopt the prize-collecting Steiner forest (PCSF) [151] network algorithm to connect differentially phosphorylated proteins through high-confidence paths that may include non-phosphorylated proteins. Like nearly all existing

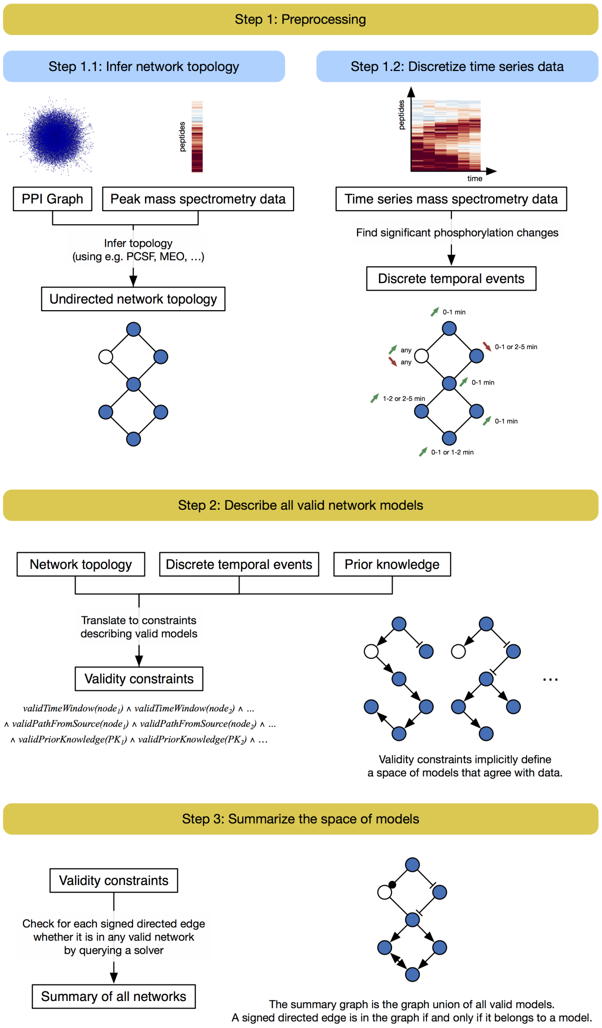

Step 1: Preprocessing
Step 1.1: Infer network topology
Step 1.2: Discretize time series data
peptides
peptides
time
PPI Graph
Peak mass spectrometry data
Time series mass spectrometry data
Infer topology
(using e.g. PCSF, MEO, ...)
Find significant phosphorylation changes
Undirected network topology
Discrete temporal events
0-1 min
any
any
0-1 or 2-5 min
0-1 min
1-2 or 2-5 min
0-1 min
0-1 or 1-2 min
Step 2: Describe all valid network models
Network topology
Discrete temporal events
Prior knowledge
Translate to constraints
describing valid models
Validity constraints
$validTimeWindow(node_1) \wedge validTimeWindow(node_2) \wedge \ldots$
$\wedge\ validPathFromSource(node_1) \wedge validPathFromSource(node_2) \wedge \ldots$
$\wedge\ validPriorKnowledge(PK_1) \wedge validPriorKnowledge(PK_2) \wedge \ldots$
...
Validity constraints implicitly define
a space of models that agree with data.
Step 3: Summarize the space of models
Validity constraints
Check for each signed directed edge
whether it is in any valid network
by querying a solver
Summary of all networks
The summary graph is the graph union of all valid models.
A signed directed edge is in the graph if and only if it belongs to a model.

Figure 2.1. TPS workflow: First, the PPI graph is combined with the phosphorylation data to obtain a condition-specific network (Step 1.1). Algorithms used in this step do not model the temporal information. Separately, the time series data are converted into discrete timed signaling events (Step 1.2). TPS then defines a space of models that agree with the data by transforming the timed events, undirected network topology, and prior knowledge (kinase-substrate interaction directions in this study) into a set of constraints (Step 2). Our system summarizes a huge solution space by computing the union of all signed directed graph models that satisfy the given constraints (Step 3). The final pathway model predicts how a subset of generic physical protein interactions coordinate to respond to a specific stimulus in a particular cellular context.

network algorithms, PCSF cannot use temporal information.

In the second step, TPS finds the orientation and sign of edges in the condition-specific interaction graph based on the order of the phosphorylation events. Phosphorylation timing is modeled separately for each observed phosphorylation site on a protein. TPS systematically explores all possible pathway models, where each model is a signed, directed graph that explains how signaling messages propagate from the stimulated source protein. In the final step, TPS summarizes the valid models into a single aggregate network that explicitly tracks ambiguous predictions. Summarization gives insight into which edges must always take a unique sign and direction across the whole solution space and enables analysis of the large number of candidate models. We created an interactive visualization tool, the Temporal Pathway Visualizer (TPV), to display the summary network alongside the temporal phosphoproteomic data (Figure 2.2).

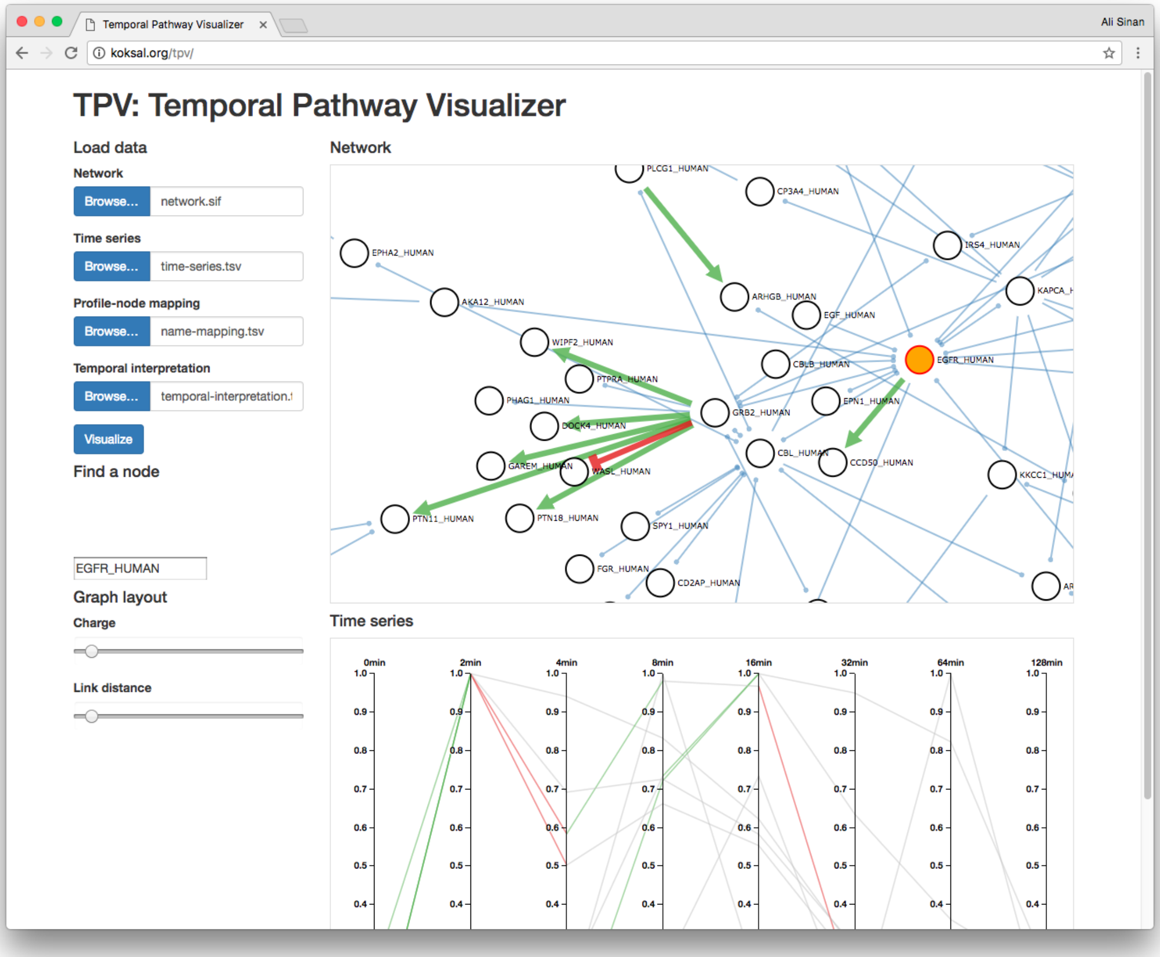
We use EGFR-mediated signaling as our primary model system for temporal phosphoproteomic and TPS analysis. TPS recovers a network that explains how EGF-responsive proteins are activated or inhibited via chains of physical interactions stemming from the EGF receptor. The highest-confidence TPS predictions are well-supported by prior knowledge and consistent with follow-up kinase inhibitions. In addition, we model the yeast osmotic stress response, recovering many of the core pathway components and predicting kinase targets that are supported by independent perturbation data. These insights into well-characterized human and yeast pathways exemplify the ability of TPS to produce condition-specific pathway maps.

2.2 Overview

As illustrated in Figure 2.1, our algorithm receives three types of input: a time series mass spectrometry phosphoproteomic analysis of a stimulus response, an undirected graph obtained by filtering a large PPI network to identify interactions that are relevant to the differentially phosphorylated proteins, and optional prior knowledge about interaction directions (for example, kinase-substrate relationships).

The undirected input graph is obtained through a static analysis in which the significantly changing proteins are overlaid on a network of physical protein interactions. A network algorithm recovers connections among the affected proteins, simultaneously removing interactions that do not form critical connections between these proteins and nominating hidden proteins that do, even if they are not themselves phosphorylated. The specific criteria used to select proteins and interactions vary based on the network algorithm. Here we use PCSF [151], but we have also successfully applied ResponseNet [164], MEO [56], and TimeXNet [113, 112] for this step.

Our method combines the input data to recover pathways embedded in the network that agree with the temporal data. TPS transforms the input into logical constraints that determine which pathway models can explain the observed phosphoproteomic data. Topological constraints stem from the filtered PPI network and require that phosphorylated proteins are



koksal.org/tpv/

TPV: Temporal Pathway Visualizer

Load data

Network: network.sif

Time series: time-series.tsv

Profile-node mapping: name-mapping.tsv

Temporal interpretation: temporal-interpretation.t

Find a node: EGFR_HUMAN

Figure 2.2. The Temporal Pathway Visualizer supports exploring a network and time series data interactively. Users can select network nodes to filter the time series view or select time series data to highlight the corresponding network nodes.

connected to the source of stimulation, such as EGF, by a cascade of signaling events. These signaling events propagate along the edges of the filtered PPI network. Temporal constraints ensure that the order of the signaling events is consistent with the timing of the phosphorylation changes. If protein B is downstream of protein A on the pathway, B cannot be activated or inhibited before A. Lastly, prior knowledge constraints guarantee that if the direction or sign of an interaction is known in advance, the pathway may not contain the edge with the opposite direction or sign. Typically, many possible pathways meet all constraints, so TPS summarizes the entire collection of valid pathways and identifies interactions that are used with the same direction or sign across all models. A symbolic solver reasons with these logical constraints and produces the pathway summary without explicitly enumerating all possible pathway models.

To illustrate this process, consider a hypothetical signaling pathway that contains a receptor node A and six other downstream proteins that respond when A is stimulated (Figure 2.3 A). We cannot directly observe the pathway structure but seek to infer it from the types of data shown in Figure 2.3 B - D. The first input is time series mass spectrometry data measuring the response to stimulating the receptor (node A), which detects phosphorylation activity for six proteins. Node B is absent from the phosphorylation data because it is ubiquitinated, not phosphorylated, by A. The second input is an undirected graph, which reveals high-confidence protein-protein interactions. These are detected independently of the stimulation condition but filtered based on their presumed relevance to the responding proteins with an algorithm such as PCSF. By combining phosphorylation data with the PPI subnetwork, this topology can recover "hidden" components of the pathway that are not phosphorylated (node B). Finally, our method accepts prior knowledge of directed kinase-substrate or phosphatase-substrate interactions, such as the edge C->D. Each of these inputs can be used individually to restrict the space of plausible pathway models. However, reasoning about them jointly produces a greater number of unambiguous predictions than considering each resource separately.

TPS exhaustively explores all signed, directed tree-structured pathway models, which are obtained by assigning signs and directions to edges of the undirected graph while restricting this space of networks through declarative constraints. These constraints are derived from the input. We next describe the constraints and how they restrict the space of models.

To formulate temporal constraints, we transform the time series data into a set of discrete signaling events (activation or inhibition) for each node, taking an event-based view of the signaling process (Table 2.1). We determine time points for each node that correspond to statistically significant phosphorylation changes. These discrete events are then used to rule out network models that contain signed, directed paths that violate the temporal ordering of these events no matter which event is chosen for each node. For example, there can be no edge from E to D in any model because D is activated strictly earlier than E regardless of whether E is activated at 1-2 min or 2-5 min. Because the time series data measures the response to a specific stimulus, we also devise topological constraints that ensure all signaling activity originates from this source. In our example, this asserts that all edges in a solution network must be on a directed path that starts at node A. Finally, our third input, the set of directed interactions,

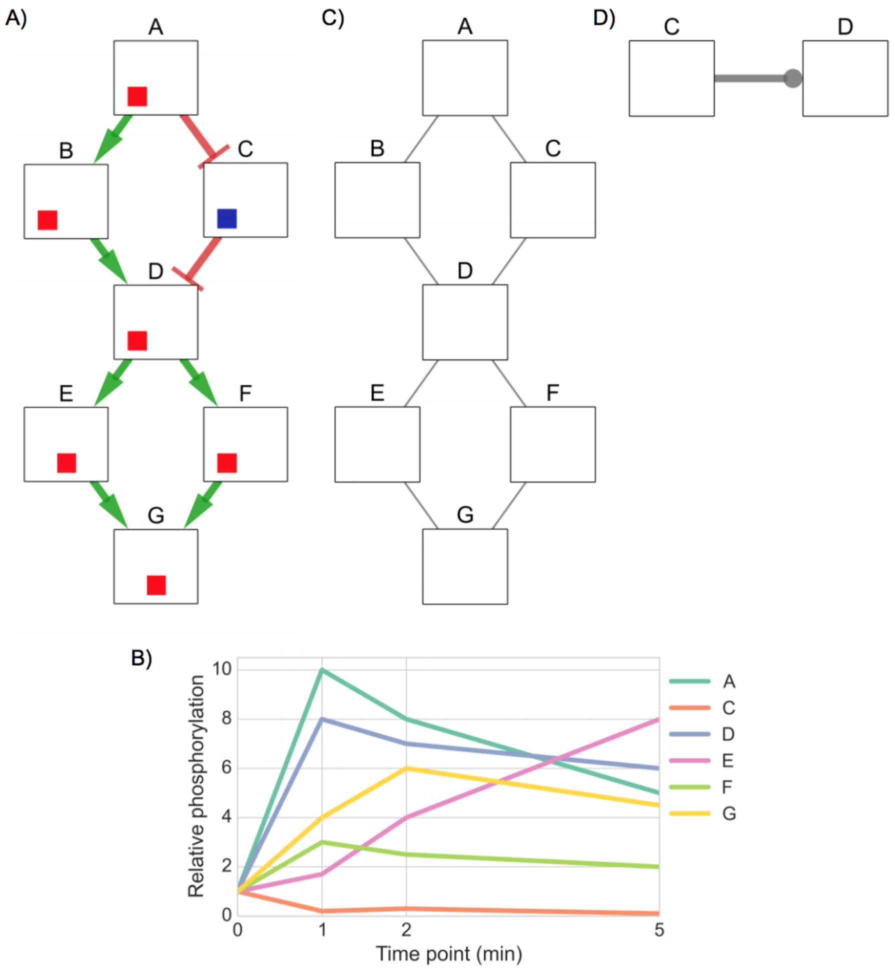


Figure 2.3. An artificial example illustrating the inputs to TPS. A) The signaling pathway that responds to stimulation of node A. The colored boxes on each node show the time at which the protein is activated or inhibited and begins influencing its downstream neighbor, with the leftmost position indicating the earlier time point. Red boxes are increases in activity, blue boxes are decreases, and white boxes are inactive time points. The left position indicates the activity at 0 to 1 min, the center position at 1 to 2 min, and the right position at 2 to 5 min. B) The first input is time series phosphorylation data of the response to stimulating node A. C) The second input is an undirected graph of high-confidence interactions that can recover hidden components that do not appear in the temporal data, such as node B. D) The last input is prior knowledge of the pathway or the protein-protein interactions, expressed as (unsigned) directed edges. We represent unsigned edges with a circular arrowhead. Here, we have one such interaction, which is from C to D.

Node	Plausible temporal signaling events
A	Activated 0-1 min
B	Activated or inhibited at any time
C	Inhibited 0-1 min or 2-5 min
D	Activated 0-1 min
E	Activated 1-2 min or 2-5 min
F	Activated 0-1 min
G	Activated 0-1 min or 1-2 min

Table 2.1. Plausible signaling events inferred for each node through a statistical analysis of the time series phosphorylation data. Although B is ubiquitinated in the 0-1 min interval, this is not observed in the phosphoproteomic input data.

requires that no model violates this prior knowledge by including an edge from D to C.

We show in Figure 2.4 the pathway models that can be learned using each type of constraint alone and by asserting them jointly. When we enforce only temporal constraints, which corresponds to reasoning locally with phosphorylation data for pairs of nodes to see if one signaling event strictly precedes another, we obtain a single precise (signed and directed) prediction from D to E (Figure 2.4 A). The topological constraints by themselves are sufficient to orient edges from the source A and from node D because D forms a bottleneck (Figure 2.4 B). The prior knowledge constrains the direction of the edge from C to D, but its sign remains unknown (Figure 2.4 C). Jointly enforcing all of these constraints has a nontrivial impact on the solution space (Figure 2.4 D). For instance, we can infer that F must activate G. If the edge direction were reversed, F would be downstream of E, but the data show that activation of F precedes activation of E. The final model that includes all available data closely resembles the true pathway structure (Figure 2.3 A). The edges incident to node B are ambiguous, and the interaction between E and G cannot be uniquely oriented, but all other interactions are recovered.

The summary for the combination of all constraints produces precise predictions that cannot be obtained by intersecting the summaries for the individual types of constraints. For instance, TPS infers that the relationship between F and G must be an activation from F to G because the sole way G can reach F in a tree rooted at A is through E, but F's activation precedes E's. This inference cannot be made by combining the models in panels A, B, and C. The simple example also highlights the differences in how the TPS constraint-based approach improves upon related methods based on correlation or the time point of maximum phosphorylation change.

2.3 Pathway Synthesis

TPS takes the undirected network of interactions produced by the PCSF algorithm and transforms it into a collection of signed, directed graphs that provide an explanation of dynamic signaling events.

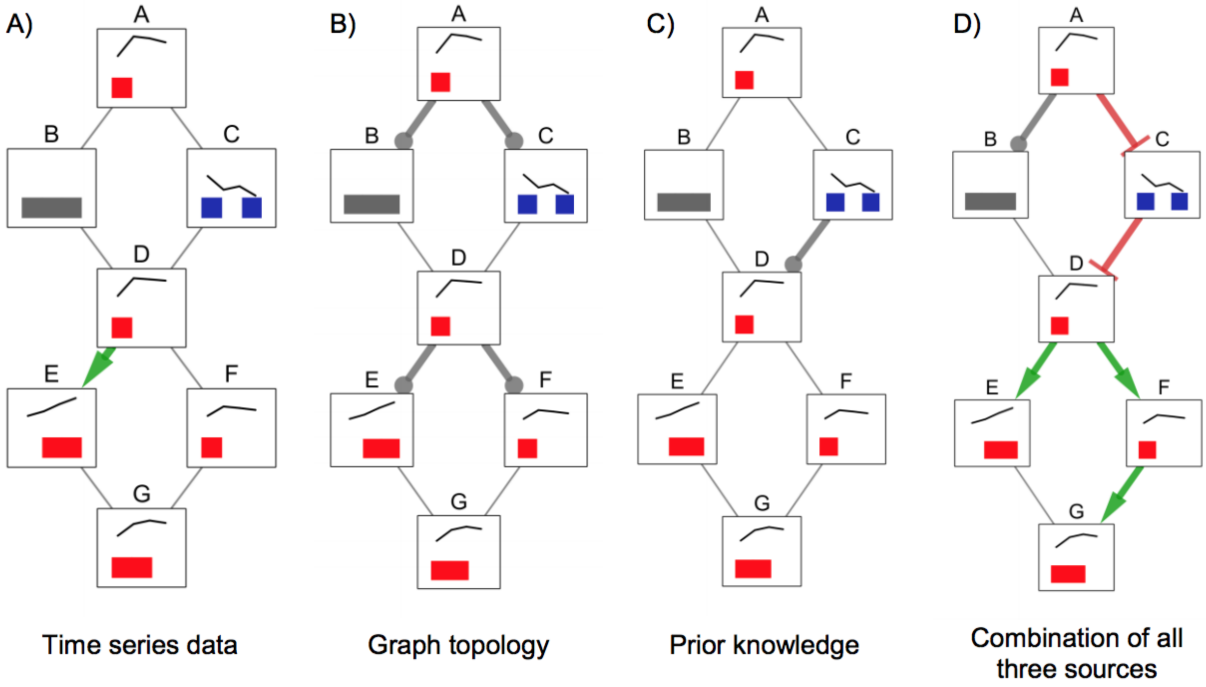


Figure 2.4. Summary graphs obtained by aggregating (via graph union) all possible signed, directed tree models for different constraints obtained from: A) time series data, B) graph topology, C) prior knowledge (in this example, kinase-substrate interaction directions), and D) all three types of input at the same time. If an edge has a unique sign and direction in a summary graph (colored green and red for activations and inhibitions, respectively), this means there are no valid models that assign a different orientation or sign to that edge. Edges that can have any combination of sign and direction in different models are gray without an arrowhead.

Discretization of Time Series Data

To find pathway models that agree with the phosphorylation dynamics, TPS first performs a discretization step that determines time intervals in which each protein may be differentially phosphorylated. The discrete set of activation and inhibition state changes is then used to rule out networks that violate the observed temporal behavior.

The transformation consists of finding time points for each profile where phosphorylation significantly differs from either the baseline (pre-stimulation) or the previous time point. In the baseline comparison, this time point is accepted only if it is not preceded by an earlier, larger change with respect to the baseline. If there is a hypothetical phosphorylation level at which the protein is activated and acts upon its downstream targets, a signaling event occurs only at the first time this threshold value is reached. This criterion does not apply when comparing to the phosphorylation level at the previous time point. In our EGF study, we use Tukey's HSD test to find significant differential phosphorylation. If comparing a time point to the baseline or the previous measurement produces a p-value below a user-defined threshold, the time point is marked as a possible activation or inhibition event depending on whether the phosphorylation level increased or decreased relative to the earlier time point to which it was compared.

As an example, when we consider the profile for node E in Figure 2.3 B, we find that both two and five minutes are time points where phosphorylation increases significantly relative to the previous time point (Table 2.1). As a result, both time points mark possible activation intervals. Even though the last measurement for node G significantly differs from the baseline, it does not constitute a possible activation because it is preceded by a larger value at 2 minutes. The hidden nodes for which there is no phosphorylation data (e.g., node B) are temporally unconstrained. They permit both activation and inhibition as possible state changes at all time intervals.

Modeling Assumptions

Characteristics of the time series data directly influence our modeling assumptions. We assume at most one signaling event happens for every node across time points. Our logical solver can explore all possible activation and inhibition events for every node, but our experience shows that the data are too ambiguous to extend our interpretation beyond one event per node when modeling a single type of stimulation (such as EGF response). We also observe that, in the absence of perturbation experiments that test the pathway behavior under different initial conditions, it is impossible to distinguish between different Boolean logic functions governing the behavior of each node (AND/OR semantics) and whether a node exhibits activity in response to one or multiple predecessors. We therefore opt for signed, directed trees as our formalism for representing pathway models because they provide a sufficient basis for explaining the dynamic system behavior under these assumptions.

Translating Input into Constraints

TPS transforms each input into a set of constraints that declaratively specify valid signed, directed tree models that agree with the data. These constraints are expressed as Boolean formulas with linear integer arithmetic, ranging over symbolic variables that represent choices on edge signs and orientations as well as how the temporal data are interpreted. The constraints can then be solved by a Satisfiability Modulo Theories (SMT) solver to find a network model that satisfies all constraints along with dynamic timing annotations for each interaction in the network.

Using constraints, we restrict the possible orientation and sign assignments to signed, directed tree networks rooted at the source node (e.g., EGF). Furthermore, constraints express how every tree model must agree with the time series data by establishing a correspondence between the order of nodes on tree paths and their temporal order of activity according to the time series data. Finally, we declaratively rule out models that contradict the prior knowledge of kinase-substrate interaction directions. In the following, we detail how such constraints are derived from the input.

A symbolic signed, directed graph that assigns a sign and direction to each edge in (a subgraph of) the undirected input graph can be represented by maintaining four Boolean variables per edge, one for each sign-orientation combination. The truth value of a variable denotes whether there is an edge with the corresponding sign and orientation in the solution network. To find a tree network model rooted at the stimulated source node, we need to constrain these truth values. First, we assert that at most one of these four variables can be true. The case where all four variables are false corresponds to the undirected edge being excluded from the solution network. Then, we assert that there are no cycles in the solution graph. To implement the acyclicity constraint, we maintain one integer-valued variable per node and assert that the integer values along all directed paths must monotonically increase. Finally, we assert that if a non-source node has an outgoing edge, it must have an incoming edge as well. This prevents modeling spurious phosphorylation changes that are not caused by the source's stimulation. These constraints together guarantee that we obtain a tree network model in which all edges are on a directed path originating at the source node.

Example 2.1 *The edge (A, B) in the undirected graph shown in Figure 2.3 C, like all other edges in the network, is translated into four Boolean variables, $activation_{A,B}$, $inhibition_{A,B}$, $activation_{B,A}$, $inhibition_{B,A}$. We ensure at most one variable is true by asserting that if one of the variables is true, the rest must be false. For instance, we assert:*

$$activation_{A,B} \implies \neg inhibition_{A,B} \wedge \neg activation_{B,A} \wedge \neg inhibition_{B,A}$$

This means that if $activation_{A,B}$ is true, the remaining variables must all be false. Both A and B have an associated integer variable, $index_A$ and $index_B$, and we state that if there is an edge from A to B, B must be assigned a greater value than A:

$$activation_{A,B} \vee inhibition_{A,B} \implies index_A < index_B$$

A similar constraint is asserted for the opposite direction. Finally, we create a constraint that requires B to have an incoming edge if it has an outgoing edge, based on its neighbors in the undirected graph:

$$\begin{aligned} &activation_{B,A} \vee inhibition_{B,A} activation_{B,D} \vee inhibition_{B,D} \implies \\ &activation_{A,B} \vee inhibition_{A,B} \vee activation_{D,B} \vee inhibition_{D,B} \end{aligned}$$

These constraints together guarantee that a valid solution must be a tree network rooted at A.

While the above constraints will ensure that solutions satisfy topological properties, they don't constrain models with respect to the temporal data. Using the temporal events computed from the time series data, TPS requires that the sequence of nodes in each signed, directed path of a tree model must be supported by a corresponding temporally ordered sequence of phosphorylation events. In the example from Figure 2.3, there can be no models that include an edge from E to D, because it is impossible for E to precede D in a directed path due to all possible activations of E being later than the possible activation of D (Table 2.1). The same example shows that the temporal ordering along paths can also have an effect beyond pairwise interactions. The sequence of nodes E, G, F cannot appear on a directed path, even though both pairwise interactions are locally consistent because E can only be activated strictly after F. Concretely, this constraint is enforced by keeping an integer-valued variable for each node, which corresponds to the choice of activation time for that node. The same is done for representing inhibitions, and we assert that at most one of the two events can occur. We restrict the values that the activation variable can take to the time points computed in the discretization step. Finally, we state that if there is an edge from A to B in the signed, directed tree, there must be corresponding choices of time points for A and B that support the interaction. An activation edge from A to B must be supported by the activation (respectively, inhibition) of A, succeeded by the activation (respectively, inhibition) of B; similarly, an inhibition from A to B requires finding an activation (respectively, inhibition) of A, succeeded by the inhibition (respectively, activation) of B.

Example 2.2 *Consider the nodes D and E in Figure 2.3. We constrain activation choices for D to be* 0 *(no activation) or* 1 *(the first time interval):*

$$activation_D = 0 \vee activation_D = 1$$

Similarly, E is either not activated or is active in intervals 2 *or* 3*:*

$$activation_E = 0 \vee activation_E = 2 \vee activation_E = 3$$

Finally, we assert that if there is an activation from D to E, both nodes must be activated or inhibited, in that order:

$$activation_{D,E} \implies activation_D \neq 0 \wedge activation_E \neq 0 \wedge activation_D \leq activation_E$$

We only show constraints for the activation of E by D through their successive activation events, the other cases are similar.

The last type of constraint that TPS enforces follows simply from the prior knowledge information. For all known kinase-substrate interactions (given as directed, unsigned edges), no pathway model can include an edge directed in the opposite orientation. This is implemented by ruling out certain values for the edge variables if data is available for a given edge. TPS currently represents kinase-substrate interactions as unsigned but could be trivially extended to treat kinase-substrate interactions as positive edges and phosphatase-substrate interactions as negative edges.

Example 2.3 *In the example from Figure 2.3, we are given the kinase-substrate interaction $C \rightarrow D$. As a result, we rule out the opposite direction:*

$$\neg activation_{D,C} \wedge \neg inhibition_{D,C}$$

Together, these constraints typically define a very large space of candidate networks that agree with the data. TPS summarizes this space without explicitly enumerating all models.

Pathway Summaries

The space of all valid pathway models with timing annotations defined by the constraints we specified is typically very large, and enumerating all models is not computationally feasible. Given an undirected network G with V nodes and E edges, along with T time points, there are 5^E ways of assigning a sign and orientation to edges of G and $(T \cdot 2 + 1)^V$ ways of assigning timing annotations to its nodes. Even for a network with 200 edges, the number of possible sign and orientation assignments is $6 \cdot 10^{139}$. TPS can reason with even larger state spaces by producing summaries of all valid pathways instead of explicitly enumerating them.

We define a summary network as the graph union of all signed, directed tree networks that satisfy the stated constraints. Timing annotations are summarized by computing the set of possible annotations for each node over all solutions. Figure 2.4 shows an example of a pathway summary obtained by computing the union of all valid models in the solution space. In this union, we observe that some edges have a unique direction and sign combination, which signifies that this was the only observed signed, directed edge between two given edges across the solution space. However, this does not guarantee that the edge between the interacting proteins must be present in all valid pathway models. Meanwhile, when there are multiple direction and sign combinations between two nodes (e.g. between B and D), we know that multiple models have a different direction or sign assignment for the pair of nodes. The fourth summary graph indicates that at least two models contain an edge between B and D in opposite directions.

We compute the summary graph by performing a linear number of SMT solver queries in terms of the size of the input graph. Each query asks whether at least one signed, directed model contains a specific signed, directed edge. These individual queries are relatively computationally cheap in practice, and we can therefore have a view of the entire solution space without enumerating all models, which is typically intractable. The summary graph over-approximates the solution space. It is not possible to recover the exact set of valid models from the summary, but only a superset of the models. This tradeoff must be made in order to analyze such a large state space.

The example summary in Figure 2.4 and the summary expansion in Figure 2.5 illustrate how this summary can contain a superset of the valid pathway models. There exists no valid model that contains an activation from A to B and an inhibition from B to D. The existence of the first edge dictates that B is activated, which implies an inhibition from B to D would decrease D's activity, contrary to what is observed in D's temporal activity profile. However, the knowledge that the edges $A - B$ and $B - D$ must have the same sign is lost through the summarization process.

For visualization and analysis purposes, pathway summaries are depicted as interactions among proteins even though the temporal consistency constraints operate at the level of individual peptides when peptide-level data are available. The protein-level summaries collapse the expanded PPI network, which can introduce ambiguities if there are interactions that are unambiguous at the peptide-level that conflict in terms of direction or sign at the protein-level. TPS is able to detect and report this loss of precision when transitioning to the protein-level network.

A final summarization observation relates to the distinguishability problem between trees and directed acyclic graphs (DAGs) that we discussed in the context of our modeling assumptions. We note that summarizing the space of all tree models as a union graph leads to the same result as summarizing the space of DAGs satisfying the same properties. This stems from the fact that for each DAG model, there exists a set of tree models whose union is the DAG. As a result, the union of all tree models corresponds to the union of all DAGs.

Using Solvers for Synthesis

TPS uses the Z3 theorem prover [37] via the ScalaZ3 interface [83] to solve the constraints it generates. It additionally provides a custom solver implemented specifically for computing pathway summaries based on data-flow analysis. The custom solver and the symbolic solver produce identical pathway summaries. However, the custom solver is much more scalable because it is specifically designed to address our synthesis task, and can handle networks containing more than a hundred thousand edges and phosphosites (Section 2.4).

To produce pathway summaries in a highly efficient manner, we devised the task of finding the union network as a data-flow algorithm that achieves 3 to 6 orders of magnitude speedup over the symbolic approach. The algorithm consists of incrementally computing the summary graph for increasing sizes of signed directed tree networks, computing a fixed point that corresponds to the summarization of the whole solution space.

Specifically, the data-flow algorithm maintains the knowledge of when a node can exhibit activity, and updates in each iterative step the time-sensitive ways of reaching a node in terms of its neighbors' activity. It starts the exploration of models at the stimulated source, and initially assumes no activity. The algorithm ends when all temporal activity information has been propagated through the graph, and all edge sign and direction assignments have been collected as a result.

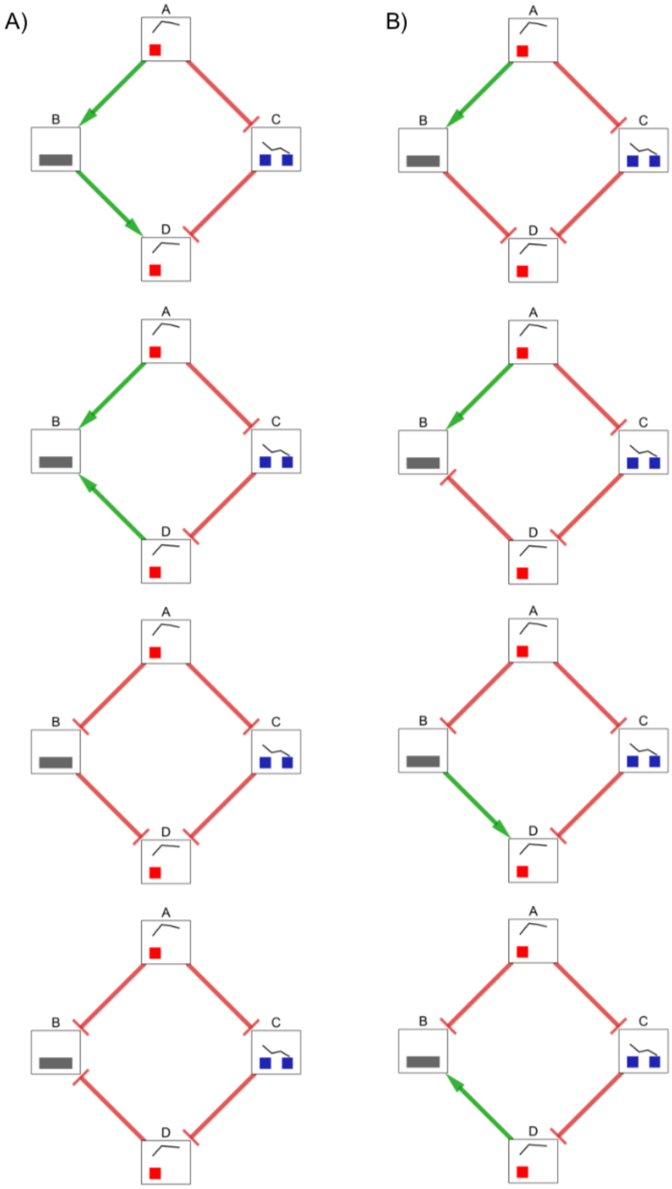


Figure 2.5. An illustration of how the pathway summary graph (Figure 2.4 D) is a generalization of the individual pathway models and can also include invalid models. Not all combinations of direction and sign assignments to the ambiguous edges yield pathways that satisfy all constraints. Here we depict only edges among nodes A, B, C, and D. For simplicity we ignore the models in which one of these edges are absent, even though these models are also included in the summarization. In some cases, B has no valid temporal activity, which we denote with the ambiguous temporal annotation (gray). A) Pathway models in which the signs of edges $A - B$ and $B - D$ are consistent with the constraint that A and D are both activated. B) Pathway models in which these edges have opposing signs and are included in the summary even though they violate a constraint.

2.4 Results

Reconstructing the EGFR pathway with TPS explains temporal phosphorylation changes

We applied TPS to model the dynamic signaling response to EGFR stimulation in EGFR Flp-In HEK-293 cells. Our workflow consists of three major steps: (1) preprocessing the protein-protein interaction network and temporal phosphorylation data; (2) transforming temporal information, subnetwork structure, and prior knowledge into logical constraints; and (3) summarizing all valid signaling pathway models to discover interactions with unambiguous directions and/or signs (Figure 2.1).

We first discretized the time series phosphoproteomic data, using Tukey's Honest Significant Difference (HSD) test [162] to determine whether a peptide exhibits a significant increase, significant decrease, or no change in phosphorylation at each post-stimulation time point. Significant phosphorylation changes can be relative to either the pre-stimulation baseline level or the previous time point. 263 peptides, corresponding to 203 proteins, significantly change at one or more time points. Second, we used PCSF to link the phosphorylated proteins to EGF, the source of stimulation, weighting proteins based on their HSD test significance. PCSF identifies a PPI subnetwork of 316 nodes and 422 edges. This subnetwork comprises the interactions through which signaling messages are most likely to propagate. Third, TPS combined the discretized temporal activities of the 263 significantly changing peptides, the PCSF network, and prior knowledge (the orientation of kinase-substrate interactions) to generate a summary of all feasible pathway models. Each type of input was translated into logical constraints, which were used to rule out pathway models that are not supported by the data.

In contrast to the reference EGFR pathway diagrams, which capture at most 11% of the differentially phosphorylated proteins, the predicted network from TPS (Figure 2.6) contains 83% of the responding proteins in its 311 nodes. Each of these proteins can be linked to the EGF stimulation with high-confidence PPI and has timing that is consistent with the temporal phosphorylation changes of all other proteins in the pathway. In addition to the phosphorylated proteins, 38 other proteins are included in the signaling pathway as hidden intermediate nodes that propagate signals via different mechanisms. Some of the differentially phosphorylated proteins may not be functional, but the TPS network provides a framework to study their role in the EGF response. The TPS pathway model includes 41 kinases and 5 phosphatases as well as adaptors and other types of proteins that coordinate with the direct phosphorylation regulators.

Like reference pathway maps, the TPS network traces the physical protein interactions used to transmit messages from EGF to the phosphorylated proteins, including PTMs and other types of interactions. These interactions are depicted as directed, signed edges in a graph, where the sign reflects that the proteins have the same (activation) or opposite (inhibition) activity changes. The timing of the phosphorylation changes supports many possible valid interpretations, and the TPS summary tracks which edges are used in different manners in different models. Of the

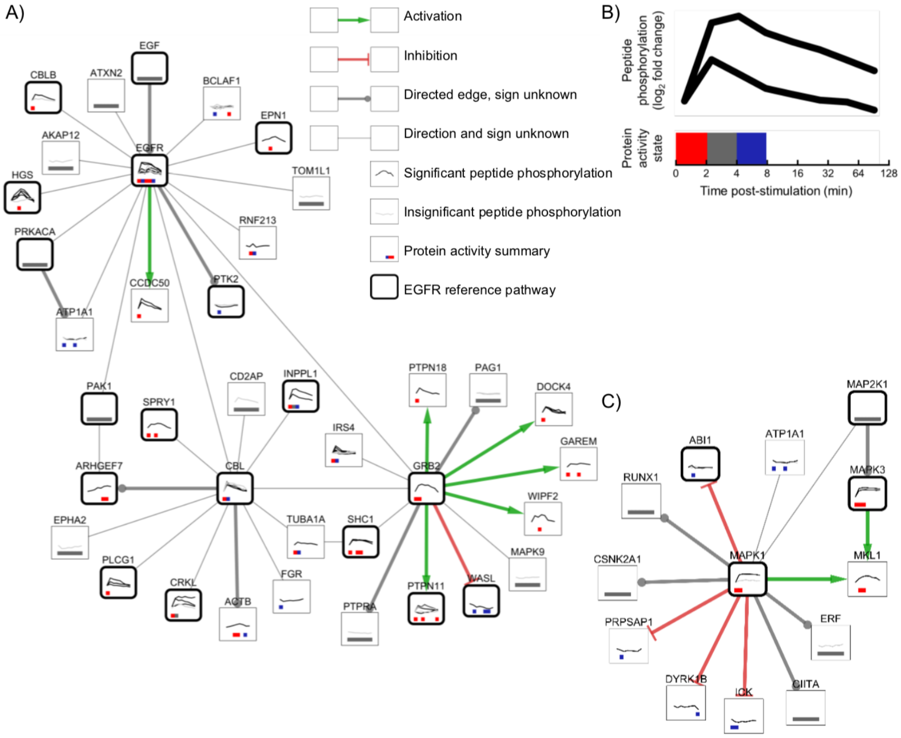


Figure 2.6. Zoomed regions of the full TPS pathway model. A) The EGFR subnetwork (EGFR, GRB2, CBL, and all their direct neighbors) depicts the proteins that first react to EGF stimulation. A substantial portion of the EGFR subnetwork (18 of 38 proteins) is known to be associated with EGFR signaling. Green and red edges depict activation and inhibition, respectively. Gray edges that terminate in a circle indicate that the interaction is used in the same direction in all possible pathway models, but the sign is ambiguous. Thin, undirected edges are used in different directions in different valid pathway models. Thick, rounded borders show which proteins are present in one or more reference EGFR pathways. Node annotations are detailed in panel B. B) Line graphs on each protein node show the temporal peptide phosphorylation changes relative to the pre-stimulation level on a $\log_2$ scale. Multiple lines indicate multiple observed phosphopeptides for that protein, where black lines denote statistically significant phosphorylation changes and gray lines indicate insignificant changes. Proteins without line graphs are connective Steiner nodes inferred by PCSF. Colored boxes summarize the TPS inferred activity state across peptides at each time point. Red indicates activation, blue inhibition, gray ambiguity, and white inactivity. C) The subnetwork surrounding MAPK1 and MAPK3. TPS uses the PPI network to correctly determine that MAP2K1 is the kinase that controls both MAPK1 and MAPK3 even though it is not observed in the mass spectrometry data.

413 edges in the network, 202 (49%) have a consistent direction in *all of the valid pathway models*, a very strong assertion about the confidence in these edge directions.

Thirty-eight of these directed edges have a consistent sign as well. The PPI connections, phosphorylation timing, and prior knowledge of kinase-substrate interaction direction all play distinct, important roles in reducing the number of valid pathway models. The timing of protein activation and inactivation in the TPS pathway reveals a rapid spread of signaling post-stimulation.

Further downstream, MAP2K1 is one of several canonical EGFR pathway members that are not phosphorylated in our mass spectrometry data but are included in the pathway. Such proteins emphasize the necessity of including PPI in the analysis of the temporal phosphorylation changes because these unobserved proteins could not be recovered by any algorithm that reconstructs the pathway from the mass spectrometry data alone. MAP2K1 is correctly recognized as the direct kinase of EGFR pathway members MAPK1 (after adding new experimental constraints described below) and MAPK3 (Figure 2.6 C). MAPK1 and MAPK3 phosphorylation levels are highly correlated and would likely be directly linked by an approach based on correlation or mutual information, but TPS correctly predicts that MAPK1 and MAPK3 correlation is due to the common upstream regulator (MAP2K1) instead. Immediately downstream of these proteins, MKL1 phosphorylation is not as strongly correlated as the two MAPKs, but TPS combines the topological constraints with the temporal information to correctly recover MAPK1 → MKL1 and MAPK3 → MKL1 [103].

Iterative experimental and computational modeling further reduces pathway ambiguity

An important feature of TPS is its flexibility to integrate different types of constraints on pathway structure. This makes it ideal for iterative modeling because computational hypotheses that are experimentally confirmed or refuted can be fed back into TPS. Based on the results from the Western blots, we added a new constraint: MAPK1 inhibits ATP1A1. We then ran TPS again, requiring pathways to be consistent with the new constraint and all previous constraints. After restricting the pathway structure, TPS correctly infers that MAP2K1 is directly upstream of MAPK1 yielding a more precise and accurate pathway. Without the Western blot-derived constraint, the direction of the MAP2K1-MAPK1 interaction was ambiguous due to the possibility that ATP1A1, rather than MAP2K1, controls MAPK1 phosphorylation changes (Figure 2.6 C). Other types of experimental corroboration can be similarly applied to iteratively improve the predictive power of TPS.

Combining multiple constraints reduces pathway ambiguity

We performed a comparative analysis of networks inferred using different subsets of constraints in order to: (1) quantify the individual contribution of different kinds of constraints in our joint

Constraints used	Directed edges	Signed, directed edges
Topo.	119	0
Temp.	27	21
PK	78	0
Topo. + Temp.	148	32
Topo. + PK	182	0
Temp. + PK	99	28
Topo. + Temp. + PK	202	38

Table 2.2. The number of edges that can be assigned: (1) a unique direction (Directed edges), and (2) a unique direction and sign (Signed, directed edges) across all valid pathway models for each combination of constraint type. In this study, "prior knowledge" refers to directed kinase-substrate interactions. The abbreviations “Topo. ”, “Temp. ”, and “PK ” correspond to topological, temporal, and prior knowledge constraints, respectively.

inference procedure, (2) show that more predictions can be made as we combine different types of data and constraints, and (3) rule out predictions that are supported by a single type of data but in conflict with other types.

Each type of constraint restricts the space of valid models in different ways and leads to different (but possibly overlapping) sets of inferred pathway edges. We compare constraint types and their combinations in terms of the set of interactions that can be assigned a unique direction (and sign) across all models. For each combination of constraints, we calculate how many interactions can be uniquely directed or uniquely signed and directed (Table 2.2). This analysis reveals the impact of different constraint types on valid pathway models. For instance, the topological constraints imposed by the input graph structure greatly contribute to the ability of TPS to infer directed edges. Meanwhile, temporal constraints are required to recover signed, directed edges. We also observe that as more constraints are added, the number of unambiguous directed (or signed, directed) interactions we recover increases. For many edges, using only one or two constraint types produces a collection of valid pathway models that conflict with respect to the direction or sign of the edge, but adding another constraint class lets TPS eliminate some of those models and make a definitive prediction. To better explain the interplay among combinations of constraints, we perform an additional analysis to identify edge predictions that can be made only if constraints are used together and conflicts that can be detected through joint reasoning (Table 2.3).

To illustrate how the combined topological and temporal constraints can exert nontrivial, non-local effects on the space of valid signed, directed networks, consider the following example. The path between YWHAG and MLLT4 cannot be oriented using any of the three constraint types alone. By joining temporal and topological properties, TPS discovers that the only way to reach YWHAG from MLLT4 goes through DBN1, which is activated later (32-128 minutes) than MLLT4 (0-4 minutes). Therefore, no network contains an edge from MLLT4 to YWHAG, and

Constraints used	Added directed edges	Added signed, directed edges	Conflicts
Topo. + Temp.	8	12	9
Topo. + PK	2	0	0
Temp. + PK	0	7	8
Topo. + Temp. + PK	7	18	9

Table 2.3. For each constraint type combination, we show: (1) "Added directed edges," or the number of edges that are uniquely directed when concurrently using all of the constraints listed in the row but not using any single constraint type, (2) "Added signed, directed edges," or the equivalent number for edges that are uniquely signed and directed, and (3) "Conflicts," or the number of edges allowed by at least one constraint type that conflict with another constraint type when the properties are concurrently asserted. Conflict edges are excluded from the final network.

the interaction can be uniquely oriented from YWHAG to MLLT4 (Figure 2.7 A).

A complex case of non-local temporal effects is observed around ABI2 (Figure 2.7 B). With either topological or temporal constraints, it is not possible to orient the two edges incident to ABI2 that link it to CCDC53 and to TRAF3IP1. Reasoning using both types of constraints, TPS narrows down the possible orientations to CCDC53 → ABI2 → TRAF3IP1. Making this inference requires looking at the temporal data for ABI2, TRAF3IP1, and TRAF3IP1's neighbor ACTB. ABI2 can be active at 0-2 or 4-8 minutes, TRAF3IP1 can be active at 0-2 or 64-128 minutes, and ACTB can be active at 4-16 or 32-64 minutes. As a result, having an edge ACTB → TRAF3IP1 implies that TRAF3IP1 is active late, and it cannot precede the activity of ABI2. There is therefore no tree network that has an edge from TRAF3IP1 to ABI2 or an edge from ABI2 to CCDC53.

The direction prediction from PTK2 to ATP2B4 (Figure 2.7 C) is inferred in two independent ways. This direction is given in the curated kinase-substrate interactions, but is also a logical consequence of combining the tree topology with time series data. The latter inference is obtained by reasoning on multiple alternative paths at once. ATP2B4 cannot precede PTK2, because all paths from the source to ATP2B4 that do not go through PTK2 must go through either SPECC1L or ARHGEF7. Both of these proteins are active strictly later than PTK2, therefore they cannot precede PTK2 in any valid model.

The same phenomenon, namely a direction prediction through two independent mechanisms, is manifested in the MAPK1 to RUNX1 interaction (Figure 2.7 D). This interaction direction is given by the kinase-substrate interactions, but the direction is independently inferred by looking at the time series data in a non-local fashion. Even though RUNX1 does not have any phosphorylation data, it cannot precede MAPK1 in any tree model because its other neighbors, ELF2 and CBFB, succeed MAPK1 in their activation.

TPS scales to large data sets

We assessed the scalability of TPS by running it on randomly generated time course data and graphs and found that TPS can scale to graphs with more than 100,000 edges and 100,000

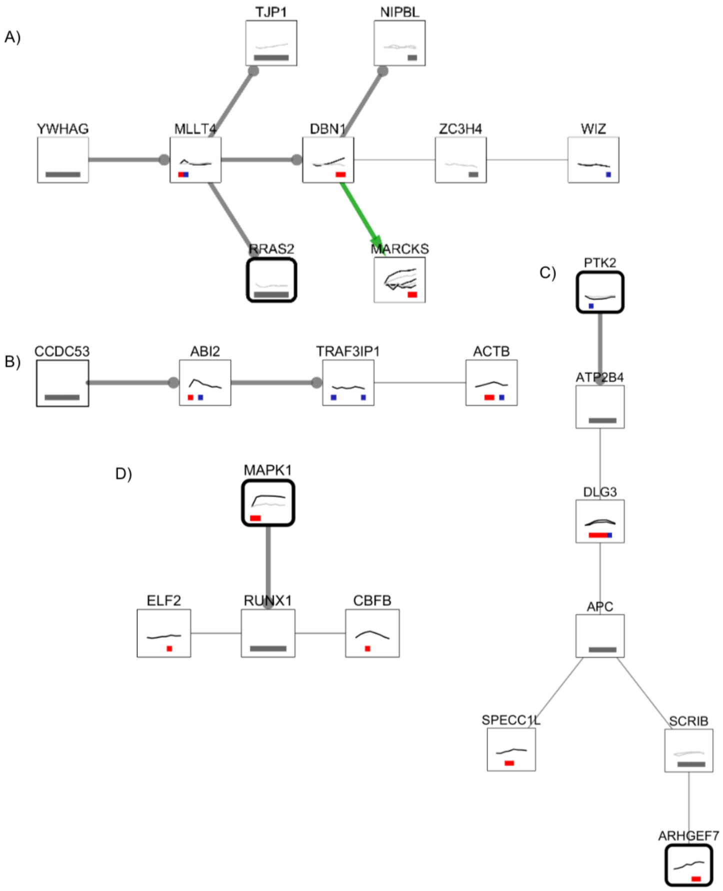


Figure 2.7. A) The subnetwork proximal to DBN1, MLLT4, and ZC3H4 shows how temporal constraints can propagate through the network to influence the direction of edges in other parts of the pathway. B) The ABI2 subnetwork (ABI2, TRAF3IP1, and their neighbors) shows how temporal constraints on one edge (TRAF3IP1-ACTB) can influence the orientation of different edges (the two directed edges involving ABI2) when performing global inference in the pathway model. C) The PTK2 subnetwork (encompassing all proteins on the path from ATP2B4 to SCRIB and their neighbors) demonstrates that SPECC1L and ARHGEF7 serve as temporal bottlenecks for paths from the source to PTK2 (these paths not shown). These proteins are activated later than PTK2, which implies that ATP2B4 cannot inhibit PTK2 at 4 min, the time it first responds to stimulation. D) RUNX1 is not observed in the phosphorylation data, but the timing of its neighbors' phosphorylation reveals that RUNX1 must be downstream of MAPK1.

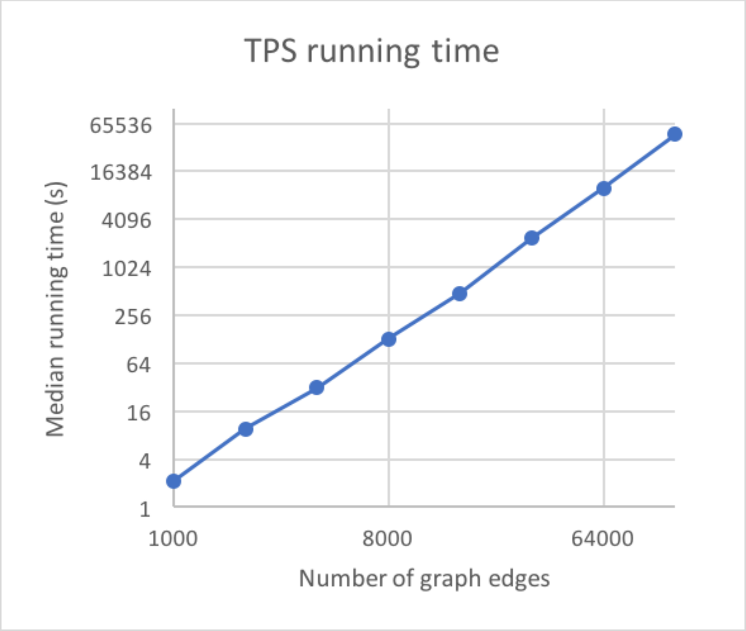


Figure 2.8. The median running time of TPS over three replicate runs against the number of input graph edges in log-log scale.

phosphosites. TPS performs similarly on our EGF case study data and randomly generated data of comparable size (graphs with 750 edges) in terms of runtime. Figure 2.8 shows the median TPS running time (in seconds) in terms of the number of input graph edges in log-log scale for three run replicates per input size. We observe that TPS takes approximately 13 hours to run on an input graph with 128,000 edges and 108,711 phosphosites.

2.5 Discussion

The pathway structure illuminated by the phosphorylated proteins in our EGFR Flp-In cells differs considerably from the simple representations in pathway databases. Interpreting signaling data requires the reconstruction of models specific to the cells, stimuli, and environment being studied. TPS combines condition-specific information, time series phosphoproteomic data and the source of stimulation, with generic PPI networks and optional prior knowledge to produce *custom pathway representations*. The predicted EGFR signaling network highlights alternative connections to classic EGFR pathway kinases and extends the pathway with interactions that are supported by prior knowledge in other contexts or kinase inhibition. Combining different constraints on pathway structure from PPI network topology and temporal information is computationally challenging, and we identify predictions that can be obtained only through joint reasoning with all available data.

Tradeoffs between ambiguity, expressiveness, and correctness

The modeling assumptions made when interpreting and translating biological data into logical constraints have complex effects on the degree of ambiguity, expressiveness, and accuracy of the resulting pathway summary. Even with temporal information, many pathway structures can explain the ordered signaling events. This motivates the reduction of ambiguity with hard logical constraints, where each constraint is fully trusted, instead of with probabilistic constraints [65, 77], where a constraint can potentially be violated.

In the PPI network, we allow paths only through chains of experimentally detected PPI. In settings where the PPI network is less complete, we could include edges among highly correlated phosphorylated proteins or predicted interactions based on protein sequence, protein structure, pathway connectivity, or literature mining [89, 101]. The pre-processing step that filters the PPI network operates on a weighted network. These additional edges could be assigned lower weights so that PCSF includes them in the TPS input network only if they are critical for connecting significantly phosphorylated proteins. This would reduce the impact of missing interactions on TPS pathways at the cost of potentially increasing ambiguity because there would be more possible paths through which signal can flow.

Likewise, we observe that some proteins, such as RAS and RAF family members, are not included in the TPS pathway because our mass spectrometry data do not detect their phosphorylation. To increase robustness to potential false negatives in the mass spectrometry, the input PPI network could be modified to include edges from relevant reference pathways with high weights (similar to [113]) so that PCSF prefers to include these interactions instead of other high-confidence connections in the PPI network. The weight of these prior knowledge edges would control the tradeoff between condition-specific *de novo* pathway discovery and conformance with prior knowledge.

Unlike single-cell mass cytometry data, where the peak activity times of a small number of phosphoproteins can be resolved precisely [87], phosphorylation timing in cell population-level mass spectrometry data is inherently ambiguous. Therefore, instead of rigidly determining a protein's time of activity by selecting the time point at which the greatest phosphorylation change is observed, TPS takes a more general approach. It allows a protein to be activated or inhibited whenever the phosphorylation significantly differs from the level before stimulation or at the immediately preceding time point as long as it is the *first time* at which that phosphorylation level has been observed. We focus on the initial pulse of signaling activity following stimulation, sampling more early time points in our EGF response study because we are more confident that these changes in phosphorylation intensity are due to PTMs instead of changes in protein abundance. Feedback loops cannot be detected when learning a single activation or inhibition time per peptide, a modeling decision we made in our three case studies. However, the TPS framework makes it possible to allow multiple activity changes per peptide in future applications. Statistical tests of the temporal phosphorylation profiles could determine the number of significant activity changes for each peptide. Then, TPS could search for pathway structures with feedback loops that explain the multiple activation or inhibition events per peptide.

Lastly, we recognize that different phosphopeptides on the same protein can have different phosphorylation changes over time, and we allow each peptide to have its own activation times instead of forcing a single time per protein. This decision can lead to ambiguous edge direction predictions at the protein-level even when the directions are consistent at the peptide level. For example, DOCK1 interacts only with BCAR1, yet the direction and sign of the interaction are ambiguous. The uncertainty arises because BCAR1 is phosphorylated on both Y249 and Y387. TPS correctly concludes that the sign cannot be determined because one site could activate DOCK1 and then feed back and affect the other BCAR1 site.

Contrasting TPS with related computational approaches

TPS provides a new way to integrate information from PPI networks, time series phosphoproteomic data, and prior knowledge by introducing a powerful constraint-based approach to build on concepts previously explored by related algorithms. Approaches for building networks from gene expression data alone (reviewed in [38]) can be applied to phosphoproteomic data as well. Extensions of these methods for temporal data introduce time lags and search for dependencies between genes' expression levels over time [176]. Methods based on Granger causality [96] identify proteins whose phosphorylation predicts behavior at later time points and provide one type of causal model. However, all methods that rely on the phosphorylation data alone [62] will miss critical signaling pathway interactions because not all pathway members have observed phosphorylation changes.

Algorithms based on gene and protein perturbations provide an alternative approach toward causal models. Transcriptional regulatory networks have been inferred from expression changes induced by gene knockouts and knockdowns [4, 95, 156, 163]. Likewise, signaling networks have been reconstructed by stimulating a pathway and perturbing signaling nodes with kinase inhibitors or RNA interference. Protein activities are observed with antibody-based assays, and pathways are recovered *de novo* [31, 52, 80, 99] or by adapting prior pathway knowledge [100]. The PHONEMeS method is unique for its ability to handle large-scale phosphoproteomic perturbation data [147].

The HPN-DREAM network inference challenge [64] spawned several new approaches for analyzing time series phosphoproteomic data in multiple biological contexts. Participants predicted signaling pathways from *in silico* time series data and temporal reverse phase protein array data for approximately 45 phosphoproteins in four breast cancer cell lines under various stimuli and inhibitor treatments. In contrast, TPS focuses on reconstructing signed, directed signaling networks from large-scale phosphoproteomic data. The TPS networks rely on physical protein-protein interactions and include proteins that are not observed in the mass spectrometry. PropheticGranger [23], the top performer in the HPN-DREAM experimental task, demonstrated the importance of prior knowledge in network inference and modified the standard Granger causality approach to assess dependencies between observed proteins. Meanwhile, TPS uses time series information to globally reason about temporally consistent network models, ensuring that all paths in a network agree with time series data and considering the temporal activities of nodes that are not direct neighbors in a path.

In our EGFR study, the TPS PPI subnetwork input is provided by PCSF, but other network algorithms can also connect phosphorylated proteins using PPI. A related algorithm interpolates between globally optimal (Steiner tree) and locally optimal (shortest path) connections to different proteins [170], and this method has been applied to link functional signaling proteins derived from phosphoproteomics data [126]. Many other approaches connect source and target proteins in a PPI network to identify pathways. ResponseNet [164] does so with a maximum flow formulation; SHORTEST [137] and PathLinker [122] use shortest paths; Maximum Edge Orientation (MEO) [56] orients the undirected edges to produce short, directed paths. Integer programs can express complex optimization preferences with multi-stage objective functions when predicting source-target connections [26, 93]. The predicted networks from any of these methods can be used as input for temporal analysis with TPS.

Among methods that integrate dynamic data and PPI networks, TPS is unique in its ability to assess and summarize all possible pathway structures that are consistent with the input network and the temporal constraints. TPS also considers all possible temporal activations for each peptide instead of mapping proteins to temporal bins in advance like TimeXNet [113, 112]. Similarly, Budak et al. use time point-specific PCSF networks to map proteins to times [21], and TimePath assigns genes to transcriptional phases based on gene expression timing [70], and Khodaverdian et al. explore theoretical properties of temporal Steiner trees [79]. The Signaling and Dynamic Regulatory Events Miner (SDREM) models temporal gene expression to infer the timing of transcription factor activity, but the pathway discovery phase does not use any temporal information [55, 57]. Vinayagam et al. used temporal phosphorylation to evaluate their predicted PPI directions but did not consider dynamics when making the predictions [154]. Time series data and interaction networks have also been combined for inferring protein complex dynamics [111], pathway enrichment [73], and related problems reviewed in Przytycka et al. [118].

The key difference between our work and other declarative computational approaches is that TPS operates on networks that are several orders of magnitude larger and summarizes very large solution spaces defined by sparser and less precise experimental data. Model checking and symbolic reasoning have been used to verify properties of manually constructed biological models [46], complete partially specified pathways using perturbation data [84], and synthesize gene regulatory networks directly from data [41, 98] (reviewed in [47]). In addition, other types of declarative approaches, such as integer programming [21, 26, 70, 108, 133, 137] and answer set programming [60], have been applied to biological pathway analysis. The TPS model summarization strategy, which makes it applicable to comprehensive signaling networks containing more than a hundred thousand edges and phosphosites, sets it apart from these related methods (Figure 2.8).

Future directions in pathway synthesis

TPS offers a powerful framework for combining multiple types of declarative constraints to generate condition-specific signaling pathways. The constraint-based approach can be extended to include many additional types of data. New types of constraints could be derived from high-level properties that proteins, interactions, or pathways must satisfy. Future versions of TPS could incorporate perturbation data that links kinase inhibition or deletion to phosphorylation changes that are far downstream from the kinase. For instance, both temporal [76] and kinase perturbation [93, 124] phosphoproteomic data are available for the yeast osmotic stress response. Modeling multiple related conditions (e.g., different ligand stimuli and inhibitor perturbations) could allow TPS to learn not only the signs of interactions but also the logic employed when multiple incoming signals influence a protein. Finally, TPS could accommodate user-defined assumptions or heuristics about pathway properties, such as restrictions on pathway length. Such complex constraints cannot be readily included in existing optimization-based approaches like dynamic Bayesian networks or TimeXNet.

As proteomic technologies continue to improve in terms of depth of coverage [135, 174] and temporal resolution [68, 76, 119], the need to systematically interpret these data will likewise grow. TPS enables reasoning with temporal phosphorylation changes and physical protein interactions to define what drives the vast protein modifications that are not represented by existing knowledge in pathway databases.

Chapter 3

Synthesis of Biological Models from Mutation Experiments

Executable biology presents new challenges to formal methods. This chapter addresses two problems that cell biologists face when developing formally analyzable models.

First, we show how to automatically synthesize a concurrent in-silico model for cell development given in-vivo experiments of how particular mutations influence the experiment outcome. The problem of synthesis under mutations is unique because mutations may produce non-deterministic outcomes (presumably by introducing races between competing signaling pathways in the cells) and the synthesized model must be able to replay all these outcomes in order to faithfully describe the modeled cellular processes. In contrast, a "regular" concurrent program is correct if it picks any outcome allowed by the non-deterministic specification. We developed synthesis algorithms and synthesized a model of cell fate determination of the earthworm *C. elegans*. A version of this model previously took systems biologists months to develop.

Second, we address the problem of under-constrained specifications that arise due to incomplete sets of mutation experiments. Under-constrained specifications give rise to distinct models, each explaining the same phenomenon differently. Addressing the ambiguity of specifications corresponds to analyzing the space of plausible models. We develop algorithms for detecting ambiguity in specifications, i.e., whether there exist alternative models that would produce different fates on some unperformed experiment, and for removing redundancy from specifications, i.e., computing minimal non-ambiguous specifications.

Additionally, we develop a modeling language and embed it into Scala. We describe how this language design and embedding allows us to build an efficient synthesizer. For our *C. elegans* case study, we infer two observationally equivalent models expressing different biological hypotheses through different protein interactions. One of these hypotheses was previously unknown to biologists.

3.1 Introduction

Diseases can be caused by perturbed gene and protein regulatory networks. For example, disease X may be related to the levels of proteins P and R, and P may negatively regulate R. Once the level of P is decreased, high levels of R may cause disease X. To avoid the disease, we may want to increase the level of P. One way to infer protein regulatory networks is to carry out *mutation experiments*, in which cells are genetically modified to suppress or enhance the activity of a certain protein, leading the cell to exhibit abnormal behavior such as uncontrolled cell divisions. If, by suppressing the activity of protein P, the resulting phenotype can be attributed to, say, an increased activity of a known protein R, we can infer from this mutation experiment that P negatively regulates R. From many such inferences, experimental biologists deduce regulatory networks that describe the causal events leading to specific cell fates and other behaviors.

Experimental biologists are concerned about the correctness of their models that give a dynamic explanation of how the observed outcomes are produced. Executable biology [45] addresses this concern by building executable models that can be verified against performed experiments. Treating cells as concurrent agents models the fact that cells do not evolve at synchronous rates [74, 75]. Verification ensures that a concurrent model is correct for all variations of cell growth rates, by exploring all possible executions of the model [48].

Unfortunately, turning informal maps of regulatory networks common in biological literature into executable models is laborious because it involves explicitly defining timing delay and strength of how multiple proteins regulate each other. In our previous work, some of us developed a model of vulval cell fate specification (i.e. how vulval cells make the decision to develop into a particular cell type) in the *C. elegans* worm [50]. This model correctly predicted an unknown protein-protein interaction, however it took several months to tweak the details of the model before it was verified against the experimental data. Whenever new experiments are added, or when the model is extended with new components, similar tweaks are required.

This chapter develops techniques for synthesizing executable models from experimental observations and prior biological knowledge. Two challenges make this synthesis problem interesting. First, the outcomes of some cellular systems, such as fates of stem cells, are non-deterministic. For example, in the *C. elegans* system that we study, some mutations cause the six observed vulval precursor cells (VPCs) to acquire one of two alternative fates, presumably due to races in the communication among cells. The desired executable model must be able to reproduce all the observed behavior in order to be correct. We synthesize concurrent cell models such that, for each observable outcome, there exists a schedule that leads the model to produce the outcome. This requirement makes our synthesis task a new problem, which is more complex than what has been previously addressed.

Second, the incomplete set of mutation experiments forms only a partial specification. Because only certain genes are mutated from the total combinatorial set of possible mutations, we cannot be certain that an executable model that verifies against these mutations, whether it is synthesized or manually constructed, is the sole explanation of the cellular regulatory process. This is because there could exist an alternative model that is observationally identical on the current specification but observationally distinct on an additional mutation. Finding such an additional mutation would uncover *ambiguity* in the current specification.

To confirm that we have synthesized a unique model, we go beyond synthesis and develop methods for the analysis of the space of plausible models, i.e., models that agree with the specification. If observationally distinct models exist, we suggest a new mutation that differentiates them. If no alternative models exist, we determine the smallest set of experiments that is sufficient to arrive at the unique model. Finding such a minimal set is interesting because, should biologists decide to redo the experiments for validation, they only need to perform the experiments that suffice to synthesize a unique model. Finally, it is interesting to ask whether there are observationally identical but internally different models. Such models present regulatory networks where the network function is "implemented" via different protein interactions. These models cannot be distinguished by observing phenotypes; we must, say, instrument proteins with fluorescent markers (similar to tracing the program) and observe the cell during its development. This is a harder experiment, but the cost of instrumentation is reduced with the help of formal methods, as we can identify which genes to mark given the internal differences between the observationally identical models.

We have built an efficient verifier, synthesizer and specification ambiguity analyzer that implements algorithms for the analyses described above. Our synthesizer takes as input the mutations, the results observed under mutations and a template structure of the cells, and from them it generates a verified model. The template of the cell defines the cell components, and a superset of their interconnections (inhibition, activation), allowing biologists to formally state existing knowledge on the system being modeled. Additionally, the granularity of the discretized concentration levels for each component is set a priori. What we synthesize is the internal logic and timing of the components, i.e., how their concentration changes in terms of their incoming signals, and we therefore off-load the most difficult task of systems biology modeling to a computation search engine.

This chapter makes the following contributions:

1. We designed SBL, a domain-specific language for expressing our models using an execution model with restricted asynchrony called bounded asynchrony [48]. We embed SBL into the Scala programming language [107] and build a lightweight synthesizer, which is publicly available [1]. We describe how to translate SBL programs into formulas in order to solve synthesis and specification analysis problems (Sections 3.3 and 3.4).

2. We formulate the verification problem (Section 3.5) and the program synthesis problem (Section 3.5). We observe that unlike previous synthesis tasks, e.g., concurrent synthesis [138] or synthesis from examples [58] or invariants [143], which are expressed as formulas with two levels of quantification (2QBF), this problem is expressed as a formula

with three levels of quantification (3QBF), which makes it a new kind of problem. We develop efficient algorithms for solving this problem that reduce to three communicating SAT solvers.

3. We develop methods for analyzing the specifications and the space of plausible models (Section 3.5): We describe algorithms for determining whether internally or externally distinguishable models exist, and for finding minimal non-ambiguous specifications. These algorithms build on our 3QBF synthesis procedure, and can potentially guide new wet-lab experiments by computing mutation experiments that disambiguate alternative models.

4. We evaluate our framework by describing that it efficiently (1) generates valid models for the *C. elegans* VPCs. The model fixes a bug in previous modeling, an incorrect modeling of a component's behavior when it is mutated; (2) shows that no behaviorally distinct models exist (even after extending the experiment space to consider mutations for each component in the VPCs), but two internally different models were synthesized, one of which expresses a previously unknown biological hypothesis; and (3) prunes the specification from forty-eight mutation experiments to a minimal set of four experiments (Section 3.6).

3.2 Overview

This section presents an overview of the program synthesis and specification analysis methods we have developed for modeling biological systems. We describe how scientists typically conduct mutation experiments to infer informal genetic regulatory networks, discuss how these models can be formalized, present our programming language for expressing, verifying and synthesizing formal biological models, and outline our synthesis and specification analysis algorithms for programs in this language.

Background on Mutation Experiments.

Here we give a brief background on mutation experiments, in the context of developmental systems biology. The role of these experiments is to understand cellular genetic regulatory networks, in particular those that control stem cell differentiation. These regulatory networks are of interest in part because their failure may trigger disease:

> Cancer is fundamentally a disease of failure of regulation of tissue growth. In order for a normal cell to transform into a cancer cell, the genes which regulate cell growth and differentiation must be altered. (Wikipedia)

Hence, to understand cancer, one needs to understand cell differentiation. There are two common mechanisms for cell differentiation: (1) a single cell divides into cells of different types based on the asymmetric accumulation of substances inside the cell; and (2) multiple identical cells differentiate by mutually communicating with the goal of arriving at coordinated fates [43]. We focus on the second mechanism, and aim to mechanistically explain cell differentiation by modeling intercellular communication.

The specific goal of developmental biologists is to infer the program that stem cells "execute" to agree on their fates. This program executes within one cell division cycle during which a pluripotent cell decides its fate, potentially by communicating with other cells.

One method for inferring this program is to mutate a set of genes in the cell and observe the resulting changes in the cell development. These experiments are particularly attractive because phenotype changes resulting from the cell taking a different fate are visually observable, avoiding the need for the more expensive tracing of temporal protein levels by the means of tagging cell proteins with fluorescent genes.

From gene mutation experiments, biologists infer protein interactions, namely which proteins are activated or inhibited by the mutated protein. For example, Yoo *et al.* [165] infers:

> In this assay, depletion of [genes] lst-2, lst-3, lst-4, or dpy-23, as well as ark-1, caused [a phenotype change, namely] ectopic vulval induction, suggesting that they function as negative regulators of the EGFR-MAPK [protein] pathway [due to the phenotype change being linked to inhibition of the pathway].

Biologists unify such piecemeal information to create informal models of cellular programs, such as the one in Figure 3.1 from [50]. This model shows how five cells—an anchor cell (AC), three vulval precursor cells (VPC), as well as the hyp7 cell—communicate to determine the fate of the VPCs. Each VPC contains the same set of components, which is composed of receptors (let-23 and lin12) and proteins (lst, sem-5, let-60 and mpk-1). The edges between these components show the activation ($\rightarrow$) vs. inhibition ($\dashv$) relationships between them.

While these informal models may capture all known interactions among cell components, they do not describe the dynamics of the cell, such as what race conditions permit the cells to take non-deterministic fates that have been observed under some mutations. Due to this lack of dynamic information, one cannot be certain that these diagrams accurately describe the cell fate specification mechanism.

Executable Biology

The goal of *executable biology* [45] is to create executable models that allow the observation of the dynamic behavior of biological systems. Furthermore, these models are verified against experimental observations. For concurrent discrete models, verification, say, with model checking, ensures that all executions of the model agree with the observed outcomes. By verifying

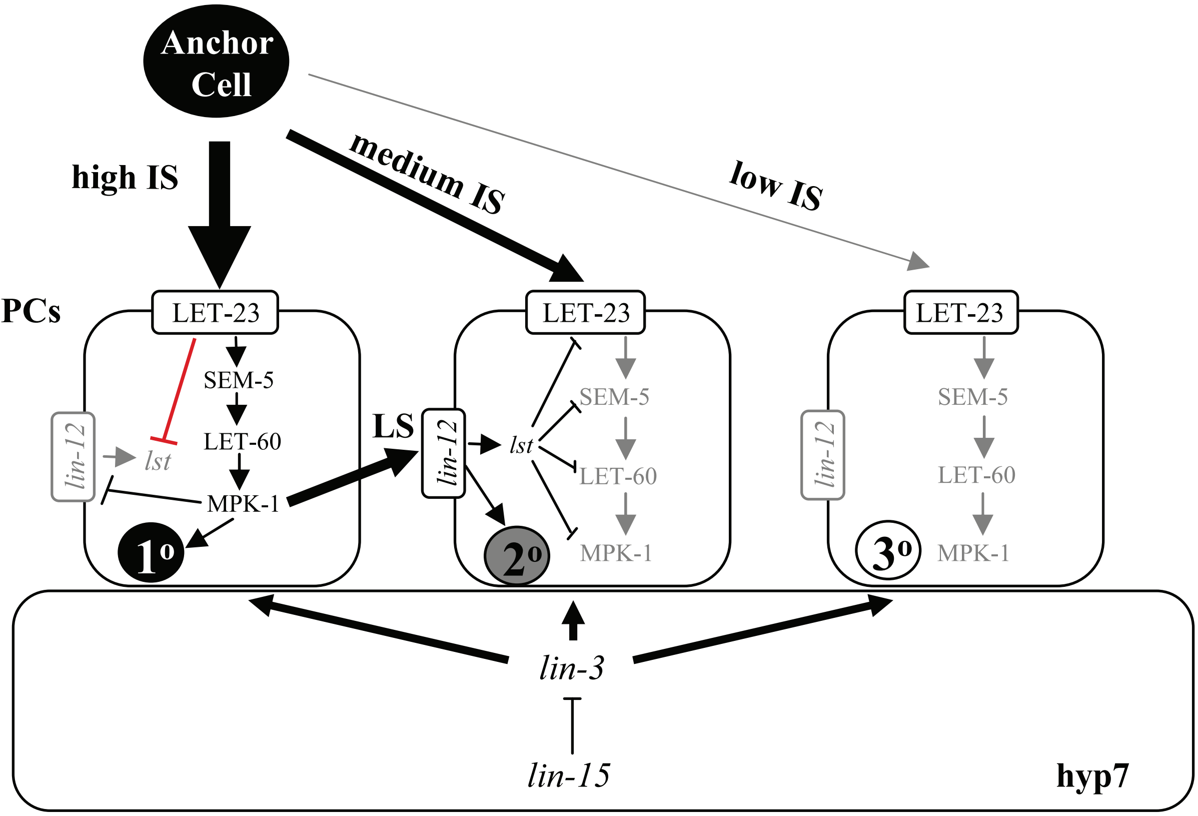


Figure 3.1. An informal diagram of cell fate specification in a system of three VPC cells [50]. These cells react to the inductive signal (IS) from the anchor cell and communicate among themselves using the lateral signal (LS) to decide one of three fates.

the program under the non-deterministic interleaving of cell steps, we ensure that a program is a faithful model of a cell system where cells may progress at varying rates[1] [75, 48].

It is challenging to create verifiable, concurrent models of communication between cells. To transform the informal model in Figure 3.1 into an executable model, the designer must model (1) protein levels; (2) timing delay or rates at which proteins react with other components; and (3) how a protein behaves when both an activator and an inhibitor of the protein are active. We have previously developed a verified model of *C. elegans* VPC cells; that model took several months to develop [50]. This chapter develops methods for automatically synthesizing executable models of concurrent cellular systems.

[1] Another way to model varying cell rates is to use stochasticity. In stochastic models [7, 97], this non-determinism takes the form of protein models making probabilistic transitions, accounting for variability of protein level change rates in nature. However, moving non-determinism from protein modeling into the scheduler allows protein models to be deterministic, which in turn enables discrete verification techniques.

Non-deterministic experiment outcomes. A mutation experiment may produce different outcomes when run repeatedly. A correct model must reproduce *all* non-deterministic outcomes of a given mutation experiment. We synthesize concurrent cell models that satisfy this requirement by ensuring that each outcome that must be observed is reproduced by the model under some interleaving of cell steps.

For biological reasons, we use a restricted model of concurrency, *bounded asynchrony* [48]. Because neighboring cells always advance at relatively similar rates, rather than at arbitrary speeds, fully asynchronous models are too unconstrainted to reproduce the observation in certain mutation experiments. One-bounded asynchrony is one way to achieve restricted asynchrony, ensuring that between two execution steps of a cell, no other cell can take more than two steps.

Because of the requirement to reproduce all possible outcomes, model synthesis in this setting is a more complex synthesis task than what has been previously addressed. In this chapter, we advance the state-of-the-art in solving this new synthesis problem.

Modeling Language

We have developed a high-level programming model, SBL, inspired by biological diagrams such as the one in Figure 3.1. SBL introduces programming abstractions for cells, cell components, and interaction between components.

Programs in SBL (Figure 3.2) are composed of cells, which execute according to a schedule s that adheres to the *1-bounded-asynchrony* constraint. The schedule is of bounded length; the number of steps in the schedule corresponds to the desired discretization of the cell division cycle. Multiple cells can take simultaneous steps. Cells are composed of components, which model proteins or cell receptors. Components communicate with other components in the same cell or in other cells; communicating components are connected with directed edges, which correspond either to activation or inhibition relationships. Components of a cell execute synchronously; all take one step when the cell is scheduled. Components have state—a discretized concentration—usually modeled at 2-5 levels. When the component executes, it updates its next state based on its current state and the states of its activators or inhibitors. Each component is modeled with an update function $(L, L^k) \to L$, where L are levels and k is the number of components activators and inhibitors, combined.

Thanks to these abstractions, SBL programs are syntactically smaller compared to models expressed in the Reactive Modules language [3], which was the modeling language used in earlier work [50]. As a result, we are able to develop efficient synthesis algorithms for programs in SBL.

Example 3.1 *To illustrate, we consider the problem of designing a simple distributed protocol. Mutations in this setting correspond to environment effects on the system being designed, and the specification consists of input-output pairs defining the desired behavior for given environments.*

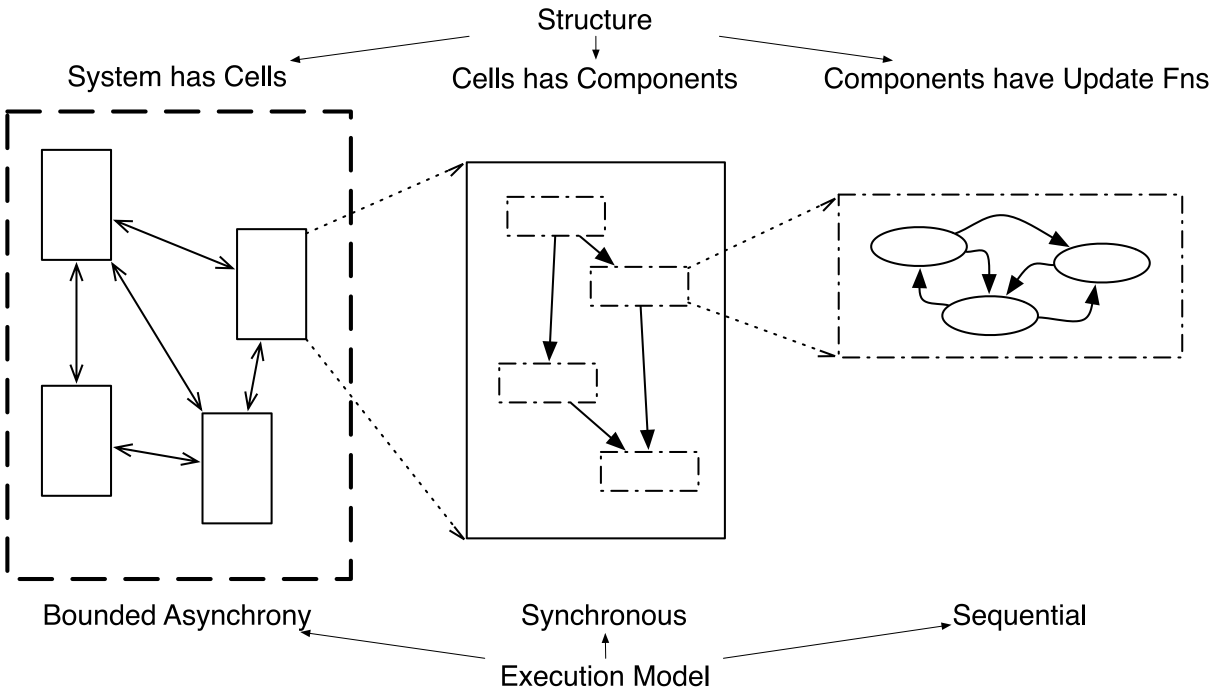
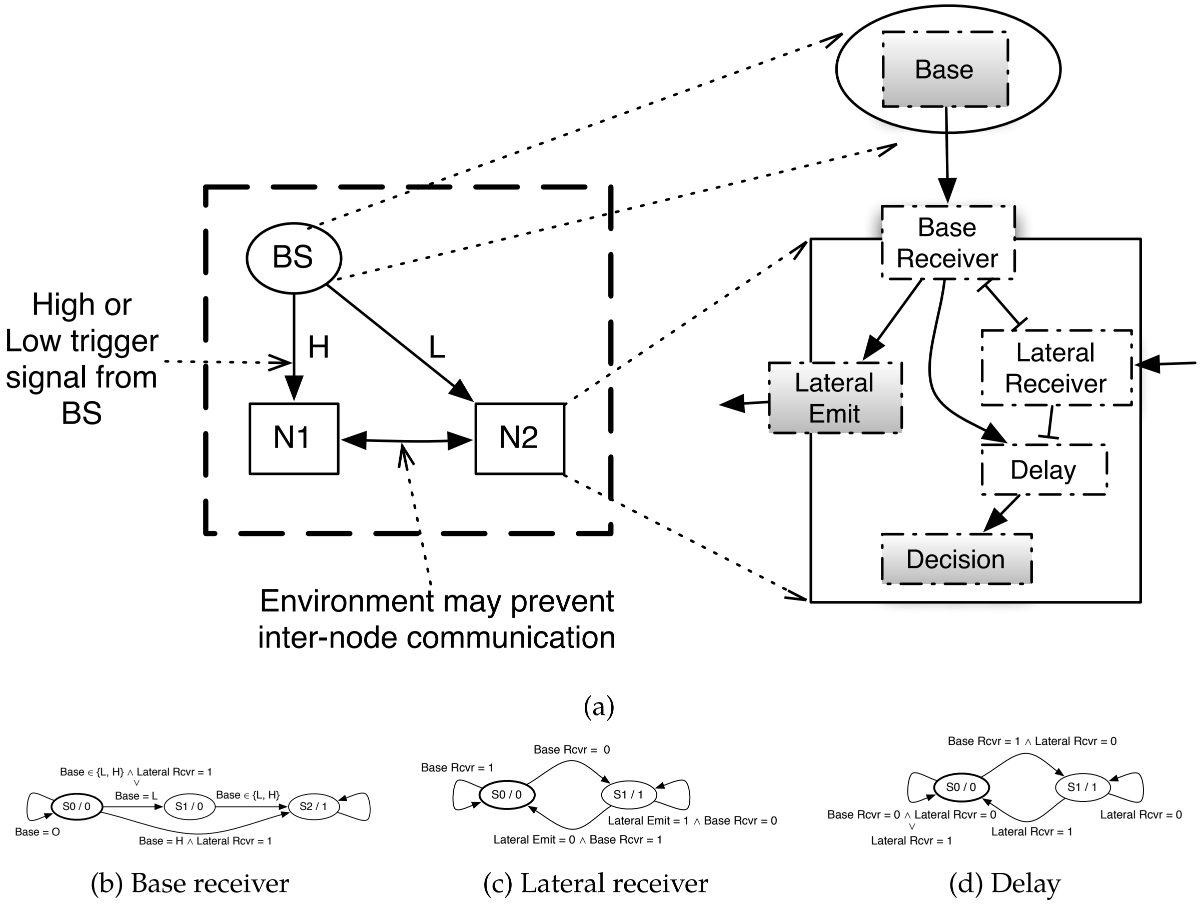


Figure 3.2. Hierarchical organization of programs in SBL. The system is composed of cells, which in turn are composed of components. At each time step, components update their discrete state using an update function, in terms of their previous state and incoming signals from other components. Edges between components denote which components can communicate between them. Cells group together components that always move synchronously, and they adhere to a restricted form of concurrency.

The goal is to design a weak consensus protocol for a three-node system. (In a biological system, these nodes would be cells, and node components would be proteins in the cells.) Two nodes (called sensors N1 and N2) are listening to a signal from a master node (a base station BS). When the base station sends a signal, at least one of the sensors must make a decision to take a measurement. When a sensor takes a measurement, it sends a release message to the other sensor permitting the other sensor not to take a measurement in order to save its power. The decision to make a measurement is made on the basis of (1) the strength from the base station; in normal conditions, the sensor that received the stronger signal should take measurement as it is closer to the base station; and (2) receiving the release message from the other signal. The environment may cause the communication between the two sensors to be down, sensors must take a measurement if no signal was received from their peer. Similar to a system of cells progressing at similar rates, we assume that sensors have bounded skew, i.e. they run under bounded asynchronous schedules.

An implementation of this protocol is presented in Figure 3.3. Figure 3.3(a) presents a hierarchical view of how cell communication is organized, and which components each cell contains. On the left is the top-most level with three nodes; the base station node (BS) contains one component, the base node,



(a)

(b) Base receiver

(c) Lateral receiver

(d) Delay

Figure 3.3. (a) Hierarchical view of node connections, and of their components. The top node is the base station, and the bottom nodes are distributed sensors which may not communicate with each other due to environment effects. (b), (c), (d) Graphical representation of update functions for base receiver, lateral receiver and delay components in the distributed sensors. Each state is labeled with its name and the output value that the state maps to.

which emits a constant high (H) or low (L) signal to nodes N1 and N2. These nodes decide to commit or to delegate by communicating with each other. Figure 3.3(b), (c) and (d) show a graphical representation of update functions for three components in nodes N1 and N2 (the remaining simpler update functions have been omitted from the figure).

Language Extensions for Verification

To make programs in our language amenable to verification, we now introduce component mutations, formalize specifications, and define a correctness condition for programs.

We model cell mutation with an adversary who perturbs the cell program such that a set of adversary-selected cell components receive adversary-supplied semantics. Typically, a cell component is mutated either to be suppressed or to stay at a high concentration level throughout the execution of the program, although we also support other mutation types.

The set of mutation experiments performed in the lab serve as our correctness specification. Let F be the set of possible outcomes of a mutation experiment. For example, if a cell can take one of three fates, the outcomes of an experiment with six cells is a six-tuple from $F = \{1, 2, 3\}^6$. Let M be the set of possible mutations that one can apply on a cell; typically, all cells involved in an experiment are mutated identically. The set of experiments Exp is a subset of $M \times F$, where $(m, f) \in Exp$ if the fate f has been observed on the mutation m. With n cell components and three possible mutations per component (e.g., no mutation; suppressed; high level), M is exponential in the number of components of the cell. As a result, biologists do not carry out all mutations.

Having an incomplete set of experiments implies that we have to accommodate partial specifications. While the set of experiments Exp is a subset of $M \times F$, we assume that once a mutation has been carried out, the lab has observed all possible outcomes for this mutation by repeating the experiment a sufficient number of times. This is a reasonable assumption for systems that have been reliably studied by many independent labs, such as our case study, vulval fate specification in *C. elegans* (Section 3.6). Without this assumption, we would have no upper bound on the specification, as any (m, f) pair could potentially be observed in experiments that have not been performed so far. The assumption allows us to synthesize with both positive examples (outcomes that must be produced by the model for an experiment) and negative ones (outcomes that must never be observed for an experiment). To model such full knowledge for a single mutation, our specification is a (partial) map $E : M \to 2^F$. The domain of E is the set of performed mutations. If $m \in dom(E) \land f \notin E(m)$, we assume that mutation m cannot result in fate f; the pair (m, f) is a negative example. We say that a program $P : M \to F$ is a *correct model* of E if, for each $m \in dom(E)$, the execution $P(m)$ may produce each element of $E(m)$ by controlling some aspect of the execution of P, namely the schedule that controls the concurrent execution of cells in the program.

Correctness Condition. To define a correctness condition, we view an SBL program as a function $P : (M, S) \to F$, where M and F are domains of mutations (input configurations) and fates, while S is the set of schedules adhering to bounded asynchrony. The explicit schedule allows us to formulate a correctness condition *correct*(P, E) of a program P on a specification $E : M \to 2^F$, which has two parts:

Base station trigger	Inter-node comm.	N1	N2
N1=H, N2=H	Y	C	D
		D	C
		C	C
N1=L, N2=L	Y	C	D
		D	C
		C	C
N1=H, N2=L	Y	C	D
N1=L, N2=H	Y	D	C
N1=H, N2=H	N	C	C
N1=L, N2=L	N	C	C
N1=H, N2=L	N	C	C
N1=L, N2=H	N	C	C

Figure 3.4. The specification for the distributed protocol example, giving required outcomes for nodes N1 and N2 under a range of scenarios of base station trigger signals and cases of whether the two nodes can communicate between themselves (Y) or not (N). C = Commit, D = Delegate.

1. *demonic scheduling:* A demonic scheduler cannot make the model produce a fate that is outside the specification, i.e. $demonic(P) = \forall m \in dom(E).\forall s \in S : P(m,s) \in E(m)$.

2. *angelic scheduling:* An angelic scheduler must be able to produce each fate in the specification, i.e. $angelic(P) = \forall m \in dom(E).\forall f \in E(m).\exists s \in S : P(m,s) = f$.

The demonic requirement asks that the model is an underapproximation of the specification, while the angelic requirement asks that it is an overapproximation. Angelic scheduling adds a layer of difficulty that is handled through the construction of a novel verifier (Section 3.5).

Example 3.2 *The specification for Example 3.1, expressed as a set of experiments, is shown in Figure 3.4. The left column shows the mutations (environment effects) M, while the right column shows the desired outcomes F. It is interesting to note that we are using the mutations as the environment adversary; the mutations describe situations under which the nodes N1 and N2 must operate according to the expected outcomes. For example, the last row describes the situation in which the signal arriving at N1 is high, while the signal arriving at N2 is low, and the communication between nodes is down. We can think of this mutation as the adversary lowering the signal to N2 and preventing the communication between the two sensor nodes. The outcome C means that a node has committed to taking a measurement while D means that the measurement was delegated to the peer node.*

Language Extensions for Synthesis

In order to allow synthesis of update functions in our programs, we extend our language such that these can be left unspecified. We describe partial programs in SBL and we define the synthesis problem.

The input to the synthesizer is the specification E and a *partial program* $P^?$ to be completed by the synthesizer, if feasible, into a program P^h such that the predicate $correct(P^h, E)$ holds. A partial program is a program template in which certain fragments are parameterized and need to be supplied by the synthesizer. Our language allows parameterization of (1) cell component behavior; and (2) how components communicate. Because update functions model timing delay and change rates of proteins, we found them to be the hardest part of the model to produce manually. By parameterizing update functions, we can indirectly leave unspecified also the connections between components: for example, if a biologist is unsure whether a protein P is inhibited by a protein Q or a protein R, both Q and R can be connected to P; if Q turns out not to influence P, the synthesizer is able to produce an update function for P that disregards the state of Q. The parameterized update functions are constrained to agree with the activation and inhibition semantics specified in the partial program by restricting their structure. This is achieved by stating monotonicity invariants on how a protein's input concentrations can influence its concentration; these invariants are described in Section 3.4.

From the user standpoint, the partial program $P^?$ encodes biological assumptions; it defines the components in the cells as well as a superset of connections between them. It thus (1) conveys the desire to model particular proteins and (2) states the knowledge of which (superset of) pairs of proteins communicate. Partial programs encoding biological assumptions form the basis for the ambiguity analysis described in Section 3.2.

Our synthesis problem is to find update functions h that yield a correct model:

Definition 3.1 (Synthesis problem) *For a partial program $P^?$ to be completed with hole values h into P^h, the synthesis problem is to find the update functions h that yield a correct model:*

$$S(h) := \exists h : demonic(P^h) \wedge angelic(P^h)$$

A correct model must reproduce all observed experiments, and this is captured in the $angelic(P)$ correctness condition, which is a formula with two levels of quantification (2QBF). This makes the synthesis problem a 3QBF problem, while typical synthesis problems are 2QBF (of the form $\exists\ hole\ \forall\ input : \phi$).

Formulas with more than one level of quantification cannot be handed off directly to an SMT solver, because the performance of SMT solvers is only reliable for existential (one quantifier) formulas. One way to tackle 2QBF problems is to develop a counterexample-guided inductive synthesis (CEGIS) algorithm. In the classical CEGIS algorithm, an inductive synthesizer produces a program that is correct on a small sample of inputs; a verifier then checks this candidate program on remaining inputs [140]. To handle the 3QBF synthesis problem $S(h)$, we develop a novel two-part CEGIS algorithm, where an inductive synthesizer communicates with two verifiers, one for each of the two correctness conditions, and collects two kinds of counterexamples, one from each verifier (Section 3.5).

Example 3.3 *The update functions for Example 3.1, presented in Figures 3.3(b), (c) and (d) are produced by our synthesizer. These update functions control how these components react to signals from the base station and the peer sensor. The synthesizer takes four seconds to generate these update functions. Intuitively, a sensor's protocol is simple: if you receive a weak signal, wait a little while and wait for the release signal from the other sensor. If it does not arrive, take a measurement. Still, even for this simple protocol, designing the update functions manually is not trivial.*

Ambiguity Analysis

Assume that a biologist produces an executable model that verifies against all performed experiments. Now imagine that after he publishes his conclusions from this model, another biologist performs a new mutation experiment whose outcome invalidates the model as well as the conclusions drawn from it. (Given a new mutation experiment m_{n+1}, a model P becomes invalid if it cannot reproduce an outcome observed for m_{n+1}, or if it produces an outcome that has never been observed after performing m_{n+1} sufficiently many times.)

Naturally, we are interested in the question of whether one can ascertain the validity of a model in the absence of complete experiments. In particular, under what assumptions can a model be considered the sole explanation of biological phenomena?

We view this question as analysis of ambiguity in the specification E, and define an *alternative model query* that answers the question. We first introduce aggregate outcomes and specification ambiguity.

Definition 3.2 (Aggregate outcome) *Let P be a model and m a mutation. The* aggregate outcome *of P on m, denoted $P[m]$, is the set of outcomes produced by P mutated with m over the set S of all schedules: $P[m] \ := \ \{P(m,s) \parallel \forall s \in S\}$*

A specification E is ambiguous for a partial program $P^?$ (that expresses a set of biological assumptions) if we can find two completions P^{h_1} and P^{h_2} that disagree on some new experiment. Of course, one of these models would become invalid given the new experiment.

Definition 3.3 (Specification ambiguity) *Given a partial program $P^?$, a specification E is ambiguous, denoted $Amb(E, P^?)$, if $\exists m \ \in \ M \ \ \exists h_1, h_2.\ correct(P^{h_1}, E) \wedge correct(P^{h_2}, E) \wedge P^{h_1}[m] \ \neq \ P^{h_2}[m]$.*

Note that m must be a new experiment, i.e. $m \in M \setminus dom(E)$, because the two models must agree with the specification E on all mutations in $dom(E)$.

In Section 3.6, we show that the specification for our case study is unambiguous given provided biological assumptions (i.e., there is no need for more experiments at the desired level of modeling). We also show that removing some historically important experiments indeed makes the specification ambiguous, permitting alternative explanations for coordination between cells.

Definition 3.4 (Alternative model query) *Given a partial program $P^?$ stating biological assumptions and an existing (perhaps previously synthesized) model P (that need not be an instantiation of $P^?$), the* alternative model query *finds a mutation m and a new model P^h such that $P[m] \neq P^h[m]$, or shows that no such h and m exist.*

We develop an algorithm to solve this query in Section 3.5.

Example 3.4 *We now ask whether we can find alternative models for Example 3.1 using the alternative model query. Suppose we relaxed the specification and do not care about the outcome on the case $N1 = L$, $N2 = L$. We ask our synthesizer to generate models under this relaxed specification such that they differ from the model in Example 3.1. Our synthesizer generates an alternative model that has much simpler behavior (as it need not be non-deterministic under the row that we ignored). The update functions are shown in Figure 3.5. When we ask for a mutation that distinguishes among the models, the synthesizer produces the omitted row. (Note that this last query is a special case of the alternative model query, such that both input programs are fully specified.)*

Now consider an experimental scenario where one wants to validate a set of experiments performed in the literature by performing them again. Is it possible to identify the smallest set of experiments whose replication is sufficient to yield a non-ambiguous specification?

To answer this question, we define a *minimization query* that computes such a minimal set.

Definition 3.5 (Minimization query) *Given a non-ambiguous specification E, the* minimization query *computes a minimal non-ambiguous specification E_m from E, i.e. $\neg Amb(E_m, P^?) \wedge \neg\exists E', E' \subset E_m \wedge \neg Amb(E', P^?)$.*

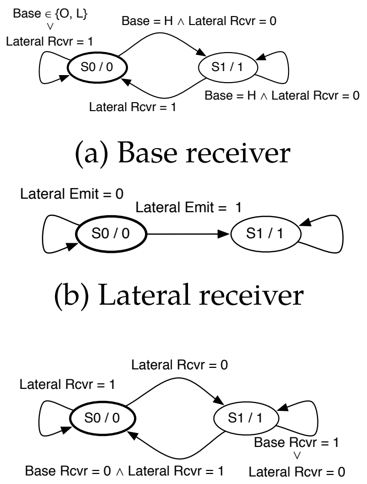
An algorithm that solves the minimization query is presented in Section 3.5.

In our case study, we show that, under our assumptions $P^?$, one needs to replicate about 10% of experiments. This result suggests that computing which experiments to perform might reduce unnecessary laboratory work.

Example 3.5 *We explored the minimization query for Example 3.1. Our synthesizer prunes E down to the first three rows of Figure 3.4 as a minimally unambiguous specification. This is somewhat surprising but it gives substantial insight into the problem, as the user can now understand that the specification of the four cases with lost communication was redundant, while the user may have presumed that it was necessary.*

3.3 Language

In this section, we present the formal semantics of our language, by first defining the language constructs, and then giving operational semantics rules for execution.



(a) Base receiver

(b) Lateral receiver

(c) Delay

Figure 3.5. Update functions generated using the alternative query model to differentiate from the model in Example 3.1 under ambiguous specification, obtained by removing row 2 of Figure 3.4.

The basic construct in SBL is a *component*. We denote the set of all components in a program by *Comp*. Components are connected via a set of directed edges, defined by the relation $Edges \subseteq Comp \times Comp$. Edges model channels of communication between cell components. For each component c, we say a component c' is an *input component* of c if there is an edge $(c', c) \in Edges$. For each c, we define the set of input components $Input_c$ as $\{c' : (c', c) \in Edges\}$. A component c has a state σ_c that takes values from a finite domain L_c. Each component c is also associated with an update function, denoted f_c, that updates its state σ_c, given the current value of its input components. The function f_c has domain $L_c \times \Pi_{c' \in Input_c} L_{c'}$ and range L_c. The update function for a component is chosen from a sequence of functions $F_c := [f_{c,1}, \ldots, f_{c,k}]$ that describe possible alternative behaviors of that component under different mutations, i.e., the natural and altered behaviors of the component.

A *cell* is a set of components. Within a cell, we have a synchronous execution model, i.e. all components of a cell update their state simultaneously. The state of a cell $\bar{\sigma}$ is defined as the set of states of the components that the cell contains. We denote the set of all cells in a program by *Cells*. *Cells* forms a partition on all the components in the program. A pair of cells $(cell_1, cell_2)$ are said to be *communicating* if there exists a pair of components $(comp_1, comp_2)$ connected by an edge in the respective cells.

The pair $(Cells, Edges)$ constitutes a *program*. The program state $\bar{\bar{\sigma}}$ is the set of all cell states in the program. The input to a program is a *configuration* (i.e. a mutation). A configuration is a function from components to integers, that expresses for each component c the index of the function in F_c that should be used as the update function f_c. The output of a program is defined as the state of user-designated components in the final state reached in an execution.

$$\textsc{Run-Program}\ \frac{S = s :: ss \qquad \bar{\bar{\sigma}}_{init} \xrightarrow{s,Cells,Edges} \bar{\bar{\sigma}}' \qquad \langle \bar{\bar{\sigma}}', ss \rangle \xrightarrow{Cells,Edges} \langle \bar{\bar{\sigma}}_{final}, [\,] \rangle}{\langle \bar{\bar{\sigma}}_{init}, S \rangle \xrightarrow{Cells,Edges} \langle \bar{\bar{\sigma}}_{final}, [\,] \rangle}$$

$$\textsc{Run-Program-Base}\ \frac{}{\langle \bar{\bar{\sigma}}, [\,] \rangle \xrightarrow{Cells,Edges} \langle \bar{\bar{\sigma}}, [\,] \rangle}$$

$$\textsc{Advance-Cells}\ \frac{\forall cell \in Cells\ \ \bar{\sigma}_{cell} \xrightarrow{\bar{\bar{\sigma}},cell,Edges,s(cell)} \bar{\sigma}'_{cell} \qquad \bar{\bar{\sigma}}' = \cup_{cell \in Cells}\{\bar{\sigma}'_{cell}\}}{\bar{\bar{\sigma}} \xrightarrow{s,Cells,Edges} \bar{\bar{\sigma}}''}$$

$$\textsc{Cell-Enabled}\ \frac{\forall \sigma_c \in \bar{\sigma}\ \ \sigma_c \xrightarrow{c,\bar{\bar{\sigma}},Edges} \sigma'_c \qquad \bar{\sigma}' = \cup_{c \in cell}\{\sigma'_c\}}{\bar{\sigma} \xrightarrow{\bar{\bar{\sigma}},cell,Edges,1} \bar{\sigma}'} \qquad \textsc{Cell-Disabled}\ \frac{}{\bar{\sigma} \xrightarrow{\bar{\bar{\sigma}},cell,Edges,0} \bar{\sigma}}$$

$$\textsc{Advance-Component}\ \frac{\Sigma = \{\sigma_{c'} : (c', c) \in Edges\} \qquad f_c(\Sigma, \sigma) = \sigma'}{\sigma \xrightarrow{c,\bar{\bar{\sigma}},Edges} \sigma'}$$

Figure 3.6. Small-step semantics for program execution. Run-Program runs a schedule by advancing the cells according to each micro-step in the schedule, with Run-Program-Base as the base case. Advance-Cells rule updates the states of cells, depending on the current micro-step s. If a cell is enabled, it is advanced by applying the Cell-Enabled rule. Conversely, if a cell is disabled, the Cell-Disabled rule keeps its state unchanged. Advance-Node rule updates the state of a component by invoking the update function on the states of all input component states and its own state

Partial Programs. The sequence F_c of functions associated with component c need not be specified concretely. When at least one component function is not concretely specified, we say the program is *partial*. Typically, users will only concretely specify the behaviors under well-understood mutations that would not make sense to redefine. For example, a typical example in the biological case is the knock-out mutation which subdues the function of the component and fixes it to the OFF state.

Operational semantics Figure 3.6 shows the small-step semantic rules for program execution. Here, we assume that the program starts in the initial state $\bar{\bar{\sigma}}_{init}$, and that it has already been preprocessed by fixing a particular update function for each component according to the input configuration. The semantics are defined recursing down the program structure. The Run-Program rule executes the program by moving all cells in accordance with a schedule S. The Advance-Cells rule captures the intuition that each schedule step s partitions the cells into the sets *enabled* (for which $s(cell) = 1$) and *disabled* (for which $s(cell) = 0$). Rules Cell-Enabled and

Cell-Disabled describe how cell states are updated for enabled and disabled cells, respectively. For the disabled cells, the state remains unchanged. Enabled cells are advanced by applying the Advance-Component rule for each component, which corresponds to updating the component state by reading the state of connected neighbors and using the component's update function.

Bounded Asynchrony. The concurrency notion that our execution model admits is bounded asynchrony. This model faithfully represents biological systems where complete synchrony is too strict, and complete asynchrony does not accurately model cells that progress at similar but not identical rates.

Fisher et al. [48] define bounded asynchrony with schedules consisting of micro- and macro-steps. Each micro-step consists of a subset of the components stepping synchronously. This is what we have been calling a schedule up to this point. Next we block micro-steps together into a macro-step. Each k-bounded macro-step consists of all components taking k steps split across multiple micro-steps. For example, let us consider three nodes and the schedule 110 (micro-step) indicates the first two take a step while the third waits. Suppose the second schedule is the micro-step 001. Then the two micro-steps together make a macro-step in which all nodes take one step and which is therefore 1-bounded.

Schedules over micro-steps are much more expensive to enumerate than schedules over macro-steps, especially 1-bounded macro-steps. Schedules over 1-bounded macro-steps (where each node necessarily moves once), can be succinctly encoded without loss of information as pairwise happens-before between connected nodes. That is, a 1-bounded macro-schedule is an assignment of $<$, $>$, or $=$ to each edge in the node topology[2]. The following lemma holds:

Lemma 1 (Fisher et al. [48]) *A micro-schedule exists if and only if a realizable macro-schedule exists over the node topology.*

Here a realizable macro-schedule is one that does not cause an inconsistent ordering of nodes in a cycle. We use this result critically to efficiently encode partial programs as formulas (Section 3.4), and restrict schedules to be 1-bounded.

Using macro-steps allows us to define a compact symbolic encoding of our programs into formulas, which would have not been possible with micro-steps.

3.4 Translating Programs into Formulas

We now describe how to translate execution of SBL programs to SMT formulas, enabling verification and synthesis. We first give rewrite rules that construct a formula corresponding to the symbolic execution of a program. We then describe additional constraints that encode biological domain knowledge to be used in synthesis of programs.

[2] Technically, for micro-steps it is the sequence of ordered bell numbers or Fubini numbers [2], while for 1-bounded macro-steps it is $3^{\text{num edges}}$.

$$\begin{aligned}
\mathrm{T}_{\textsc{Run}}[\![Cells, Edges]\!] &:= \bigwedge_{t\in\{1,\ldots,k\}} \bigwedge_{cell\in Cells} \bigwedge_{c\in cell} \mathrm{T}_{\textsc{Read}}[\![t,c,Edges]\!] \wedge \mathrm{T}_{\textsc{Update}}[\![t,c,Edges]\!] \\
\mathrm{T}_{\textsc{Read}}[\![t,c,Edges]\!] &:= \bigwedge_{\substack{(c',c)\in Edges \\ c'\in cell' \\ c\in cell}} \begin{pmatrix} ((channel_{t,cell',cell}=\text{“}<\text{”})\Rightarrow(\sigma_{read,t,c,c'}=\sigma_{t-1,c'})) \\ \wedge \\ ((channel_{t,cell',cell}=\text{“}=\text{”})\Rightarrow(\sigma_{read,t,c,c'}=\sigma_{t-1,c'})) \\ \wedge \\ ((channel_{t,cell',cell}=\text{“}>\text{”})\Rightarrow(\sigma_{read,t,c,c'}=\sigma_{t,c'})) \end{pmatrix} \\
\mathrm{T}_{\textsc{Update}}[\![t,c,Edges]\!] &:= \bigwedge_{f_i\in F_c} m_c = i \Rightarrow \left(\bigwedge_{(v_c,v_{c_1},\ldots,v_{c_n})\in dom(f_i)} \begin{matrix} (\sigma_{t-1,c},\sigma_{read,t,c_1,c},\ldots,\sigma_{read,t,c_n,c})=(v_c,v_{c_1},\ldots,v_{c_n}) \\ \Rightarrow \\ \sigma_{t,c}=table_{v_c,v_{c_1},\ldots,v_{c_n}} \end{matrix} \right)
\end{aligned}$$

Figure 3.7. Translation rules for symbolic execution of programs.

Translation of Program Execution

The translation of program execution is parameterized by the following symbolic variables:

- For each time step t and each pair of connected cells (c_1, c_2), we define a channel configuration variable $channel_{t,c_1,c_2}$ that must hold exactly one of the three values "<", ">" and "=". These variables encode the symbolic schedule for program execution. Variables $channel_{t,c_1,c_2}$ and $channel_{t,c2,c1}$ are asserted to be consistent in the following way:

$$\begin{gathered}
channel_{t,c1,c2} = \text{“}>\text{”} \Leftrightarrow channel_{t,c2,c1} = \text{“}<\text{”} \\
\wedge \\
channel_{t,c1,c2} = \text{“}<\text{”} \Leftrightarrow channel_{t,c2,c1} = \text{“}>\text{”} \\
\wedge \\
channel_{t,c1,c2} = \text{“}=\text{”} \Leftrightarrow channel_{t,c2,c1} = \text{“}=\text{”}
\end{gathered}$$

- For each component c, we represent each function $f_i \in F_c$ as a lookup table with symbolic values for each value in its domain $L_c \times \Pi_{c'\in Input_c} L_{c'}$. Entries of the lookup table are represented by the variables $table_{v_c,v_{c_1},\ldots,v_{c_n}}$ that take values in L_c.
- For each component c, we represent its mutation symbolically as a variable m_c, that encodes the index of the function to use among F_c. If m_c has value i, then the function $f_{c,i}$ will be used as the update function of component c.
- For each component c at each execution step, we create a variable $\sigma_{t,c}$ that takes values in the domain of L_c. These variables represent the component state symbolically over the execution of the program.

Translation rules for compiling program execution to an SMT formula are shown in Figure 3.7.

T_{Run} is the top-level rule for translating the execution of a program, unrolling the execution for k steps. T_{Read} uses the symbolic macro-schedule values to assert the input states that should be read by each component at a given macro-step. If a cell $cell$ runs before or at the same time as another cell $cell'$ (i.e. the macro-step variable between the two cells at a time step has value "$<$", or "$=$"), the components in $cell$ reads their input states from $cell'$ at the previous time step. On the other hand, if $cell$ runs after $cell'$, it reads its input states from the current time step. Finally, T_{Update} asserts that the state of a component is updated in terms of its symbolic mutation, input state, own state, and update function. The outermost conjunction enumerates over possible update functions. For each mutation, the inner conjunction enumerates possible input value tuples of the update function. The symbolic state is updated given symbolic lookup variables for the chosen mutation.

This translation does not impose any constraints on the parameterized update functions, and therefore encodes a very large space of possible update functions. To help with the program synthesis task, we need to restrict this space. This is achieved in Section 3.4 by asserting biologically motivated constraints on the structure of the parameterized update functions.

Domain-Specific Constraints on Update Functions

The translation in Section 3.4 does not impose restrictions on the structure of the update functions that are left unspecified by the user. When modeling biological systems, formulating a hypothesis typically involves stating high-level invariants about whether a component *activates or* inhibits another one. In this section, we describe how the space of update functions is restricted using this high-level knowledge.

We first formalize how the high-level biological invariants are stated by defining a partial labeling of edges with activation and inhibition semantics.

Definition 3.6 (Edge labeling) *Given a partial program $P^?$, the partial function label : $Edges \rightarrow \{activating, inhibiting\}$ annotates edges in $P^?$ as either activating or inhibiting.*

As a component's state expresses its activation level, we assume the existence of a total order on its possible states. This will allow us to state the properties that restrict the space of update functions.

Definition 3.7 (State ordering) *Let c be a component, and L_c the set of possible state values for c. The state ordering $\leq_c$ is a total order on L_c.*

Using the edge labeling and the state ordering for each component, we now define a partial order on the combined input values for a component.

Definition 3.8 (Input ordering) *Given a component c with update function $f_c : L_c \times L_{c_1} \times \ldots \times L_{c_n} \rightarrow L_c$, the partial order $\preceq_c$ on elements of $L_{c_1} \times \ldots \times L_{c_n}$ is defined as:*

$$\begin{aligned}
&(v_1, \ldots, v_n) \preceq_c (u_1, \ldots, u_n) \\
&\quad := \; \forall i \in \{1, \ldots, n\}. \\
&\qquad (label((c_i, c)) = activating \wedge v_i \leq_{c_i} u_i) \vee \\
&\qquad (label((c_i, c)) = inhibiting \wedge v_i \geq_{c_i} u_i)
\end{aligned}$$

Intuitively, $\preceq$ is a partial order on the strength of the input values to a component, based on the activation and inhibition annotations. We now describe two kinds of invariants that restrict the space of possible update functions.

Input monotonicity Our first property is motivated by the following observation: If there is an activating edge from component c_1 to component c_2, then an increase in σ_{c_1} should not have by itself the effect of decreasing σ_{c_2}. Conversely, if c_1 and c_2 are connected through an inhibiting edge, then a decrease in the value of σ_{c_1} should not result by itself in the decrease of σ_{c_2}:

$$\begin{aligned}
&\forall i_1, i_2 \in L_{c_1} \times \ldots \times L_{c_n}. \forall v \in L_c. \\
&\qquad i_1 \preceq_c i_2 \Rightarrow f_c(v, i_1) \leq_c f_c(v, i_2)
\end{aligned}$$

State monotonicity The second property that we assert imposes a monotonicity constraint on f_c in terms of the value of σ_c. This property expresses that, for the same input value, a greater the activation level of the component cannot be updated to a smaller value:

$$\begin{aligned}
&\forall i \in L_{c_1} \times \ldots \times L_{c_n}. \forall v_1, v_2 \in L_c. \\
&\qquad v_1 \leq_c v_2 \Rightarrow f_c(v_1, i) \leq_c f_c(v_2, i)
\end{aligned}$$

We found that asserting constraints that encode these two invariants based on user annotations on component connections is crucial for ensuring that the structure of update functions agree with existing biological knowledge.

3.5 Synthesis and Querying Spaces of Models

In section 3.4, we described how we translate program execution to formulas. In this section, we present algorithms that leverage this translation for verification and synthesis, as well as specification ambiguity analysis.

The formula that encodes program execution parameterizes (1) update functions (which are the holes in partial programs); (2) schedules; and (3) input configurations (i.e. mutations). The space for update functions and the space for schedules are typically very large. However, specifications are typically wet-lab experiments which are sparse and inherently small (of order

10^2 experiments). Based on this observation, we develop algorithms that unroll quantifications for input configurations only.

In the following, we refer to the symbolic output parameter of translating the execution of P with input m and schedule s as $P(m,s)$, and we denote by E the specification (given as a partial function from M to 2^F).

Verifying Programs

The correctness condition presented in Section 3.2 is defined as:

$$\mathit{correct}(P) := \mathit{demonic}(P) \wedge \mathit{angelic}(P)$$

The properties $\mathit{demonic}(P)$ and $\mathit{angelic}(P)$ are in 1QBF and 2QBF respectively. As a result, the correctness condition $\mathit{correct}(P)$ is in 2QBF.

We verify correctness conditions $\mathit{demonic}(P)$ and $\mathit{angelic}(P)$ separately, using a verifier V_d that searches for demonic schedules that lead to the violation of the specification, and a verifier V_a that checks whether all non-deterministic outcomes for a given mutation can be reached for some angelic schedule.

Verifying for demonic schedules. The formula $\mathit{demonic}(P)$ states that the set $E(m)$ is an upper bound for all observed outcomes of P with input m:

$$\mathit{demonic}(P) := \forall m \in \mathit{dom}(E).\ \forall s \in S.\ P(m,s) \in E(m)$$

To check this property, we attempt to disprove it by searching for a demonic schedule that produces an unobserved outcome for an input in $\mathit{dom}(E)$, the domain of E. Given the observation that there is a small set of input values in $\mathit{dom}(E)$, we solve this formula by unrolling the existential quantification over this set, and by querying symbolically for a demonic schedule. The condition $P(m,s) \notin E(m)$ is expressed by unrolling over values in $E(m)$, which is also a small set. We thus solve the 1QBF formula:

$$\bigvee_{m \in \mathit{dom}(E)} \exists s. \bigwedge_{f \in E(m)} P(m,s) \neq f$$

If this formula is satisfiable, P does not satisfy *demonic*, and we obtain a concrete counterexample (m,s) such that running P on input m and schedule s leads to an unobserved fate. If it is unsatisfiable, then P is correct with respect to *demonic*.

Verifying for angelic schedules. The *angelic* condition states that all outcomes in the set that m maps to must be observable, i.e. appear in some execution of P on m:

$$\mathit{angelic}(P) := \forall m \in \mathit{dom}(E),\ f \in E(m).\ \exists s.\ P(m,s) = f$$

This amounts to searching for an angelic schedule for each $f \in E(m)$. We reduce the 2QBF correctness property to an efficiently solvable 1QBF problem by unrolling values of the domain $dom(E)$, again based on the assumption that this is a small domain. To unroll $angelic(P)$, we construct the following query for each $m \in dom(E)$ and for each $f \in E(m)$:

$$\exists s.\ P(m,s) = f$$

If the above formula is unsatisfiable for some m and f, then no angelic schedule can be found for reaching that outcome when running P, and (m, f) is a counterexample input/output pair witnessing that $angelic(P)$ does not hold. If the formula is satisfiable for each $m \in dom(E)$ and for each $f \in E(m)$, then verification for angelic schedules succeeds.

A program verifies against the specification E if it verifies against both V_d and V_a.

Synthesizing Programs

In our language, it is possible to define a partial program $P^?$ that admits freedom in the update functions of its components. We now present a synthesis algorithm for finding update functions in $P^?$ such that the completed program P^h is correct with respect to the correctness condition $correct(P)$. Our procedure leverages the two verifiers V_d and V_a to check correctness properties *demonic* and *angelic* respectively, in order to solve the following synthesis problem:

$$S(h) := \exists h.\ demonic(P^h) \wedge angelic(P^h)$$

This formula is in 3QBF, due to the quantifier alternation $\exists\forall\exists$ resulting from $angelic(P^h)$ being nested within the quantification over h.

We solve $S(h)$ by developing a counterexample-guided inductive synthesis (CEGIS) algorithm, which decomposes the 3QBF problem into two 1QBF solvers (an inductive synthesizer and the demonic verifier V_d) and one 2QBF solver (the angelic verifier V_a). The inductive synthesizer produces a candidate model that is correct on all counterexamples and sends this model to both verifiers. If both approve the model, the synthesis successfully terminates. If either fails, counterexamples are produced, refining the correctness constraints placed on the inductive synthesizer, making it eventually produce a correct model (or conclude that no model exists in the model space described by $P^?$). The solver architecture is shown in Figure 3.8.

Precisely, the synthesizer maintains two sets of counterexamples, $CE_1 \subseteq dom(E) \times S$ and $CE_2 \subseteq dom(E) \times F$. The first set contains pairs of inputs and schedules, and is computed with counterexamples given by the verifier for demonic schedules. The second one is a subset of the input/output specifications, and is in turn computed with counterexamples found by the verifier for angelic schedules. Starting with initial sets CE_1 and CE_2, the synthesizer solves at

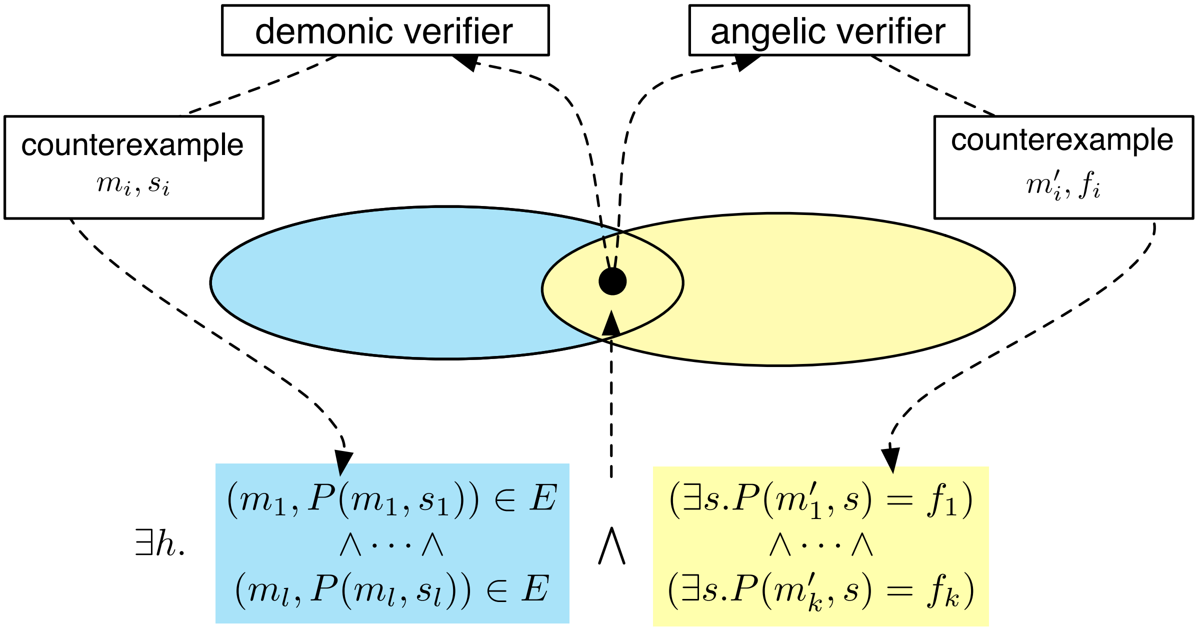


Figure 3.8. The synthesizer consists of three communicating solvers. The two verifiers generate two kinds of counterexamples, and the synthesizer generates models that satisfy the constraints for all counterexamples.

each step the following formula to find a candidate model:

$$\begin{aligned} \exists h. \quad & \left(\bigwedge_{(m,s)\in CE_1} P^h(m,s) \in E(m) \right) \\ \wedge \quad & \left(\bigwedge_{(m,f)\in CE_2} \exists s.\ P^h(m,s) = f \right) \end{aligned}$$

If the above formula is unsatisfiable, the partial program cannot be completed, i.e. synthesis fails. Otherwise, the valuation of h defines a candidate model that we attempt to verify using verifiers V_d and V_a. If at least one of the verifiers returns with a counterexample, the synthesizer attempts to find a new candidate after updating the sets CE_1 and CE_2 with the counterexamples returned by either verifier. If a candidate model is validated by both verifiers, we obtain the completed program P^h that is correct with respect to the specification E.

Querying for Ambiguity Analysis

Given the above procedure for synthesizing programs, we are now interested in querying spaces of possible models. In particular, we analyze ambiguity of specifications. If a specification is underspecified, we aim to reduce ambiguity by expanding it. If, on the other hand, it is overspecified, our goal is to reduce the specification size without introducing ambiguity.

Computing Aggregate Outcome We first give an iterative algorithm to find aggregate outcome set for a given program p and a given input m. The aggregate outcome set $P[m]$ is the set of outcomes of P on m over all schedules. We approach the task by first computing the outcome of P on m under an initial schedule s. We then enlarge the set of observed outcomes Obs by searching for a schedule leading the program to produce a previously unseen outcome. To find such an outcome, we build a formula that states that the new outcome must differ from each value in the Obs set inferred so far. Each step of the algorithm thus attempts to extend Obs by solving the following formula:

$$\exists s. \bigwedge_{f \in Obs} P(m, s) \neq f$$

If this formula is satisfiable, we obtain an outcome that we add to the set Obs, and then attempt to solve the formula with the updated set. If it is unsatisfiable, we have obtained all outcomes that can be produced by P on input m.

Alternative Models

To ascertain that a given hypothesis is the sole explanation to a biological phenomenon, a biologist would like to learn whether there exists another hypothesis that differs from the first on its observable outcome on an unperformed experiment, but is correct on the known experiments. Given a program P_1 that expresses the first hypothesis, and a partial program $P_2^?$ that expresses a space of alternatives for the second, we can state this query formally as:

$$\exists m. \exists h. correct(P_2^h) \wedge P_2^h[m] \neq P_1[m]$$

If this query is satisfiable, then there is an alternative program P_2^h and a new experiment m such that performing the experiment m will invalidate at least one of P_1 and P_2^h. We now describe an algorithm to solve this query.

Given the hypothesis that the space of mutation experiments M is small, we approach this task by unrolling the existential quantification over m. The problem then reduces to synthesizing P_2^h for a given mutation m, such that $P_2^h[m] \neq P_1[m]$.

$P_2^h[m]$ can differ from $P_1[m]$ in two distinct ways: (1) It can either contain an output value not in $P_1[m]$; or (2) it can be a strict subset of $P_1[m]$. We give one algorithm for each case.

Case 1. A program P_2^h that produces an outcome not seen in $P_1[m]$ can be found by augmenting the synthesis query described in Section 3.5 with a constraint asserting that there exists a schedule that leads P_2^h to produce an outcome not in $P_1[m]$, i.e. $P_2^h[m] \setminus P_1[m] \neq \varnothing$. We solve the following formula to answer this query:

$$\exists h.\ correct(P_2^h) \wedge \exists s. P_2^h(m, s) \notin P[m]$$

This formula is satisfiable if and only if there exists a completion of program P_2^h that produces an outcome not in $P[m]$. It is 3QBF, and is handled using the mechanism of a synthesizer communicating with two verifiers to perform inductive synthesis described in Section 3.5.

Case 2. Alternatively, P_2^h may be found by attempting to synthesize a model that always produces outcomes in a strict subset of $P_1[m]$. This is achieved by discarding elements of $P_1[m]$ one at a time, to see if such a model can be found. We do not need consider all subsets of $P_1[m]$, as we only state that $P_1[m] \setminus \{f\}$ is only an upper bound of the possible outcomes for input m.

$$\begin{aligned} \exists h. \qquad & correct(P_2^h) \wedge \\ & \Big(\bigvee_{f \in P_1[m]} \forall s.\ P_2^h(i,s) \in P_1[m] \setminus \{f\} \Big) \end{aligned}$$

This formula is satisfiable if and only if there exists a completion of program P_2^h such that its observable outcome set is a strict subset of observable outcomes of P_1 on input m. Similarly to Case 1, we use the scheme of cooperating solvers described in Section 3.5 to solve this formula.

Minimization

In a context where performing experiments is an expensive process, a researcher may want to obtain a minimal non-ambiguous specification that sufficiently constrains the space of models to validate a hypothesis. Given a partial program $P^?$ that expresses a hypothesis, and a specification E that is non-ambiguous with respect to $P^?$, the task of finding a minimal non-ambiguous specification E_m is stated as:

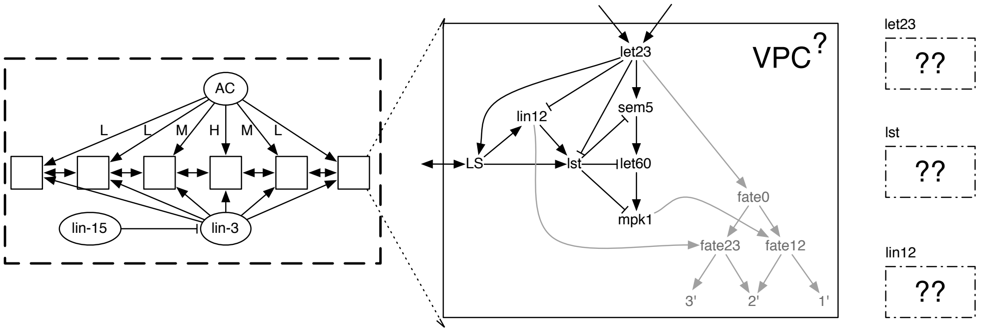
$$\neg Amb(E_m, P^?) \wedge \neg \exists E', E' \subset E_m \wedge \neg Amb(E', P^?)$$

We compute a minimal specification E_m by iteratively restricting the domain of E for the partial program $P^?$. This can be done by invoking the *alternative model* query once for each mutation in $dom(E)$.

At each step, we check whether program $P^?$ can be completed to a program P^h that decides a set of outputs $P^h[m]$ distinct from $E(m)$, considering as specification the set of currently non-redundant input values. This check is performed using the *alternative model query* described in Section 3.5. If synthesis fails, m is marked as redundant. Otherwise, removing m from the specification leads to ambiguity, and as a result m should be kept in the final set of pruned inputs. Upon considering all inputs in the domain of E, a minimal specification is obtained by removing from the domain of E those inputs that are marked as redundant.

3.6 Case Study: *C. elegans* vulval development

We attempt to synthesize a model for the vulval precursor cells (VPCs) that start off identical but through coordination among themselves and with the Anchor Cell (AC) agree on specific fates.



Exp#	Mutations					Fate pattern					
	AC	lin12	lin15	Vul	lst	P3.p	P4.p	P5.p	P6.p	P7.p	P8.p
1	Formed	wt	wt	wt	wt	3	3	2	1	2	3
5	Formed	wt	ko	wt	wt	1/2	1/2	2	1	2	1/2
7	Formed	wt	ko	ko	wt	3	3	3	3	3	3
13	Formed	ko	ko	wt	wt	1	1	1	1	1	1

Figure 3.9. (a) The template $VPC^?$ we use for our experiments, which is derived as simply the union of connections known to biologists [50] as informally shown by Figure 3.1. The "fate" nodes are instrumentation nodes to help read out the outcome. (b) A small fraction of the specification E (4 rows out of 48), obtained from literature in biology [50]. A fate pattern of $1/2$ indicates that both 1 and 2 are outcomes observed in experiments.

From informal descriptions of protein interactions found in biological literature, we develop our template $VPC^?$. The template is shown in Figure 3.9(a) (derived from Figure 3.1.)

From the template, we observed that there are nodes with extremely simplistic on-off behavior. These are LS, the downstream nodes of the cascade (sem5, let60, and mpk1) and the fate nodes. While we can introduce holes in them (with expected performance degradation, yet not being intractable), biologists have a very clear understanding of these nodes, and so expect to see a simple and known behavior in them. Additionally, introducing holes in these nodes leaves too much freedom to the synthesizer, such that generated models do not have a biological interpretation.

Therefore we run our tool with unknown update functions for lin12, let23, and lst. The generated update functions satisfy the specification and template structure of the program. On the other hand, lin12, which has a very well understood behavior, colludes with the other components to give models that are hard to explain to the biologists. Therefore, we additionally allow the user to specify the behavior of lin12 concretely and synthesize let23 and lst. Let23 and lst are indeed the most complex functions in their timing delays (and have the most complex interconnection dependences). Indeed, in our attempts prior to synthesis (when designing the verifier) to write the model by hand, we actually failed. Additionally, the models previously written did not maintain the requisite lin12 behavior. Therefore, our synthesizer was solving a problem that had been impossible to solve manually, even after considerable effort.

The specification consists of forty-eight experimental observations of the fate outcomes of six VPC cells in sequence. Some of these observation have non-deterministic outcome fates. A fragment of the specification is shown in Figure 3.9(b).

From the template and these experiments, our synthesizer generated update function solutions to let23 and lst that were confirmed by the biologists to be plausible behaviors. The output from the synthesizer is shown in Figure 3.10(a).

It is important to note that this is a very significant achievement. Previously, when we had written down a model for VPCs in RM [50] it had the following flaws: (1) The previous model did not satisfy a biological invariant required on the lin12 component, and all efforts to fix the model failed, (2) RM is too expressive and therefore there were cases where the model "read the future" which was hard to interpret biologically, (3) the model lacked readability prohibiting debugging, extension, and biological interpretation. Our synthesized alternative model solves all these. Our first biologically relevant result is therefore that through synthesis we have revalidated the (experimentally-confirmed) prediction from previous work, *without the vagaries of human modeling*.

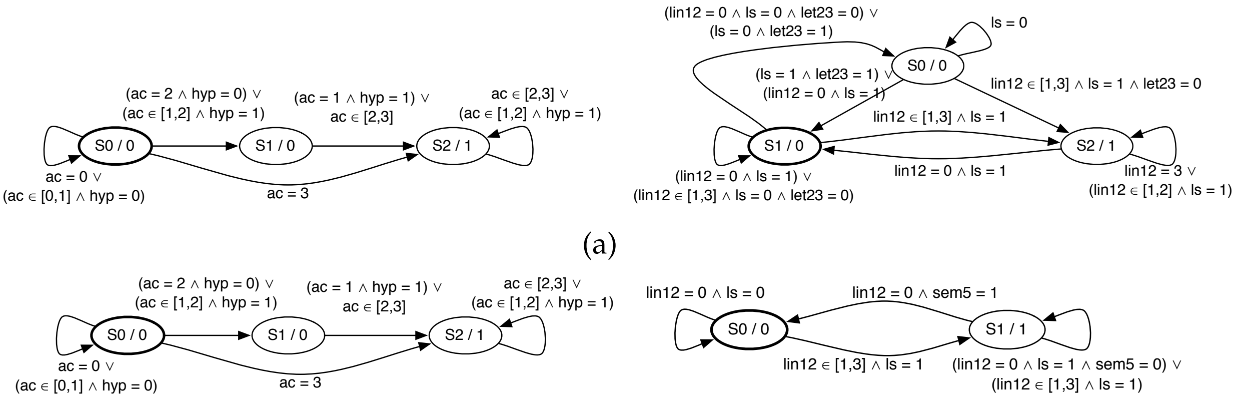
Specification ambiguity for *C. elegans* VPC models

Next, we analyzed the ambiguity in the specification. The important biological unknown is the specific node within the cascade let23-sem5-let60-mpk1 that sends out the inhibitory signal to lin12 and lst. We attempted experimenting with all four options under our definition of understanding the specification ambiguity:

Alternative models for particular input configuration 44 of the 48 experimental observations are deterministic. We wanted to know how many models exist if only the deterministic outcomes are asserted. We found that under this relaxed specification, *all* four options of inhibition coming from any node of the cascade work.

Then using the alternative model query from Section 3.5, we asked for a model including any one of the four remaining outcomes. The synthesizer eliminates two that have inhibition emanating from let60 and mpk1. This was significant since it formally confirmed the biologist's intuition that the inhibition comes from higher up in the cascade. Additionally, it showed that sem5 (in addition to let23, which was conjectured earlier) was a valid possibility for the inhibitor.

Input configuration for disambiguating models Next, we attempted to observationally distinguish these two remaining valid models. Our 48 observations mutate the entire cascade (all nodes let23 to mpk1) together. We wanted to infer if a finer-grained mutation exists that distinguishes these two remaining mechanistic hypotheses. We expanded the experimental set by enumerating all possibilities of the cascade nodes (2^4 possibilities of expansion for each of the 48 rows) leading to 384 experiments. Our synthesizer shows that *no other mutations exist* that would observationally distinguish these two hypotheses. This saves the biologist significant



(a)

(b)

Figure 3.10. Synthesized update functions given two different connection topologies, for let23, and lst. (a) The topology with lst and lin12 inhibited by let23. The template allows for three let23 states and three lst states. (b) The topology with lst and lin12 inhibited by sem5. The template allows for three let23 states and two lst states.

effort (336 experiments, each of which are expensive and time-consuming) as they now know that mutation experiments will not suffice to distinguish these explanations and out-of-band experiments need to be performed.

Inferring the minimal specification We run our minimization query from Section 3.5 for each of the VPC queries, with significant results. We infer that, for the space we are searching over, only four experimental observations suffice to yield a unique model. This set contains all non-deterministic outcomes, and additionally others that together constrain the system enough to yield the unique model that is explained by the 48 experiments.

Wet-lab predictions Our exploration demonstrated that (1) let23 is not the only possibility for inhibition, but sem5 is as well; (2) let60 and mpk1 cannot play that role; and (3) the models using let23 and sem5 cannot be distinguished observationally. These suggest a possible inhibition from sem5, that cannot be distinguished through mutation experiments on the components included in our model, therefore other types of experiments would need to be done.

3.7 Performance evaluation

We implemented our language as an embedded DSL in Scala. Our synthesis and analysis framework, also implemented in Scala, uses the Z3 theorem prover [102] as its underlying constraint solver. We interface with Z3 through the Scala^{Z3} library [82]. Our framework consists of 5K lines of code.

example	time	mem.	# calls	$\frac{\text{time}}{\text{\# calls}}$	holes	search space
VPC1	96.64	2.34	282	0.09	$(3,3)$	$2.25 \cdot 10^{34}$
VPC2	87.77	2.33	285	0.08	$(3,2)$	$1.21 \cdot 10^{21}$
VPC3	48.29	0.77	139	0.10	$(4,3)$	$1.47 \cdot 10^{42}$
VPC4	49.18	1.26	133	0.09	$(5,3)$	$7.25 \cdot 10^{50}$
Sensors	4.30	2.40	51	0.01	$(3,2,2)$	$2.53 \cdot 10^{13}$

(a)

example	time	mem.	# calls	$\frac{\text{time}}{\text{\# calls}}$	$\frac{\text{pruned}}{\textit{total}}$
VPC1	2964.82	2.20	3805	0.54	$4/48$
VPC2	1845.94	1.69	3544	0.31	$3/48$
VPC3	273.77	1.31	491	0.29	$4/44$
VPC4	316.32	1.35	482	0.37	$4/44$
Sensors	14.46	0.71	167	0.04	$3/8$

(b)

Figure 3.11. All times are in seconds, and memory usage is in gigabytes. (a) Evaluation results for synthesis. The number of levels (i.e. states) for each synthesized update function is shown in the *holes* column. (b) Evaluation results for specification pruning. We report for each example the size of the pruned specification domain and the size of the original specification domain.

We show performance results for the evaluation of our synthesis procedure in Figure 3.11(a). For each example, we present total execution time, maximum memory usage, number of calls to the underlying SMT solver Z3, average call time, the structure of holes in the partial programs, as well as the search space for synthesizing update functions. VPC1, VPC2, VPC3 and VPC4 are models of the fate decision in *C. elegans* vulval precursor cell development that express each different biological hypotheses about the cells through their topology. VPC1 and VPC2 are synthesized using a specification E with domain size 48, while VPC3 and VPC4 are synthesized using a specification E' whose domain is restricted to 44 elements. Sensors is the example introduced in Section 3.2. For each example, we report the total running time for synthesis, the maximum memory usage, number of calls to the underlying SMT solver Z3, the average time Z3 takes to solve these queries, a description of holes in the partial program as a sequence of number of states for each unspecified update function, and the size of the search space for synthesizing these functions.

In all cases, we find that even for a complicated synthesis problem such as the VPCs, our synthesizer is efficient.

In Figure 3.11(b), we present performance results for the pruning procedure described in Section 3.5. We report the domain size for the result of the procedure and the initial domain size in the *pruned/total* column.

As expected, the time for pruning is significantly higher than for only synthesis. This is because multiple synthesis and verification queries are solved in the process of minimization.

However, compared to the amount of time this could potentially save the biologists, i.e. months or even years of work in doing redundant experiments, our inference times are insignificant.

3.8 Related Work

Inference of biological models While model checking of (manually written) logical biological models has been an active area of research, we are not aware of work that synthesizes these models. In contrast, a growing body of literature exists on inference of non-logical models. The first class of such models uses ordinary differential equations (ODEs). An example of ODE model inference from temporal and spatial data is the work by Aswani et al., who reduce the amount of prior knowledge needed to infer an accurate model [8]. Rizk et al. find parameters for ODE models by optimizing a notion of continuous degree of satisfaction of temporal logic formulas [123]. Because ODE models are continuous, these techniques do not appear directly applicable for inference of logical models based on concurrent systems.

Machine learning has also been used to infer biological models. Barker et al. use time series data of protein levels to infer whether a protein is an activator or a suppressor of another protein [13]. Time series data of concentrations is not available in our setting, so these approaches do not apply to the inference of our models.

Stochastic modeling An alternative to modeling biological systems using non-deterministic concurrency is to use stochasticity [7, 97, 61]. If we were interested in making predictions on the system's output behavior, i.e. the *most likely* the behavior of the cell for a given mutation, we might select a model that predicts concentrations of proteins under varying initial parameters, including those not yet measured in the lab. Such predictive models are often stochastic.

In contrast, we care to only discover a mechanistic explanation for the cellular system (i.e. how proteins communicate to agree on a particular cell fate). It is appropriate here to rely on a discrete model because the modeling problem is to find a program that reproduces each observed outcome on at least one execution — as opposed to some ratio of all executions. The existence of such a schedule is sufficient to determine the need to have the crucial protein-protein interaction.

Synthesis algorithms for concurrent systems. Our synthesis algorithm extends the synthesis algorithm for concurrent data structures [138]. That work showed how to extend the CEGIS algorithm [139] from the sequential setting into the semantics of concurrent programs. The resulting algorithm however did not handle the richer specification used in this work (i.e. the angelic correctness). Indeed, new algorithms had to be developed for the specifications of this work. The Paraglider project developed synthesizers for concurrent data structures by deriving them from high-level specifications [153]. It is not clear how these derivation algorithms can be adapted to synthesis of concurrent systems under input-output examples such as ours.

Model checking [61, 14, 39, 15, 24] and abstract interpretation [35] have been applied to analyze various biological systems. All such efforts to manually construct and validate models have severely demonstrated the need for a synthesis system.

Various other paradigms have been used to model biological systems, including Petri nets [25], boolean networks [90], and process algebras [120]. While our techniques are not directly applicable, our success in synthesis for a model previously expressed in the expressive RM formalism demonstrates potential for synthesis in these other formalisms as well.

3.9 Conclusion

We present a language and develop algorithms for synthesizing concurrent models from experiments that perform mutations on biological cells and observe the results of the mutation on developed cells. We synthesize models that reproduce all non-deterministic outcomes of experiments. This variant of synthesis requires a 3QBF algorithm, which we design by allowing three solvers to communicate counterexamples. We also develop algorithms for analyzing specification ambiguity, ascertaining that a model is the sole biological explanation whenever possible under given biological assumptions, and computing minimal non-ambiguous specifications. We carried out a significant case study, synthesizing a model of vulval cell fate specification in the *C. elegans* earthworm that expresses a previously unknown biological hypothesis.

Chapter 4

Investigating the Identifiability of Boolean Network Models from Single-Cell Data

Single-cell data offer a high-resolution view of dynamic cellular processes that can be used to uncover the underlying mechanisms from a single “run”. It has been demostrated that symbolic reasoning enables an efficient search that can exhaustively explore the space of Boolean models that agree with such data. However, data imperfections such as measurement errors and sparse sampling make it challenging to infer such executable models. We present Karme, a framework we developed to infer Boolean network models of gene regulation from single-cell data and cell trajectories, and a workflow to evaluate the identifiability of Boolean network models in face of varying degrees of measurement errors, sample sparsity, and prior knowledge. This workflow consists of applying Karme on *in silico* observations of myeloid differentiation, and evaluating its performance in predicting behavior under unseen perturbations. We further propose an experimental design approach to distinguish inferred Boolean network models using user-defined metrics, and demonstrate its ability to identify perturbation experiments that target genes that have a central role in the modeled system.

4.1 Introduction

A fundamental goal in biology is to reveal the mechanisms controlling the expression of genes in dynamic cellular processes, such as the differentiation of a progenitor cell type into multiple cell types. Gene regulatory networks (GRNs) regulate gene expression in the cell, and discovering their structure is useful to understand cellular development. Traditional transcriptomics assays show gene expression levels averaged over a population of cells, hiding heterogeneity between distinct cell types that have different gene expression patterns. With recent technological advances, it is possible to perform single-cell transcriptomics assays, which offer a high-resolution view of heterogeneous cell populations, by measuring gene expression in individual cells. However, measuring an individual cell destroys it, and it is therefore impossible to trace a single cell's evolution through a cellular process.

The first challenge in modeling GRNs from single-cell transcriptomics data is to recover the temporal expressions of genes from snapshots of individual cells. A multitude of methods aiming to identify cell lineage trees and order cells across each lineage "trajectory" have been developed in recent years [10, 128]. Popular trajectory inference approaches include Monocle [150], which builds a minimum spanning tree (MST) on cells after a dimensionality reduction step, and Slingshot [145], which builds a MST on cell clusters.

Once cell trajectories have been identified, it is desirable to produce models of the GRNs that control the observed dynamic processes. Computational modeling allows to give a mechanistic explanation for cellular processes observed through single-cell transcriptomics data. One way to model GRNs is to use continuous representations of entities, using differential equations or probabilistic models to explain the interactions within networks. Jang et al. [71] propose a probabilistic model of gene regulatory networks to explain a lineage tree of discrete cell states inferred through a Bayesian framework [53], and explore a sample of models that fit the data. Weinreb et al. [157] use differential equations to model GRNs from single-cell data, and explicitly address ambiguity in the model space through a choice of assumptions.

An alternative to continuous modeling of GRNs is the use of discrete models. Discrete modeling has the benefit of allowing exhaustive exploration spaces of models, and enables model inference methods that use symbolic reasoning to perform this exploration in a highly efficient manner. Woodhouse et al. [159] introduce the idea of viewing discretized single-cell gene expression data as states that a model must visit, and use symbolic reasoning to infer Boolean models from the data. The method makes the assumption that sufficiently many states have been observed in the input data to obtain a state graph that is connected by edges that change only one gene's value at a time, and does not handle the case of missing states in the input. Lim et al. [91] approach the same problem using a local optimization procedure that uses a hill climbing strategy to modify an initial network. Unlike the work by Woodhouse et al. [159], this approach avoids the necessity to observe a connected state graph. However, as it is a local search procedure, it relies on the existence of a good initial network as a starting point for the search.

In this work, we investigate the identifiability of Boolean network models from single-cell data. We assess the robustness of model identifiability in presence of data imperfections, such as measurement errors and sparse data sampling, through an *in silico* empirical analysis that uses a realistic model of myeloid cell differentiation [88]. For this purpose, we use a collection of high-level and low-level model behavior metrics.

As part of our work, we developed Karme, a method to infer Boolean network models from single-cell data and cell trajectories. Karme leverages cell trajectories to construct a state transition graph on cells, and does not require all intermediary states to be observed in the single-cell data. Karme exhaustively explores the space of gene regulatory network models that satisfy a customizable set of optimality criteria and that maximally fit a set of weighted constraints derived from the inferred state transition graphs.

We evaluate Karme's performance using metrics that quantify state graph construction success, and similarity between the behavior and structure of inferred models and the reference model. In our analysis, we make the following observations:

1. We find that model inference is especially robust against false positives in the input data when it is equipped with the knowledge of candidate stable states.

2. We observe that inferred models are able to generalize from input data on a wild-type execution to behavior under unobserved perturbations.

3. We investigate the impact of stable state knowledge on state graph construction and model synthesis by holding out stable state knowledge partially. We find that while state graph construction is not particularly hindered by the lack of data, the structural and behavioral quality of inferred models is affected when prior knowledge about stable states is partially missing.

We further characterize models enumerated by Karme by their distinguishability under previously unseen experimental conditions, and propose an experimental design approach that can suggest the next experiment to perform for user-defined model distinguishability metrics. We perform an experimental design analysis using two model difference metrics, comparing state reachability behavior and stable state behavior of inferred models. We identify Gata2, which is an early hematopoietic factor in myelod differentiation, as the gene whose perturbation leads to the maximum difference among inferred models. Our approach extends existing experimental design approaches through the joint handling of novel model difference metrics and model semantics allowing cyclic functional dependencies [9].

4.2 Overview

In this work, we are interested in the identifiability of Boolean network models of gene regulatory networks (GRNs) from single-cell data. We define identifiability as the ability to infer a semantically unique model given experimental data. Specifically, we build a workflow to generate *in silico* single-cell data from a simulation of a reference Boolean network model, exhaustively explore the space of Boolean models that agree with the generated observations, and compare inferred models to the hidden reference model through several behavioral and structural criteria. An overview of our approach is presented in Figure 4.1.

In Boolean modeling of GRNs, each modeled gene is either in an "off" or an "on" state. Together, the Boolean gene values constitute the state of a cell. Boolean networks capture combinatorial relationships between genes in a regulatory network, by modeling each gene as a Boolean function (Section 1.3).

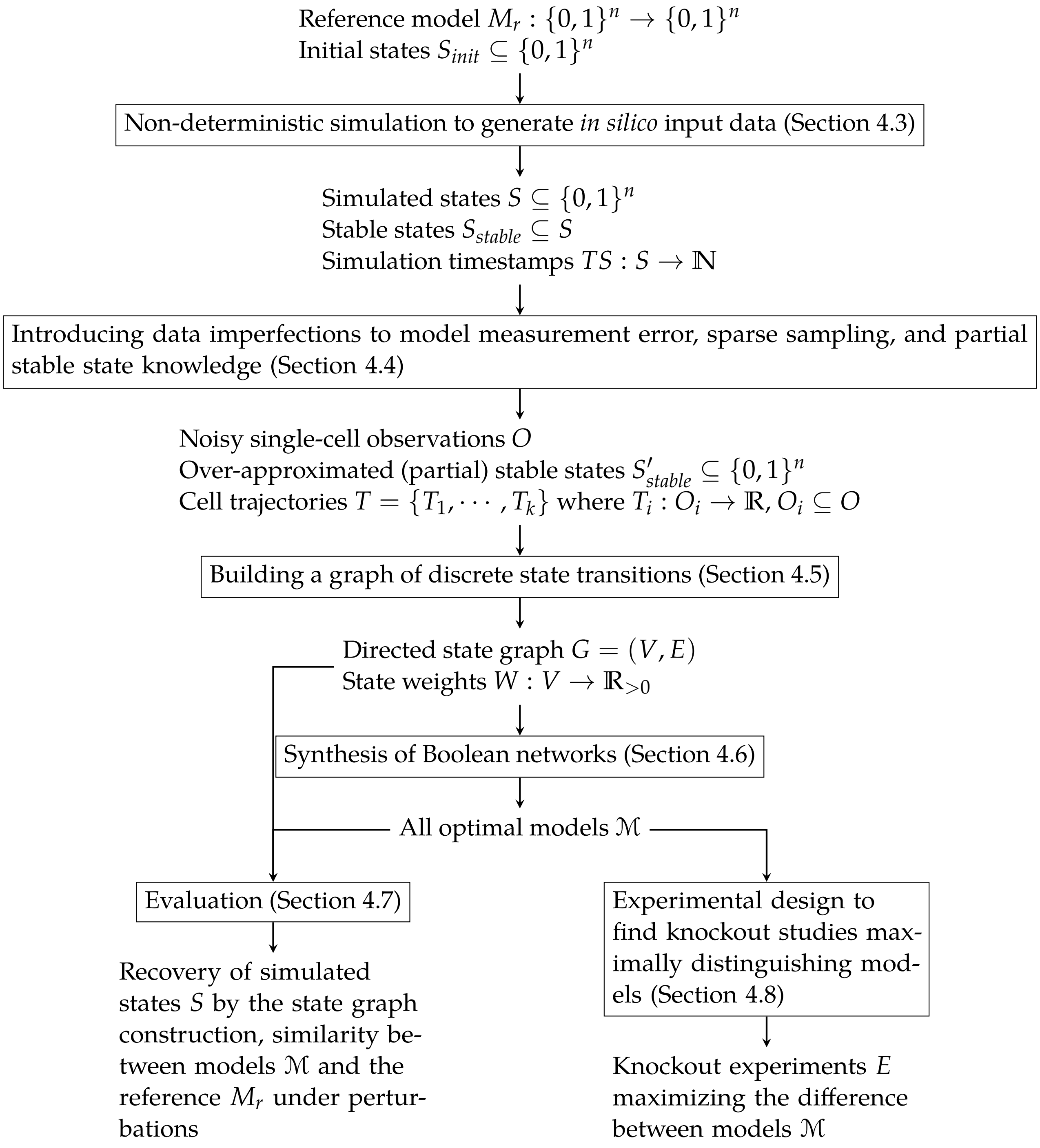

Reference model $M_r : \{0,1\}^n \to \{0,1\}^n$
Initial states $S_{init} \subseteq \{0,1\}^n$
Non-deterministic simulation to generate *in silico* input data (Section 4.3)
Simulated states $S \subseteq \{0,1\}^n$
Stable states $S_{stable} \subseteq S$
Simulation timestamps $TS : S \to \mathbb{N}$
Introducing data imperfections to model measurement error, sparse sampling, and partial stable state knowledge (Section 4.4)
Noisy single-cell observations O
Over-approximated (partial) stable states $S'_{stable} \subseteq \{0,1\}^n$
Cell trajectories $T = \{T_1, \cdots, T_k\}$ where $T_i : O_i \to \mathbb{R}, O_i \subseteq O$
Building a graph of discrete state transitions (Section 4.5)
Directed state graph $G = (V, E)$
State weights $W : V \to \mathbb{R}_{>0}$
Synthesis of Boolean networks (Section 4.6)
All optimal models $\mathcal{M}$
Evaluation (Section 4.7)
Recovery of simulated states S by the state graph construction, similarity between models $\mathcal{M}$ and the reference M_r under perturbations
Experimental design to find knockout studies maximally distinguishing models (Section 4.8)
Knockout experiments E maximizing the difference between models $\mathcal{M}$

Figure 4.1. Workflow overview.

The cell differentiation mechanism can typically be modeled as a system that transitions from an initial progenitor state to multiple stable attractor states describing Boolean gene expression levels in mature cell types. Using the formalism of Boolean networks, modeling the differentiation mechanism requires an asynchronous schedule policy in order to achieve non-deterministic behavior, as the network must reach different stable attractor states when executed from the same initial state. The asynchronous Boolean network formalism has been used in previous work for modeling cell differentiation [88, 159, 91] (Section 1.1).

In the first stage of the workflow, an asynchronous Boolean network M_r is used as a reference model to generate *in silico* input data. The model is executed from the set of initial states S_i using an asynchronous update schedule, until all reachable states have been explored. The outcome of this execution is the set of reachable states S, and state timestamps TS that maps each state in S to an integer value corresponding to the execution iteration in which that state was reached for the first time. The reachable states S correspond to a Boolean representation of single-cell gene expression measurements, while the timestamps TS approximate the output of a cell trajectory algorithm applied to S. We define the set of stable attractor states S_{stable} as the subset of reachable states that are mapped to themselves by each local activation function in the reference model M_r. In Section 4.3, we describe a modification to the execution model in order to more realistically model the system's intermediary behavior between the initial and final states.

The simulation of the asynchronous Boolean network gives an idealized view of experimental data, lacking measurement errors, and representing all states that the model went through during execution. In practice, experimental data sets can have measurement errors, leading to false positive states in the input data, and can be subject to sparse sampling, leading to false negatives in the observed state space. In order to model such data imperfections, we sample noisy observations O from S, given target type I and type II error rates. Each sample corresponds to an individual cell. Using the state timestamps TS that map each Boolean state to a timestamp, we generate a single cell trajectory that maps sampled observations to *pseudotime* values, establishing an order over observations. Generally, there might exist multiple cell trajectories $T = \{T_1, \cdots, T_k\}$, corresponding to distinct cell lineages appearing in the data. Each cell trajectory $T_i : O_i \to \mathbb{R}$ defines a total order over a subset of observations $O_i \subseteq O$. Furthermore, knowledge of the stable states S_{stable} that the system must reach may be partial, such that the value of some genes are unknown in the stable states. We over-approximate these partial states to capture all possible valuations of the unknown gene values. Section 4.4 describes these data transformations in detail.

After we obtain the *in silico* data set that captures imperfections in the input data, we perform model inference to obtain the set $\mathcal{M}$ of all models that agree with the data. Karme decomposes the model inference task into two sequential steps:

1. Building a state transition graph that represents a progression of cell states over time (Section 4.5). The output of this step is a directed graph $G = (V, E)$ where each vertex in

V corresponds to a distinct Boolean state in $\{0,1\}^n$, and $W : V \to \mathbb{R}_{>0}$, a mapping from vertices to weights, corresponding to the number of observations that map to each given state.

2. Synthesizing the set of models $\mathcal{M}$ that capture the state transitions observed in the graph G. $\mathcal{M}$ contains all models that satisfy optimality criteria regarding fit to data and syntactic properties (Section 4.6).

Performing model synthesis using a state transition graph has two added benefits compared to synthesizing models using only initial and final states [85]. First, the state transition graph gives us more information for inference of functions, because it rules out certain transition orders. The observed transition orders induce particular dependencies between variables. Such dependencies are not necessarily visible when we only observe initial and final states, and inference from state graphs can produce models that generalize to unobserved experimental conditions from a single "run" of the system. Second, using a state graph for synthesis paves the way for an efficient model inference method, where the problem of synthesizing the Boolean network can be decomposed into the inference of each Boolean function independently from the other functions. In contrast, without the knowledge of intermediary states, functions must be jointly inferred through a symbolic encoding of the system's execution.

We evaluate the output of both stages of model inference against the reference model M_r and its noise-free simulation.

- To evaluate state graph construction, we compare the set of states captured by G against the simulated states S.
- To evaluate the set of inferred models $\mathcal{M}$, we measure the similarity of the inferred Boolean functions to the reference, and compare the behavior of inferred models to the reference under unseen perturbations.
- Additionally, we evaluate the impact of prior knowledge about expected stable states by performing hold-out experiments (Section 4.7).

Finally, we characterize the model space $\mathcal{M}$ by searching for previously unobserved network perturbations under which inferred models exhibit different behavior. We identify perturbation experiments that lead to the maximum difference in the behavior of pairs of models, similarly to previous approaches to experimental design [9]. Adopting an enumerative approach to explore experiments, we demonstrate the value of using custom model difference metrics to highlight different key genes in modeled systems (Section 4.8).

4.3 Reference Model

We base our analysis on a well-established Boolean gene regulatory network model of myeloid progenitor cell differentiation [88]. Working with a synthetic model allows us to evaluate our two core algorithmic tasks: We can quantify the success of state graph reconstruction by comparing it to the state graph generated during simulation of the reference model, and we can evaluate model synthesis by comparing inferred models to the hidden reference model used to generate the data. Using simulated data also gives us the flexibility to vary noise and sampling characteristics of the data (Section 4.4), and evaluate their impact on our methods (Section 4.7).

This Boolean model aims to explain how the common myeloid progenitor (CMP) cell type is differentiated into four myeloid cell types: erythrocytes, megakaryocytes, monocytes, and granulocytes. Krumsiek et al. [88] identify and model 11 genes, which are transcription factors known to guide differentiation in this context. The model consists of one Boolean function for each gene, and is manually curated, based on a comprehensive literature survey ([88], Table 1).

The CMP cell type corresponds to what the authors call an "early, unstable undifferentiated state". The work focuses on executions of the model from this state only. States upstream of the initial state are considered "physiologically irrelevant", justifying the starting point of the modeled system. The four final cell types are *stable attractors* with respect to the Boolean model, i.e., all Boolean functions in the model map each final cell type state to itself. The correspondence between the expected cell types and Boolean states has been established by analyzing two independent microarray studies that give the expression profiles of the cell types.

Reaching all four cell types from the single initial state requires non-deterministic execution, since otherwise the model could reach at most one stable attractor state. As a result, the authors use asynchronous state updates for execution semantics (Section 1.3). Under this execution model, the Boolean model simulated from the initial state reaches four stable attractor states, which correspond to the four expected myeloid cell types.

The authors perform individual *in silico* knockouts (i.e. perturbation experiments) of each of the 11 modeled genes, and observe the set of reachable stable attractors for each knockout. Experimental data is available for 8 of the knockouts, confirming the model's behavior in reaching the correct set of stable attractor states for each knockout ([88], Figure 5.B). The effects of gene overexpression experiments is also discussed, and compared to observations from literature ([88], Supporting Information S4).

This Boolean model has been used as an *in silico* benchmark in further research: Woodhouse et al. [159] use the model for benchmarking the performance of an inference method based on single-cell data, treating each state reached in the unperturbed execution of the model from the CMP state as a single cell observation. The Boolean function for one gene ($C/EBP\alpha$) is semantically modified from the original model, due to syntactic limitations on Boolean expressions in [159]. The model has also been used as a benchmark in Lim et al. [91], where the authors take the model as a starting point and propose modifications to it through the use of additional single-cell qPCR and single-cell RNA-Seq data, via genetic programming. Yordanov et al. [166] also use the model as a case study, proposing modifications to the original model to better match the expressed value of Gata2 in the megakaryocyte stable state. The authors use a symbolic synthesis approach, similar to [85], using the wild-type stable attractor reachability constraints. However, the behavior of the model under knockout or overexpression studies is not discussed.

Adjusting the Execution Semantics

We begin our analysis with a study of the state space produced by the myeloid differentiation model under asynchronous Boolean network semantics. While Krumsiek et al. [88] mainly focuses on the stable states that the Boolean network converges to, we are interested in the biological plausibility of the intermediate states that the model visits. As we will see below, the simulation leads to superfluous execution paths that do not appear to faithfully model the intermediate states that the system goes through. We propose a modification to the execution semantics to help prevent problematic execution paths, and use the modified semantics to produce input data for our evaluation.

Executing the asynchronous Boolean network model produces a state graph with 160 nodes that correspond to unique Boolean states, and 477 edges that correspond to state transitions that occur during model execution. We note that this differs from the 232 nodes and 789 edges reported by Krumsiek et al. [88], while it matches the state graphs produced by Woodhouse et al. [159]. In this state graph, we observe paths corresponding to the (up or down) regulation of one gene, followed by the regulation of another, immediately followed by the opposite regulation of the first gene. An example can be seen in the subgraph shown in Figure 4.2. In this example, we observe that Pu_1 is downregulated from state s_1 to s_2. Meanwhile, there is an alternative path from s_1 to s_2 that goes through two intermediary states s_3 and s_4. Along this path, EgrNab, which is a factor that represents an integration of Egr-1, Egr-2 and Nab-2, is first upregulated, and is then downregulated. If we use the execution steps in which a state is reached as a proxy for the time of observations, the above kind of behavior leads to unexpected relative temporal orderings of states. For example, if state s_1 is reachable at time t, then s_2 can be reached at time $t+1$ through downregulation of Pu_1. Meanwhile, as the alternative path shows, s_3 and s_4 are reached at times $t+1$ and $t+2$, respectively. At the same time, s_2 is reachable from s_4 at time $t+3$. As a result, neither the minimum, nor the average time value for state s_2 allows us to properly orient the edge between s_2 and s_4.

To address this intermediary behavior issue, we propose two modifications to the execution semantics, resulting in a simpler state graph:

1. We allow distinct execution paths reaching the same state if their lengths from the initial node are equal. We only allow a unique, earliest time point for reaching each state. This condition prevents execution paths of the type described above, where an execution path leads to a state reachable through a shorter path.

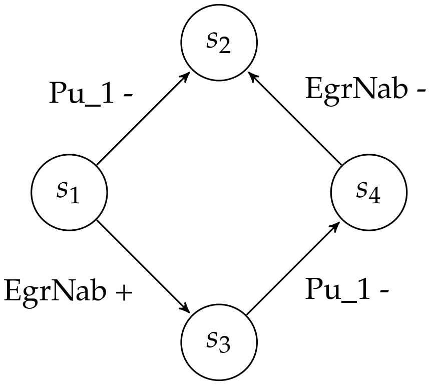


Figure 4.2. Subgraph of the state graph illustrating superfluous transitions. The state graph admits a path $s_1 \to s_3 \to s_4 \to s_2$ along which EgrNab is first upregulated, then downregulated.

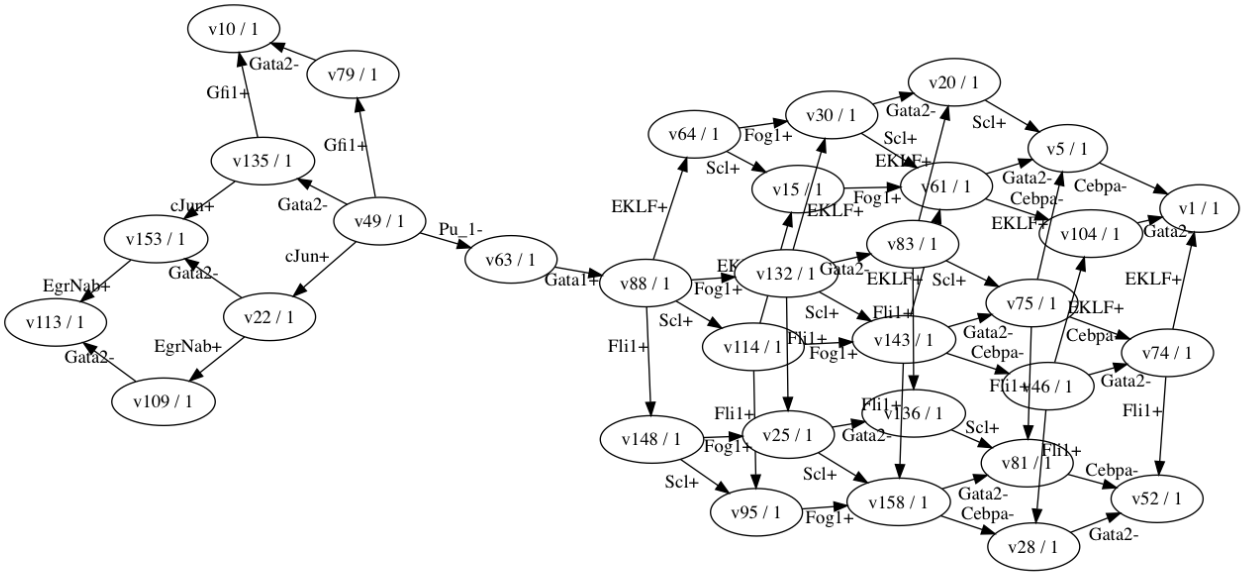


Figure 4.3. Trimmed state graph in which the undesired intermediate state behavior has been eliminated.

2. We rule out states that cannot reach any of the stable attractor states that were reachable from initial states under the original execution semantics. This results in the removal of additional states without outgoing edges introduced due to the first rule.

We call this modified execution model the *trimmed* execution semantics. The state graph produced by the trimmed execution semantics can be seen in Figure 4.3. This state graph is a subgraph of the state graph produced using the original execution semantics, and contains fewer non-deterministic interleavings of function applications. It contains 33 nodes and 58 edges, as opposed to 160 nodes and 477 edges for the former state graph.

In Section 4.4, we use trimmed execution semantics to generate synthetic single-cell data in our evaluation of the impact of sources of approximation on model synthesis. In Section 4.7, we use a hidden reference model that reproduces the trimmed state graph when it is executed with non-trimmed execution semantics. This new hidden reference model is obtained by running our synthesis procedure (Section 4.6) with the trimmed state graph as input.

4.4 Generating *in silico* single-cell observations

Our goal in this section is to generate input data that models measurement errors, sparse sampling, and partial prior knowledge about the stable attractor states that the system may reach.

The execution of the reference model M_r from the initial states S_{init}, as described in Section 4.3, produces the set of states $S \subseteq \{0,1\}^n$ that are reached during execution, and a function $T : S \rightarrow \mathbb{R}$ that maps each reached state to the first execution step in which it was observed. T represents a cell trajectory obtained through existing trajectory inference methods. We also define the stable attractor states of the execution as $S_{stable} = \{s \in S \mid \forall f_i \in M_r.\ f_i(s) = s\}$, i.e. the reachable states which are mapped to themselves by all local activation functions in the reference model M_r.

We transform the set of states S to a multiset O, representing multiple single-cell observations. Distinct observations may map to the same Boolean state, corresponding to cells in which the same genes are expressed according to a Boolean view of gene expression. To generate O, we first sample k instances of each state in S. We then remove observations from O via random sampling until a target type II error rate is reached. Finally, we add noisy observations to O, obtained by introducing bit flips to states randomly sampled from S, until a target type I error rate is achieved. This gives us an input data set of single-cell observations with controlled amounts of measurement error and sparse sampling. We exclude the initial and stable state sets from our random sampling of states during this transformation, with the assumption that sparse sampling cannot lead to missing these key states abundant at the beginning and the end of execution, respectively.

The cell trajectory $T : S \rightarrow \mathbb{R}$ is transformed into $T' : O \rightarrow \mathbb{R}$ by preserving the mapping of existing states, and defining the value for a noisy observation based on the state that was altered to obtain the observation.

During input data generation, we also optionally hide knowledge about the value of certain genes in the stable states. Precisely, to hide the value of gene a in the stable state knowledge, we duplicate the states in S_{stable} to include both values of a:

$$S'_{stable} = \bigcup_{s \in S_{stable}} \{s[a \mapsto 0], s[a \mapsto 1]\}$$

4.5 State Graph Construction

In this section, we describe how Karme builds a directed graph $G = (V, E)$ in which V is the set of discrete states that correspond to the observed single-cell measurements, and E is the set of state transitions that the system goes through. This state graph is built in presence of errors in the observed state space, i.e., missing observations and measurement errors.

Our goal is to obtain a state graph that uses edges with least Hamming distance to connect initial states with remaining states. This rules out using a simple approach where each state is linked with their immediate successors with respect to the cell trajectory information, since such an approach might lead to adding edges with large Hamming distance to noisy states. Instead, we design an iterative graph construction algorithm that prioritizes edges with least Hamming distance to ensure reachability, using the knowledge of both cell trajectories and the Hamming distance between discrete states.

The input components to the state graph construction are:

1. Single-cell observations O: A multiset of Boolean states, in which each element corresponds to one cell. Multiple cells may map to the same discrete state.
2. Cell trajectories $T = \{T_1, \cdots, T_k\}$: A set of partial functions that map observations to pseudotime values. Each trajectory T_i is a total order over a subset of measurements. Multiple trajectories represent branching lineages. Such cell trajectories are inferred by cell reordering algorithms.
3. Optionally, a candidate set of stable states S_{stable}. If provided, all states without outgoing transitions in the constructed state graph must appear in S_{stable}.

Cell trajectories as strict partial orders. In order to build state graphs in which edge orientations are consistent with cell trajectories, we transform the cell trajectories into a strict partial order $\prec$ over discrete cell states. Conceptually, to relate Boolean states x and y, we compare pseudotemporal values for all observations corresponding to each state on common cell trajectories. We say that $x \prec y$ if and only if the pseudotemporal sample for y is not less than those for x for any trajectory, and there exists at least one trajectory for which the pseudotemporal values for x are less than those for y.

Precisely, given cell trajectories $T_i : O_i \to \mathbb{R}$, where each trajectory assigns pseudotemporal values to a subset O_i of all observations O, we define the order $\prec$ as:

$$\begin{aligned} x \prec y \doteq \forall\, T_i \in T.\, (x \in O_i \ \wedge y \in O_i) \implies p_{T_i}(y < x) > \mathit{threshold} \\ \wedge \exists\, T_i \in T.\, x \in O_i \ \wedge y \in O_i \wedge p_{T_i}(x < y) \leq \mathit{threshold} \end{aligned}$$

In the above, p_{T_i} is the probability of $x < y$ given all the pseudotemporal values that correspond to states x and y in trajectory T_i. This probability is computed by a sample comparison method, such as rank-sum or Kolmogorov-Smirnov.

Finding initial states. Given a definition of a strict partial order over discrete states, the initial states are defined as minimal elements with respect to the partial order, i.e. there exists no state that is less than any initial state according to the order $\prec$:

$$I = \{s : \nexists u.\, u \prec s\}$$

Iterative graph construction. State graph construction is an iterative procedure that aims to connect initial states to the remaining states by minimal Hamming distance:

1. Start with partial-order-minimal states as the initial vertex set.
2. Repeat until no more edges can be added:
   a) Saturation step: Add all d-Hamming edges from already reachable nodes to any node, where d is the least Hamming distance between a reachable node and any other node.
   b) Extension step: Add all d'-Hamming edges from already reachable nodes to currently unreachable nodes, where d' is the least Hamming distance between currently reachable and unreachable nodes. This step allows the graph construction algorithm to extend graph connectivity by considering multi-Hamming edges that cannot be added in the saturation step.
3. If a candidate set of stable states S_{stable} is provided, remove all graph nodes that do not have a path to any stable state in S_{stable}.

This iterative approach allows us to build state graphs that connect states using edges that have as small Hamming distance as possible. Edges of a given Hamming distance d are only added to make it possible to reach states that could not be reached via edges with Hamming distance less than d. We note that, due to the strict partial order $\prec$ being transitive, the resulting state graphs may not contain any cycles. Our approach is akin to minimum spanning tree approaches to build cell lineage graphs [128], but differs from them in its handling of cells as discrete states.

In Section 4.7, we evaluate the performance of the state graph reconstruction algorithm by measuring its ability to recover the Boolean states S that appear in the original simulation.

4.6 Best-Fit Synthesis from State Graphs

In this section, we describe a synthesis procedure to infer mechanistic models that explain the progression of discrete cell states in a system over time. The input to the synthesis procedure is a directed state graph $G = (V, E)$ with node weights W. Our goal is to infer a set of asynchronous Boolean network models $\mathcal{M}$ that can reproduce G when it is executed from its initial states.

Fisher et al. [159] introduced the idea of viewing discretized single-cell data as a state space from which a Boolean network can be reconstructed. In this work, we generalize this approach to handle state graphs in which edges might connect pairs of states that are separated by a Hamming distance greater than 1. Additionally, while previous work relies on finding shortest paths from initial to final states to orient the state graph, we aim to maximally fit state transitions induced by G, whose edges have been oriented with the help of pseudotemporal information during state graph construction (Section 4.5).

The state graph $G = (V, E)$ induces a (partial) truth table for every modeled variable. For each state $v_i \in V$, if there is an outgoing edge from v_i to v_j along which the k-th gene changes its value, then this edge constitutes an input-output constraint for the activation function f_k in the form of $f_k(v_i) = v_j$. If, on the other hand, v_i does not have an outgoing edge along which gene k changes its value, then we have a constraint of the form $f_k(v_i) = v_i$. A model in which each local update function satisfies all of its induced constraints is guaranteed to perfectly reproduce G if it is executed from the initial states of G.

Meanwhile, it may be infeasible to represent the induced truth tables as Boolean update functions in presence of syntactic limitations on models, such as a bound on the depth of function expression trees. Imposing a limit on function complexity is both important from a scalability perspective, and can act a means of regularization to avoid overfitting to state graphs inducing inconsistent state transitions due to data noise. When such inconsistencies occur, our strategy is to bias satisfying state transitions between frequently observed states. We therefore derive constraint weights from the node weights W, and infer models that satisfy sets of constraints with maximal weight sum. The weight of the constraint induced by the edge (v_i, v_j) is defined as the product of node weights $W(v_i) \cdot W(v_j)$.

A key strategy for the scalable synthesis of Boolean networks is the decomposition of the model synthesis task into the inference of individual local update functions. If each local activation function satisfies all of its input-output constraints induced by the state graph G, then they jointly form a Boolean network model that can perfectly reproduce G when executed from its initial states.

In the remainder, we describe our approach to constraint satisfaction and resolving inconsistencies in the data, and we characterize the properties of the models $\mathcal{M}$ that we enumerate.

Transforming Multi-Hamming Edges for Synthesis

The directed graph built using the state graph construction procedure described in Section 4.5 may contain edges between states that are separated by a Hamming distance greater than 1. We interpret these multi-Hamming-distance edges as independent transitions of multiple variables.

In order to achieve compatibility between multi-Hamming transitions and the asynchronous Boolean network execution model that schedules one local update function application at a time, we transform each multi-Hamming edge into a set of 1-Hamming-distance edges expressing an arbitrary interleaving of function applications. This transformation favors the inference of models that do not potentially get stuck in an intermediary state, depending on the order of function application.

Specifically, for each n-Hamming distance edge $(v_i, v_j$, we model all $n!$ paths by introducing an n-dimensional hypercube whose nodes correspond to variable flips from v_i to v_j, and whose edges are oriented from v_i to v_j. If a Boolean network model perfectly reproduces the state transitions along the hypercube, then its nondeterministic execution from the state v_i will reach v_j irrespective of the order in which local activation functions are applied.

Viewing the State Graph as Hard and Soft Constraints

The state graph G induces two types of constraints for each variable k. For each vertex $v \in V$, G induces a *switching* constraint for k if v has an outgoing edge along which k switches its value, and a *looping* constraint if there exists no such edge. In this work, we treat switching constraints as hard constraints that must be satisfied by inferred models, and looping constraints as soft constraints that are maximally satisfied. This strategy prioritizes the reachability of states in G (through the satisfaction of switching constraints), and provides a best-effort approach to avoid the production of states that are not in G (through the satisfaction of looping constraints).

We define the weight of each constraint as the product of the weights of its endpoints. As a result, looping constraints induced by more frequently observed states are assigned a larger weight, and are prioritized in the synthesis process.

Synthesis with Maximally Satisfied Soft Constraints

Once we have weighted hard and soft constraints induced by G, we search for models that satisfy all hard constraints, and soft constraints with a maximal sum of weights. If the hard constraints for a variable are not consistent, we partition the set of hard constraints into maximally consistent subsets of constraints, and perform the synthesis procedure on each subset individually. The partitioning is a greedy procedure that expands the current subset of constraints with the constraint with the next largest weight, until all constraints have been considered.

For each hard constraint subset, we find a maximal set of soft constraints that are consistent with the hard constraints given the syntactic bounds on the model search space, i.e., the size of inferred Boolean functions. For finding a maximal set of soft constraints, we again adopt a greedy approach, considering each soft constraint by decreasing weight.

Once a maximally satisfiable set of constraints is obtained, we enumerate all minimal models that satisfy them. Model minimality is defined as each update function having (1) minimum expression tree height, and (2) the minimum number of variables in function expressions. Therefore, we obtain a set of models $\mathcal{M}$ that is the union of all minimal models for eeach hard constraint subset and its associated soft constraint subset with maximal sum of weights.

4.7 Evaluating Model Inference from Imperfect Data

In this section, we evaluate the performance of model inference from single-cell data in presence of data imperfections. As described in Section 4.4, we model data imperfections as false positives and false negatives in the input data, as well as uncertainty about the potential stable states that the system being modeled is expected to reach.

As we work with *in silico* data obtained by simulating a reference model, we are able to precisely quantify the success of both state graph construction and model synthesis steps. To evaluate state graph construction, we compare the inferred state graph $G = (V, E)$ to reference model simulation data before introducing data imperfections. To evaluate model synthesis, we compare the inferred models $\mathcal{M}$ to the reference model M_r both according to the similarity of inferred Boolean functions, and according to model behavior under previously unseen perturbations. As this section will present, this evaluation strategy allows us to observe that increased performance in state graph construction leads to better generalizability in model behavior.

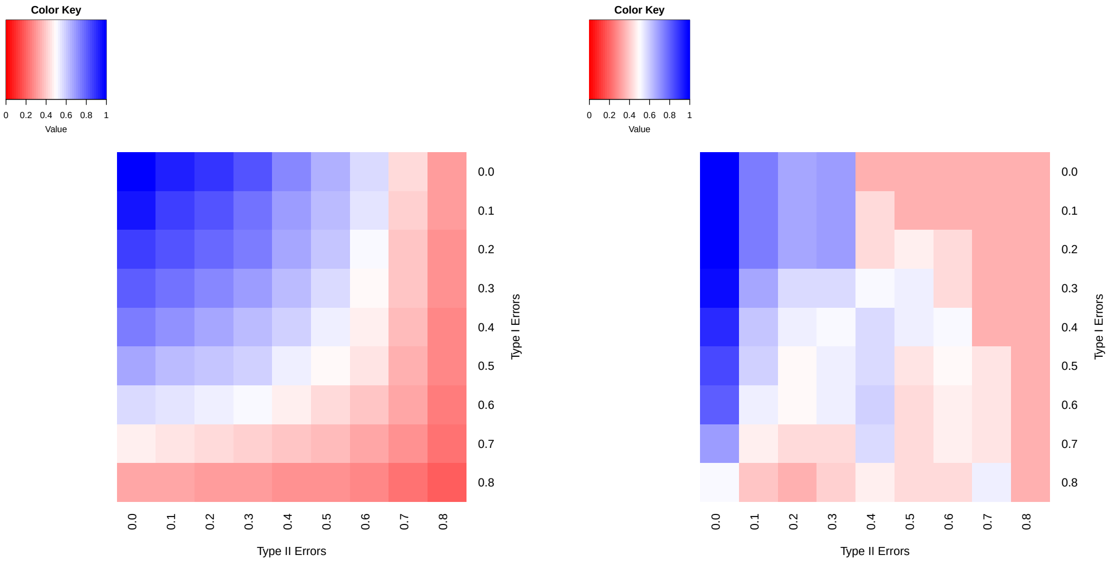
Evaluating State Graph Construction

Evaluating state graph construction consists precisely in comparing the set of states recovered as the vertex set V to the set of simulated states S before introducing type I and type II errors to the input data. For each combination of type I and type II error ratios, we run five replicates of the workflow with different seeds for random number generation. We measure Boolean state recovery by computing the F_1 score (the harmonic mean of precision and recall) where we compare V against the ground truth S. We present results as heatmaps showing the median value over all replicates for each pair of type I and type II error ratios.

Figure 4.4a shows the F_1 scores for Boolean states observed in the noisy input. As expected, input data without any type I or type II errors (upper left corner of the graph) achieves a perfect F_1 score of 1. and the F_1 score decreases as we increase the type I or type II error ratio. Figure 4.4b shows the F_1 scores for Boolean states that appear in the state graph G.

We observe that state graph construction is particularly resilient against false positives in the input Boolean states, in the absence of false negatives. Upon closer examination, we observe that the last state graph construction step that filters out vertices that do not have a path to any stable state substantially removes false positive states introduced into the data.

We also observe an amelioration of the median F_1 score for high type I and type II error ratios in the reconstructed state graphs compared to input data (lower right quadrant). The breakdown of the F_1 score into the precision and recall components is shown in Figure 4.5b and Figure 4.6b. As this breakdown shows, both precision and recall scores are improved in the reconstructed state graphs compared to input data characteristics. We attribute the improvement in precision to the ability to exclude false positive Boolean states in state graph construction, and the improvement in recall, to the recovery of false negative states influenced by the higher



(a) F_1 scores for input data

(b) F_1 scores for state graphs

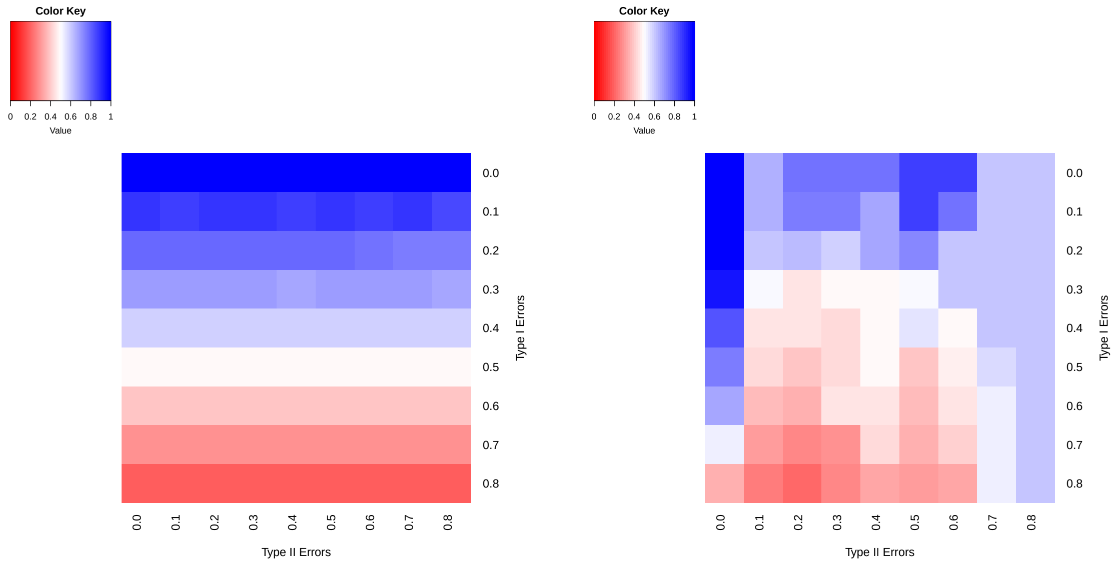
Figure 4.4. F_1 scores for Boolean states as they appear in the input data, compared to F_1 scores for Boolean states that appear in the constructed state graphs. We observe that state graph construction is able to recover states in S that do not appear in the input, for the cases where the input data contains false positive states but no false negative states.

amount of Boolean states observed (by addition of hypercubes of states to the state graph), even though some of the observed states are false positives.

Evaluating Model Synthesis

Next, we investigate model synthesis performance under the type I and type II error conditions explored above for evaluating state graph construction. In each setting, we compare the collection of inferred models $\mathcal{M}$ to the reference model M_r based on their dynamic behavior under the unperturbed and previously unseen, perturbed contexts. In addition, we compare the inferred Boolean expressions to their reference counterpart according to how similarly they behave on the same input, by comparing their truth tables. For our analyses, we approximate the set of inferred models by a random sample of 10 models from the enumerated solution space. In 301 runs out of 405 runs, there exist strictly fewer than 10 enumerated models; in 151 runs, a unique model is inferred.

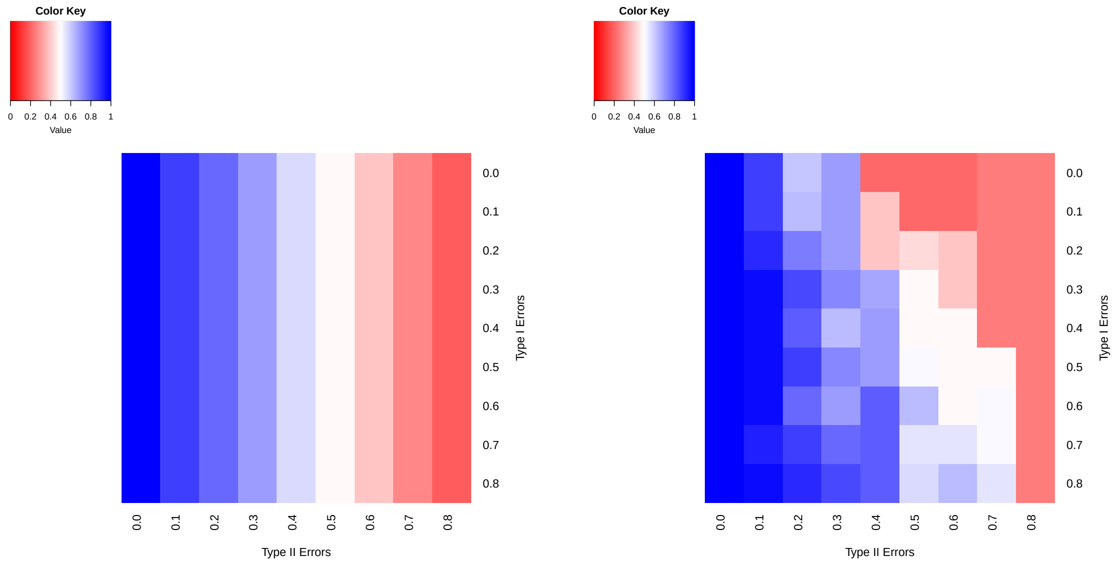
We first measure the ability of inferred models to reproduce the stable state behavior of the reference model. For each inferred model, we compare the stable states reached by the model to those reached by the reference, and compute F_1 scores to quantify success. We perform this comparison for both the wild-type setting and the eight genetic knockout perturbations for which we have experimental evidence ([88], Figure 5b). Figure 4.7 shows the results in



(a) Precision scores for input data

(b) Precision scores for state graphs

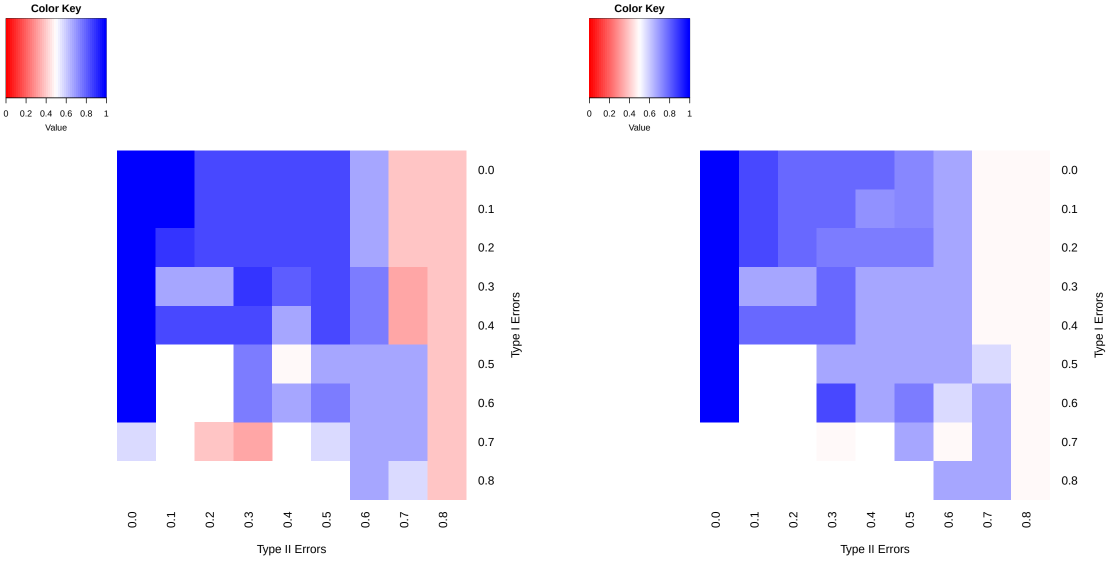
Figure 4.5. Precision scores for Boolean states as they appear in the input data, compared to precision scores for Boolean states that appear in the constructed state graphs.



(a) Recall scores for input data

(b) Recall scores for state graphs

Figure 4.6. Recall scores for Boolean states as they appear in the input data, compared to recall scores for Boolean states that appear in the constructed state graphs.



(a) Wild-type stable state recovery

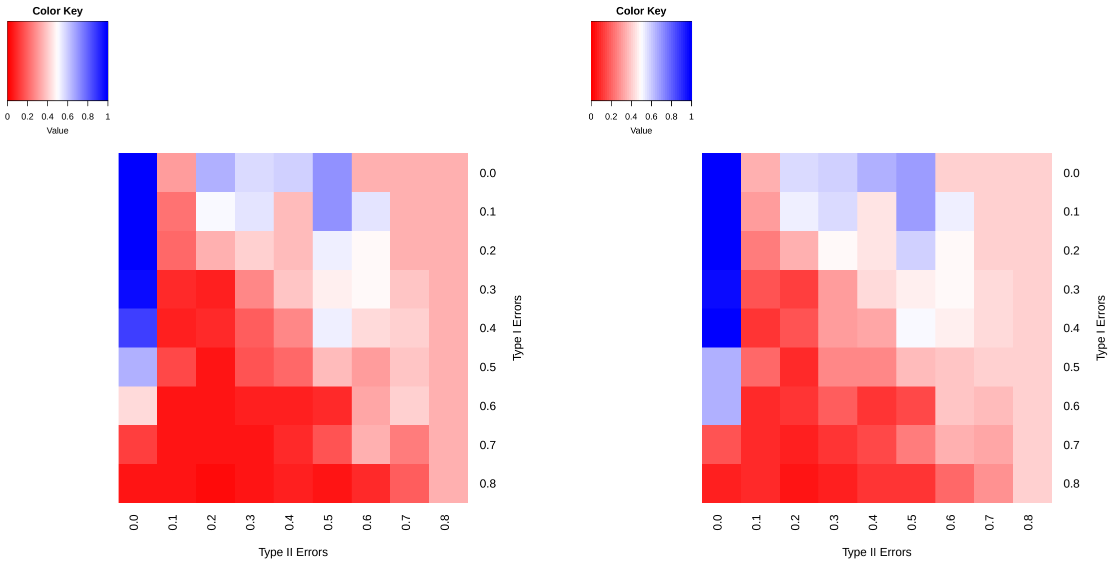
(b) Knockout stable state recovery

Figure 4.7. F_1 scores for stable state recovery by inferred models in the wild-type (unperturbed) case and knockout (perturbed) cases.

both settings. The F_1 scores shown in Figure 4.7a are median values across all models in five replicates in wild-type, and the scores shown in Figure 4.7b are median values across all models and all perturbations in five replicates.

For stable state recovery in the wild-type case, we observe a pattern of success similar to the state graph construction success. We note that with the knowledge of desired stable states, it is possible to filter the reconstructed state graph to only include those specific states as nodes without outgoing edges, and the synthesis procedure is able to find models that can reach the stable states and avoid making transitions from them. For the case of previously unseen perturbations, we note that synthesis is able to generalize to accurately predict stable state behavior under knockout experiments. For large values of the type II error ratio, we observe an improvement in predicting stable states in the perturbed settings compared to the wild-type setting. We attribute this to the smaller set of stable states reached by the reference model in perturbed settings, making it easier for inferred models to capture the expected stable states.

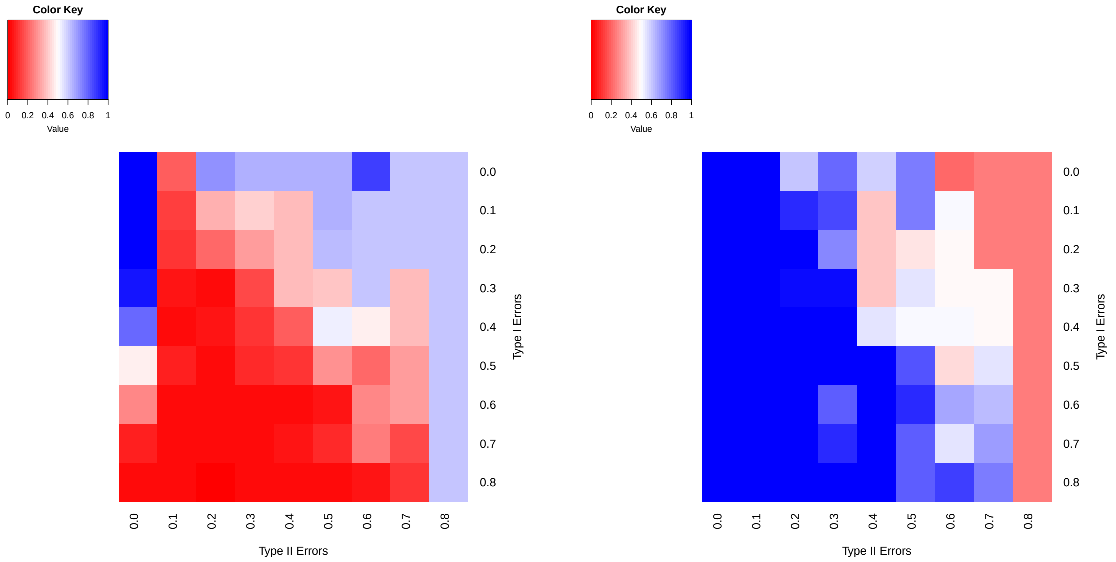
We then extend this analysis to the recovery of all reachable states via model execution. Figure 4.8 shows the F_1 score for the Boolean states reached by executing inferred models against the reference reachable states. We observe that synthesized models are able to generalize from the wild-type case to the previously unseen perturbations. We also observe better reachable state reproducibility for higher ratios of type II errors, compared to higher ratios of type I errors. To get more insight into this pattern, we break down the F_1 scores into precision and recall scores in Figure 4.9. This breakdown shows that while the models can achieve good recall for high amounts of false positives, the precision score greatly reduces the F_1 score in this setting.



(a) Wild-type reachable state recovery

(b) Knockout reachable state recovery

Figure 4.8. F_1 scores for reachable state recovery by inferred models in the wild-type (unperturbed) case and knockout (perturbed) cases.



(a) Wild-type reachable state precision

(b) Wild-type reachable state recall

Figure 4.9. Precision and recall scores for reachable state recovery by inferred models in the wild-type (unperturbed) case.

We further evaluate the inferred models $\mathcal{M}$ by comparing the inferred Boolean activation functions to the reference functions in M_r. We define two function similarity metrics. The first metric, $similarity_b$, measures the similarity between each inferred Boolean function and its reference counterpart according to the number of input states for which the two functions produce the same output value. We perform this comparison on the set S of states that are reached by the reference model M_r. Precisely, the similarity between the Boolean functions f_i and and f'_i is defined as:

$$similarity_b(f_i, f'_i) = \frac{|S_{match}|}{|S|}$$

where:

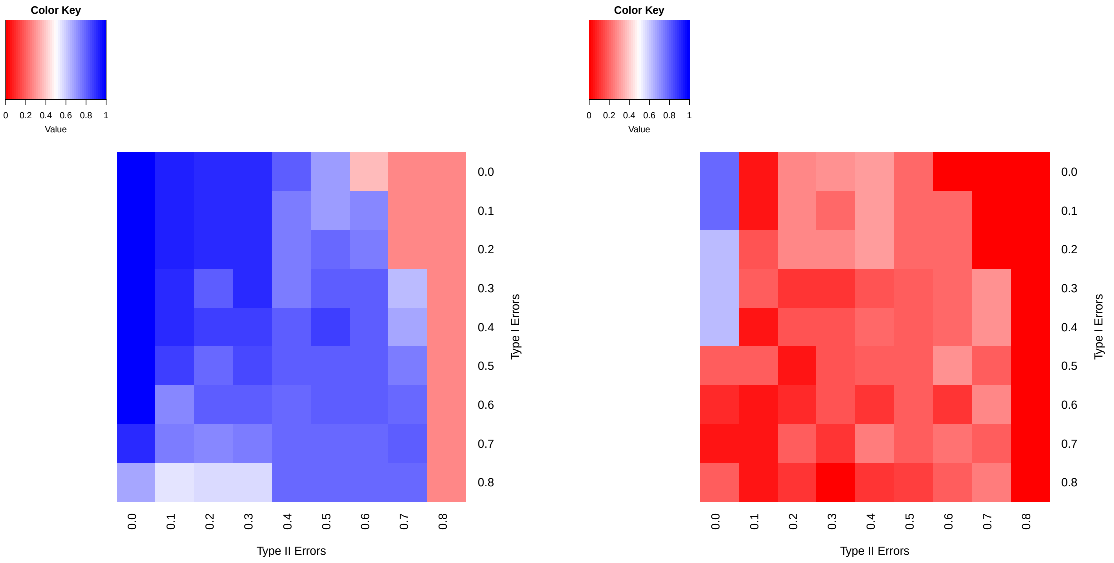
$$S_{match} = \{s \in S : f_i(s) = f'_i(s)\}$$

The second similarity metric we compute, $similarity_{I/O}$, abstracts over the structure of Boolean functions and compares an inferred model M to the reference model M_r based on the set of input-output dependencies that appear in the local activation functions. To do this, we aggregate all input-output pairs over the local activation functions of a model, define the input-output pair similarity between two models as the Jaccard index (intersection over union) of the two input-output pair sets.

Figure 4.10 shows the median function similarity results over all inferred models in all replicates for every combination of type I and type II error ratio parameters. The function behavior similarity metric $similarity_b$, which evaluates inferred functions according to their behavior on the set of simulated states S (in contrast with the full state space), shows a high degree of correspondence between the inferred models and the reference for low levels of false negative states (Figure 4.10a). Behavior similarity is lowest for the highest amounts of type II errors in the input data. This corresponds to cases where we see also the worst state graph reconstruction performance, due to the reconstructed state graph G missing the largest amount of simulated states (Figure 4.6b). We observe that the $similarity_{I/O}$ metric is lower in absolute terms, since it evaluates inferred models based on functional dependencies that are not necessariy observed in the input (wild-type) execution of the system.

Impact of Stable State Knowledge

Since we observed a high impact of the knowledge of candidate stable states on the recovery from high amounts of false positive states, we extended our analysis to evaluating the effect of partially hiding the stable state knowledge in the input. The set S_{stable} is a set of Boolean states that correspond to gene expression values observed in mature cell types. This knowledge can be obtained by measuring mRNA expression levels in each mature cell type ([88], Figure 4). Our methodology for removing partial stable state knowledge is to make the input stable states partial, making individual genes' value unknown in the set of stable states S_{stable} (Section 4.4).



(a) Function behavior similarity

(b) Function I/O pair similarity

Figure 4.10. Similarity of inferred functions with the reference model.

We hid each variable's value individually in the stable states, and in each case, we aggregated evaluation metrics over a range of type I and type II error ratios. We limited this range to type II error ratios of 0 and 0.1, focusing on the cases where the input data may have a high amount of false positive states, but always has a low amount of false negative states.

Figure 4.11 shows the F_1 scores for Boolean state recovery in reconstructed state graphs. This analysis does not show a significant difference in graph construction performance when values of individual genes are held out from the input stable state knowledge. However, as we expand our analysis to evaluating the behavior of inferred models, we notice decreased performance when knowledge about certain genes are held out. Recovery of stable state behavior is worse in most cases where we hold out a single gene's stable state values, with the exception of EKLF. (Figure 4.12). We note that the value of EKLF only changes on execution paths that lead to a single stable state in the original model execution, which may explain why holding out knowledge about its stable state behavior does not affect algorithm performace. For the rest of the genes, decreased performance is due to changes in both the precision and recall of recovered stable states (Figure 4.13).

Finally, we analyze input-output pair similarity between inferred models and the reference model when stable state knowledge about individual genes is held out. Similarly to our analysis of stable state recovery, we observe a decrease in performance when an individual gene's value is hidden in the stable state knowledge, for all genes except EKLF. Our analysis reveals the importance of stable state knowledge in recovering the structure and the end-to-end behavior of inferred models.

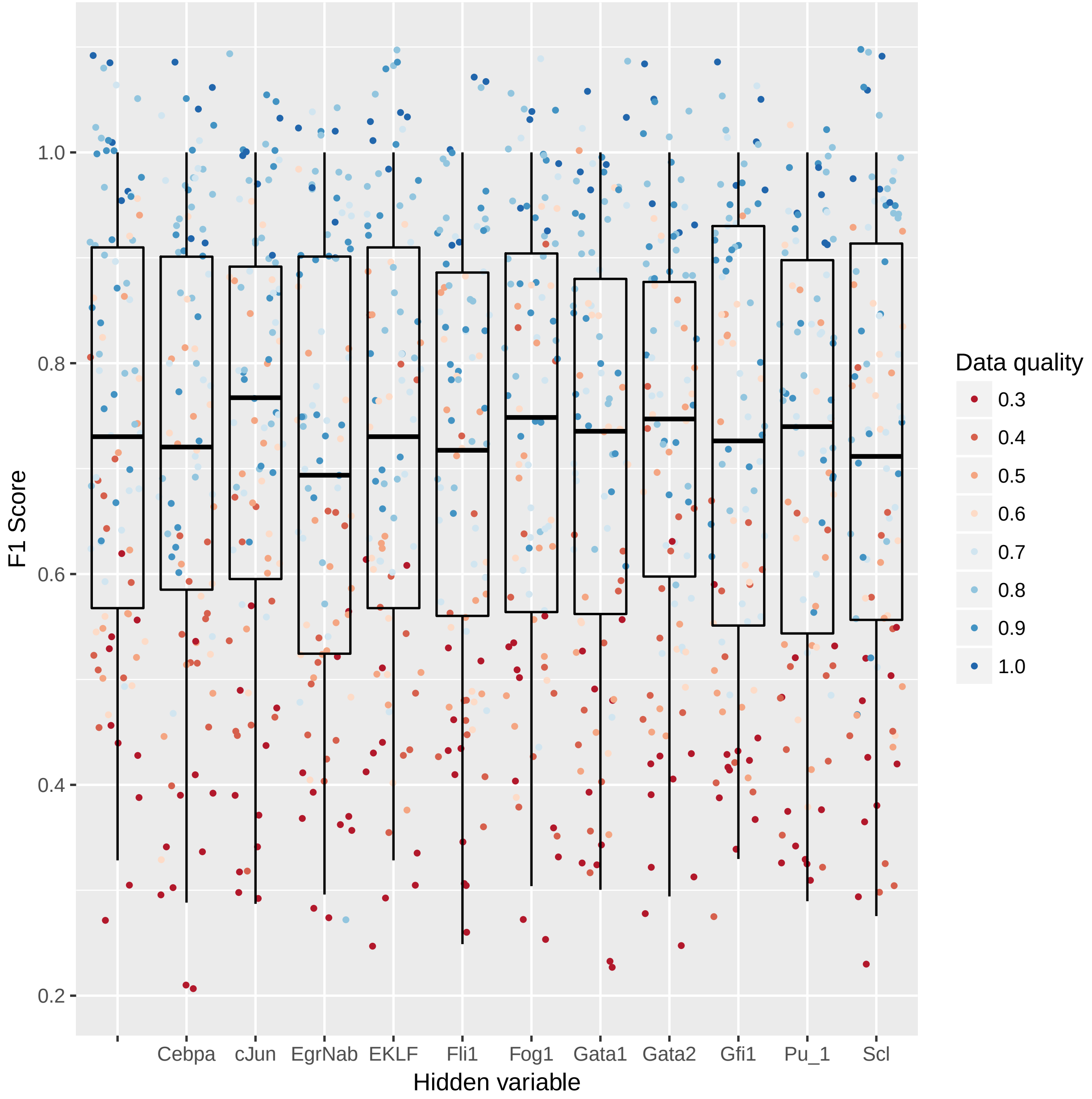
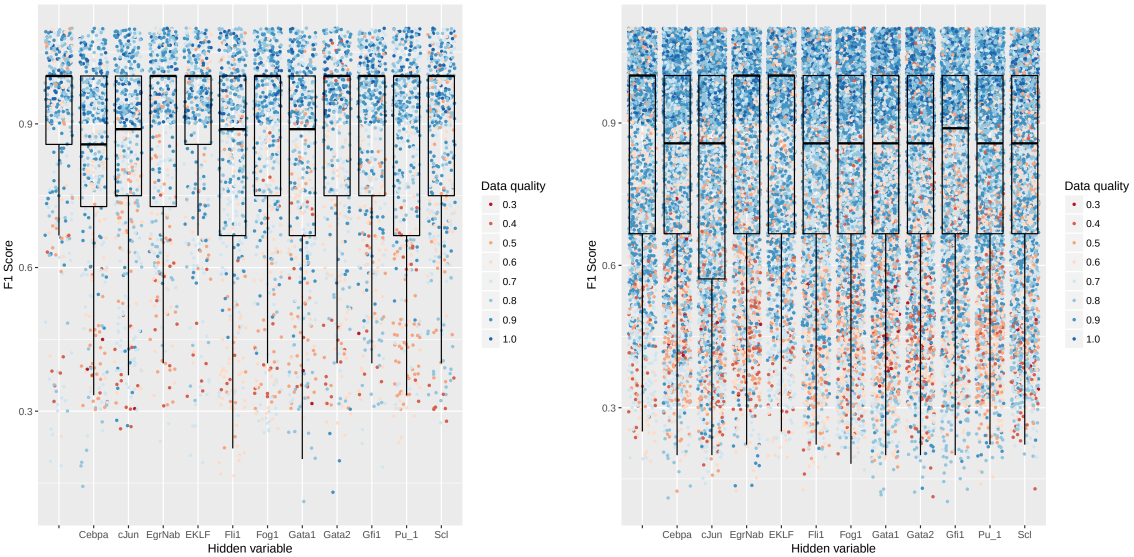


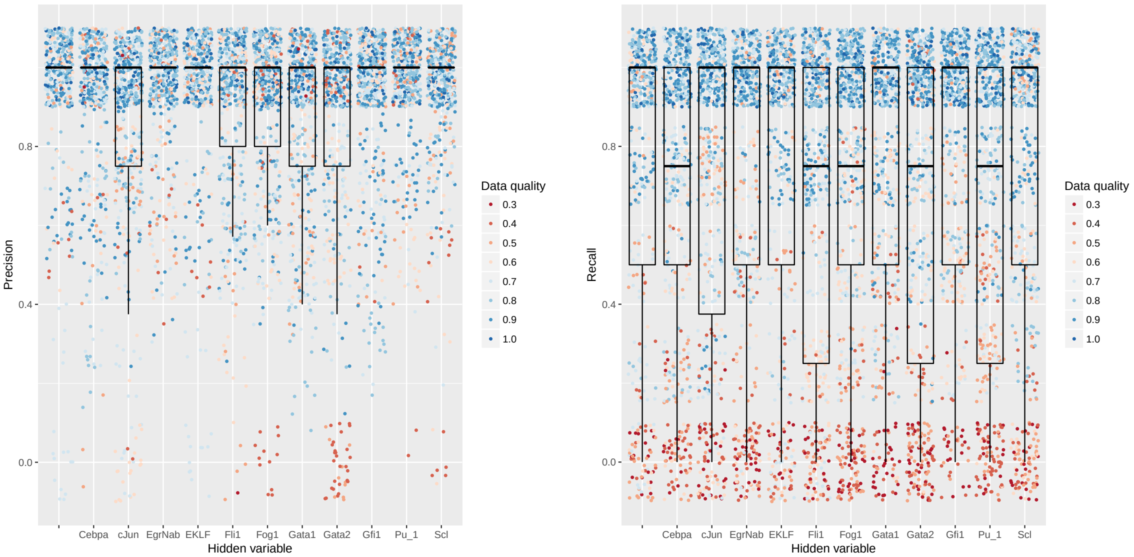
Figure 4.11. F_1 scores for Boolean states recovered by state graph construction, for partially hidden stable state knowledge.



(a) Wild-type stable state recovery

(b) Knockout stable state recovery

Figure 4.12. F_1 scores for stable state behavior recovery in the wild-type and knockout cases, for partially hidden stable state knowledge.



(a) Wild-type stable state precision

(b) Wild-type stable state recall

Figure 4.13. Precision and recall scores for stable state behavior recovery in the wild-type case, for partially hidden stable state knowledge.

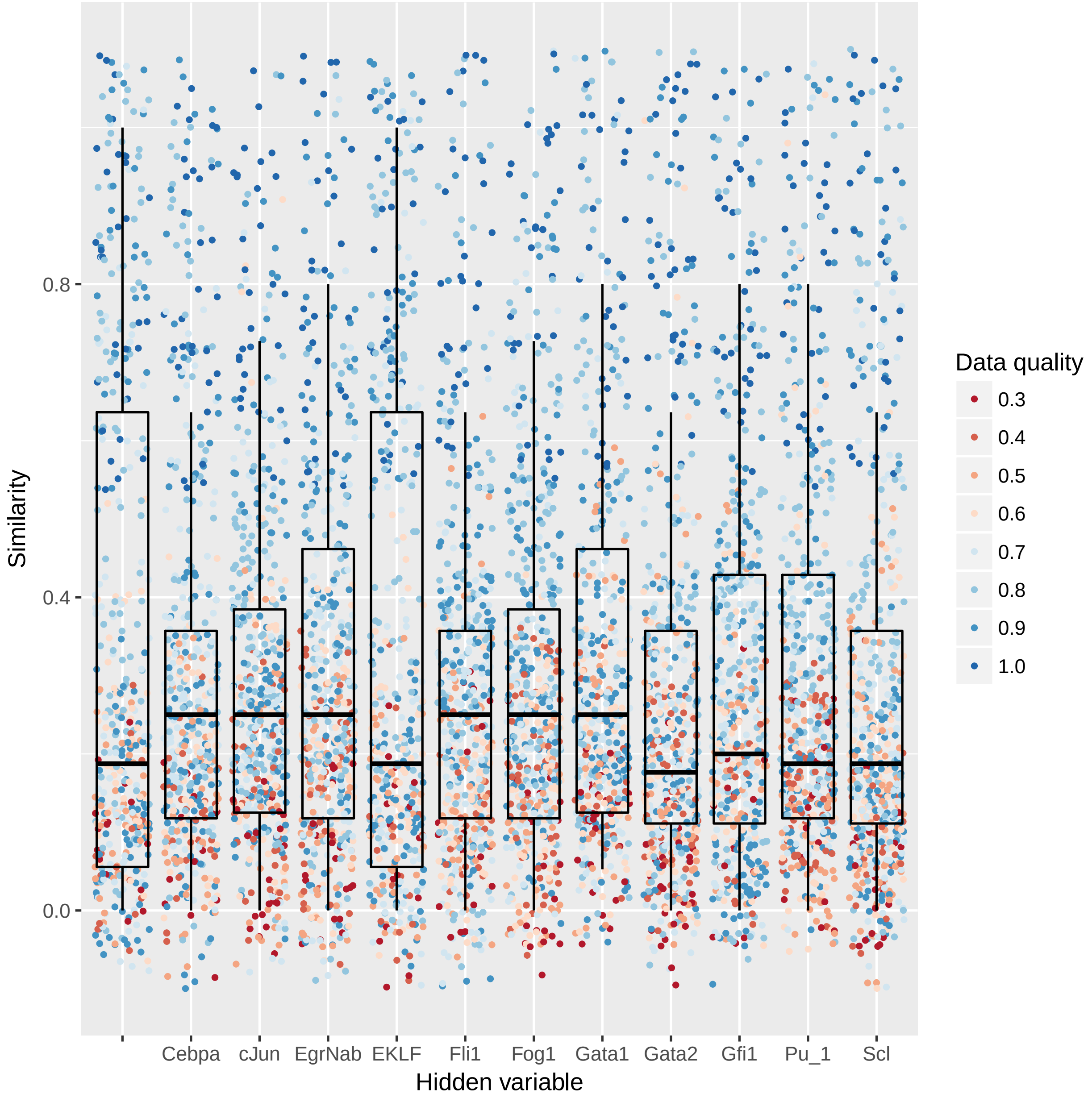


Figure 4.14. I/O pair similarity of inferred functions with the reference model, for partially hidden stable state knowledge.

4.8 Experimental Design with User-Defined Objectives

In this section, we investigate the ability to distinguish between inferred models through perturbation experiments. Our goal is to design *in silico* methods to guide the experimental process, and reduce the ambiguity in the model space as a result.

In order to design an experimental design approach, we need to define a space of experiments that can be performed on models, and a metric to measure the difference between models under a given experimental condition. The language of experiments can typically be defined as all combinations of gene mutations [85, 9]. An example distinguishing factor between models is the difference between stable state behavior of models for a given mutation. With the help of symbolic solvers, one can efficiently find mutation experiments for which there exists non-deterministic executions of distinct models leading to different stable state outcomes, without explicitly enumerating all models [85]. Furthermore, if the modeling formalism is restricted to non-cyclic models that are executed deterministically, one can formulate an integer linear programming task that symbolically searches for the experiment that leads to the maximum stable state difference between any two models in the solution space [9].

In this work, we propose an experimental design approach for distinguishing asynchronous Boolean networks with cyclic dependencies. Our approach allows for user-defined model difference metrics. As we show below, considering alternative model difference metrics can reveal key entities in inferred models, which may not be identified by solely observing the stable state behavior of models. Similarly to [9], we identify experiments that enable observing the maximum difference between pairs of models among all inferred ones; unlike [9], our approach can handle cyclic, asynchronous Boolean network models.

We tackle the challenge of identifying maximum difference experiments for user-defined model difference metrics for asynchronous, cyclic Boolean network models, using an approach that performs an enumerative exploration of mutation experiments on an explicitly enumerated set of inferred models. We note that symbolically searching for maximum difference experiments without explicitly enumerating experiments and models would pose a substantial challenge, since it would require a symbolic encoding of end-to-end model behavior, and a symbolic characterization of model difference metrics. The scalability of our synthesis approach relies on the decomposition of the model synthesis task into the synthesis of individual local activation functions, without a symbolic representation of whole execution traces.

For our analysis, we define the space of experiments as single-gene knockout studies, where the knockout experiment for gene a consists in overriding the Boolean function f_a with the Boolean constant function *false*, and executing the model from the set of initial states in which a's value has been set to *false* as well. We define two criteria for distinguishing a pair of models under the same experimental condition. The first metric compares the set of stable states reached by each model, and the second one compares the set of all states reached during execution. For two sets of states S_1 and S_2, we define the distance metric to be $1 - d(S_1, S_2)$, where d measures the Jaccard index between two sets. The distance between two models is 0 when the set of predicted states are identical, and it is 1 when the sets are disjoint.

We limited the maximum size of inferred Boolean functions to expression trees of height 3, and computed the maximum model difference among synthesized models across the range of type I and type II error ratios explored in Section 4.7. For each error ratio combination, we ran five replicate runs. In each setting, we analyzed the space of models for the two model distinguishability metrics. Figure 4.15 shows model distinguishability results according to the dissimilarity between reachable states, and Figure 4.16 shows the results for the dissimilarity between stable states.

We observe that in the absence of noise in input data, the six inferred models are indistinguishable from each other through single-gene knockout experiments, because they produce the same set of states when simulated under each perturbation. To distinguish among these models, it is necessary to exercise different execution paths through a perturbation to the initial states, or to perform multiple-gene knockout experiments.

For each run replicate with noisy data, we choose the knockout experiment that achieves the maximum difference between pairs of models among all inferred models. If there exists no such experiment for a replicate, we omit the results. We observe that using the reachable states difference metric identifies Gata2 as a key gene: The Gata2 mutation is the most distinguishing experiment in most cases (Figure 4.15a), and the maximum model difference it achieves ranks higher than most other single-gene mutation experiments (Figure 4.15b). This observation is consistent with the fact that Gata2 is an early hematopoietic factor in the differentiation of myeloid progenitors, and plays an active role in the differentiation of the common myeloid progenitor cell type into all four myeloid cell types 4.3.

When we perform the same analysis with the stable state model difference metric, we do not observe any genes that has a comparable impact in distinguishing inferred models (Figure 4.16). We note that this metric compares state sets of relatively smaller size than the reachable states difference metric, and as a result produces fewer distinct model difference values.

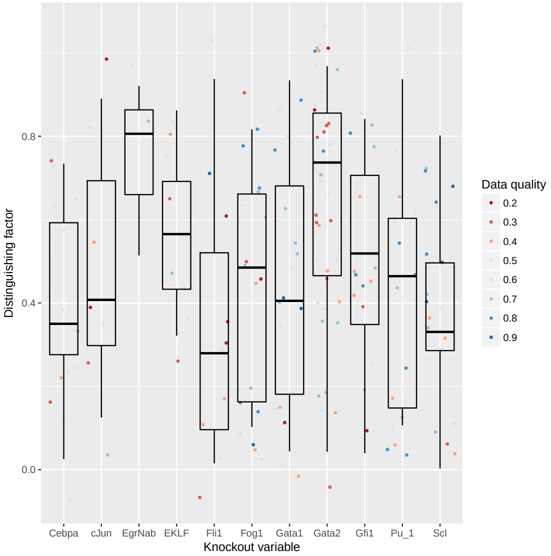
This analysis illustrates the significance of the ability to explore user-defined model difference metrics, as different metrics may differ in their ability to identify key genes for experimental design.

4.9 Conclusion

Discrete modeling of biological systems enables efficient model synthesis approaches that can exhaustively explore spaces of models that are consistent with experimental data. We presented an investigation of Boolean network synthesis from single-cell measurements, focusing on the impact of data quality and prior knowledge on the performance of model inference. Additionally, we desribed an experimental design appraoch that explores a space of perturbation experiments for user-defined model difference metrics.

Gene	Nb. most dist. exp.
Gata2	43
Gata1	18
Fog1	18
Scl	17
Pu_1	16
Gfi1	15
Cebpa	15
Fli1	10
EKLF	8
cJun	7
EgrNab	3

(a) Number of most distinguishing experiments

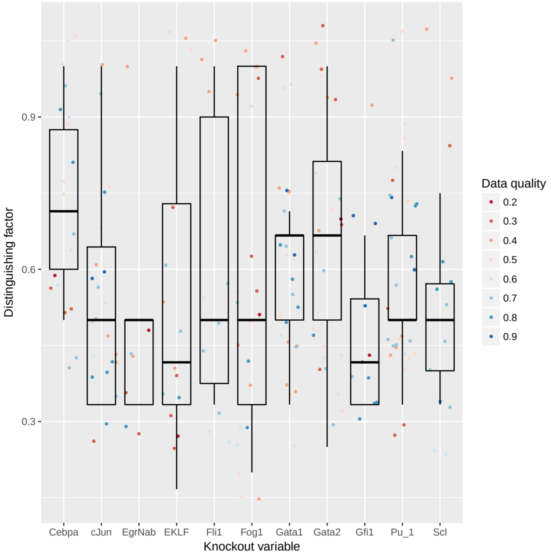


(b) Maximum model distances for most distinguishing experiments

Figure 4.15. Maximum difference results for each knockout experiment, based on the difference between set of reachable states. The analysis identifies the mutation of Gata2, an early hematopoietic factor, as the most distinguishing experiment in 42 cases.

Gene	Nb. most dist. exp.
Pu_1	31
Gata1	25
cJun	24
Gata2	24
Fog1	21
Cebpa	19
EKLF	14
Scl	13
Gfi1	12
Fli1	10
EgrNab	7

(a) Number of most distinguishing experiments



(b) Maximum distinguishing experiments

Figure 4.16. Maximum difference results for each knockout experiment, based on the difference between set of stable states.

Our empirical study on an *in silico* model of myeloid differentiation showed that knowledge of stable states of a system can boost synthesis performance in situations where there is a high degree of false positive observations in the input data. Furthermore, our analysis showed the value of customizability of model difference metrics, revealing key players in modeled systems when novel distinguishability metrics are used.

Chapter 5

Conclusion

This thesis explored program synthesis approaches to model biological systems from various types of experimental data, prior knowledge, and domain-specific assumptions. We presented three systems for the automated inference of Boolean network models. Using symbolic reasoning, our methods exhaustively explore spaces of models, and help guide the experimental process for refining the computational models that express the current biological knowledge about the modeled systems.

We expect improvements in the quality of experimental data (e.g. a decrease in data noise, and an increase in data resolution) to have a positive impact on the performance of automated model inference methods. Furthermore, as we showed throughout this dissertation, the declarative nature of specifications in program synthesis approaches makes them amenable to asserting an increasing amount of prior knowledge, coming from increasingly varied sources of data.

Bibliography

[1] http://www.cs.berkeley.edu/~koksal/.

[2] https://oeis.org/A000670.

[3] Rajeev Alur and Thomas A. Henzinger. “Reactive Modules”. In: *Formal Methods in System Design* 15.1 (1999), pp. 7–48.

[4] Benedict Anchang et al. “Modeling the Temporal Interplay of Molecular Signaling and Gene Expression by Using Dynamic Nested Effects Models”. In: *Proceedings of the National Academy of Sciences* 106.16 (Apr. 2009), pp. 6447–6452. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.0809822106.

[5] Julio Aracena et al. “On the robustness of update schedules in Boolean networks”. In: *Biosystems* 97.1 (2009), pp. 1–8. DOI: 10.1016/j.biosystems.2009.03.006. URL: http://dx.doi.org/10.1016/j.biosystems.2009.03.006.

[6] Gustavo Arellano et al. “Antelope: a hybrid-logic model checker for branching-time Boolean GRN analysis.” In: *BMC bioinformatics* 12.1 (2011). http://turing.iimas.unam.mx:8080/AntelopeWEB/content/about.jsp, p. 490.

[7] A. Arkin, J. Ross, and H. H. McAdams. “Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected Escherichia coli cells”. In: *Genetics* 149.4 (Aug. 1998), pp. 1633–1648.

[8] Anil Aswani et al. “Nonparametric identification of regulatory interactions from spatial and temporal gene expression data”. In: *BMC Bioinformatics* 11 (2010), p. 413.

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

[10] Rhonda Bacher and Christina Kendziorski. “Design and computational analysis of single-cell RNA-sequencing experiments”. In: *Genome biology* 17.1 (2016), p. 63.

[11] Albert-László Barabási, Natali Gulbahce, and Joseph Loscalzo. “Network medicine: a network-based approach to human disease”. In: *Nature Reviews Genetics* 12.1 (2011), pp. 56–68.

[12] Ziv Bar-Joseph, Anthony Gitter, and Itamar Simon. "Studying and modelling dynamic biological processes using time-series gene expression data". In: *Nature Reviews Genetics* 13.8 (Aug. 2012), pp. 552–564. ISSN: 1471-0056. DOI: 10.1038/nrg3244.

[13] Nathan A. Barker, Chris J. Myers, and Hiroyuki Kuwahara. "Learning Genetic Regulatory Network Connectivity from Time Series Data". In: *IEEE/ACM Trans. Comput. Biology Bioinform.* 8.1 (2011), pp. 152–165.

[14] Grégory Batt, Calin Belta, and Ron Weiss. "Temporal Logic Analysis of Gene Networks under Parameter Uncertainty". In: *IEEE Transactions of Automatic Control* (), p. 2008.

[15] Grégory Batt et al. "Analysis and Verification of Qualitative Models of Genetic Regulatory Networks: A Model-Checking Approach". In: *IJCAI*. 2005.

[16] Anna Bauer-Mehren, Laura I. Furlong, and Ferran Sanz. "Pathway databases and tools for their exploitation: benefits, current limitations and challenges". In: *Molecular Systems Biology* 5.1 (Jan. 2009). PMID: 19638971, p. 290. ISSN: 1744-4292, 1744-4292. DOI: 10.1038/msb.2009.47.

[17] Sean C. Bendall et al. "Single-Cell Mass Cytometry of Differential Immune and Drug Responses Across a Human Hematopoietic Continuum". In: *Science* 332.6030 (May 2011). PMID: 21551058, pp. 687–696. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1198704.

[18] Steven A. Benner and A. Michael Sismour. "Synthetic biology". In: *Nature Reviews Genetics* 6.7 (July 2005), pp. 533–543. ISSN: 1471-0056. DOI: 10.1038/nrg1637.

[19] David Benque et al. "BMA: Visual tool for modeling and analyzing biological networks". In: *Computer Aided Verification*. http://biomodelanalyzer.research.microsoft.com/. Springer. 2012, pp. 686–692.

[20] Gilles Bernot et al. "Application of formal methods to biological regulatory networks: extending Thomas' asynchronous logical approach with temporal logic". In: *Journal of theoretical biology* 229.3 (2004), pp. 339–347.

[21] Gungor Budak et al. "Reconstruction of the temporal signaling network in Salmonella-infected human cells". In: *Frontiers in Microbiology* 6 (2015), p. 730. DOI: 10.3389/fmicb.2015.00730.

[22] Lulu Cao et al. "Quantitative Phosphoproteomics Reveals SLP-76 Dependent Regulation of PAG and Src Family Kinases in T Cells". In: *PLoS ONE* 7.10 (Oct. 2012), e46725. DOI: 10.1371/journal.pone.0046725.

[23] Daniel E. Carlin. "Computational evaluation and derivation of biological networks in cancer and stem cells". PhD thesis. University of California, Santa Cruz, 2014. URL: http://gradworks.umi.com/36/88/3688771.html.

[24] Nathalie Chabrier and François Fages. "Symbolic Model Checking of Biochemical Networks". In: CMSB '03. 2003.

[25] C. Chaouiya. "Petri net modelling of biological networks". In: *Brief. Bioinformatics* 8.4 (July 2007), pp. 210–219.

[26] Deborah Chasman et al. "Pathway connectivity and signaling coordination in the yeast stress-activated signaling network". In: *Molecular Systems Biology* 10.11 (Nov. 2014). PMID: 25411400, p. 759. ISSN: 1744-4292, 1744-4292. DOI: `10.15252/msb.20145120`.

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

[28] Chunaram Choudhary and Matthias Mann. "Decoding signalling networks by mass spectrometry-based proteomics". In: *Nature Reviews Molecular Cell Biology* 11.6 (June 2010), pp. 427–439. ISSN: 1471-0072. DOI: `10.1038/nrm2900`.

[29] L. Church, K. Apagyi, and J. Fisher. "Languages for Biological Models: Importance, Implications and Challenges - A Work In Progress". In: *Proceedings of the Psychology of Programming Interest Group*. Jan. 2008. URL: `http://research.microsoft.com/apps/pubs/default.aspx?id=68096`.

[30] Mark F. Ciaccio et al. "Systems analysis of EGF receptor signaling dynamics with microwestern arrays". In: *Nature Methods* 7.2 (Feb. 2010), pp. 148–155. ISSN: 1548-7091. DOI: `10.1038/nmeth.1418`.

[31] Mark F. Ciaccio et al. "The DIONESUS algorithm provides scalable and accurate reconstruction of dynamic phosphoproteomic networks to reveal new drug targets". In: *Integrative Biology* 7.7 (July 2015), pp. 776–791. ISSN: 1757-9708. DOI: `10.1039/C5IB00065C`.

[32] Koen Claessen et al. "Model-checking signal transduction networks through decreasing reachability sets". In: *Computer Aided Verification*. Springer. 2013, pp. 85–100.

[33] Byron Cook et al. "Finding Instability in Biological Models". In: *Computer Aided Verification*. 2014, pp. 358–372.

[34] Byron Cook et al. "Proving stabilization of biological systems". In: *Verification, Model Checking, and Abstract Interpretation*. Springer. 2011, pp. 134–149.

[35] Vincent Danos et al. "Abstract interpretation of cellular signalling networks". In: VMCAI'08, pp. 83–97.

[36] Hidde De Jong. "Modeling and simulation of genetic regulatory systems: a literature review". In: *Journal of computational biology* 9.1 (2002), pp. 67–103.

[37] Leonardo De Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. TACAS'08/ETAPS'08. Springer-Verlag, 2008, pp. 337–340. ISBN: 3-540-78799-2. URL: `http://dl.acm.org/citation.cfm?id=1792734.1792766`.

[38] Riet De Smet and Kathleen Marchal. "Advantages and limitations of current network inference methods". In: *Nature Reviews Microbiology* 8.10 (Oct. 2010), pp. 717–729. ISSN: 1740-1526. DOI: `10.1038/nrmicro2419`.

[39] David L. Dill. "Model Checking Cell Biology". In: *CAV*. 2012, p. 2.

[40] Rochelle C. J. D'Souza et al. "Time-resolved dissection of early phosphoproteome and ensuing proteome changes in response to TGF-*beta*". In: *Science Signaling* 7.335 (July 2014). PMID: 25056879, rs5. ISSN: 1945-0877, 1937-9145. DOI: `10.1126/scisignal.2004856`.

[41] S. J. Dunn et al. "Defining an essential transcription factor program for naïve pluripotency". In: *Science* 344.6188 (June 2014), pp. 1156–1160. ISSN: 1095-9203. DOI: `10.1126/science.1248882`. URL: `http://dx.doi.org/10.1126/science.1248882`.

[42] Adrien Fauré et al. "Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle". In: *Bioinformatics* 22.14 (2006), e124–e131.

[43] Jasmin Fisher, David Harel, and Thomas A. Henzinger. "Biology as reactivity". In: *Commun. ACM* 54.10 (2011), pp. 72–82.

[44] Jasmin Fisher and Thomas A Henzinger. "Executable cell biology". In: *Nature biotechnology* 25.11 (2007), pp. 1239–1249.

[45] Jasmin Fisher and Thomas A. Henzinger. "Executable cell biology". In: *Nature Biotechnology* 25.11 (Nov. 2007), pp. 1239–1249. ISSN: 1087-0156.

[46] Jasmin Fisher and Nir Piterman. "Model Checking in Biology". In: *A Systems Theoretic Approach to Systems and Synthetic Biology I: Models and System Characterizations*. Ed. by Vishwesh V. Kulkarni, Guy-Bart Stan, and KarthikEditors Raman. Springer Netherlands, 2014, pp. 255–279. ISBN: 978-94-017-9040-6. URL: `http://link.springer.com/chapter/10.1007/978-94-017-9041-3_10`.

[47] Jasmin Fisher, Nir Piterman, and Rastislav Bodik. "Toward synthesizing executable models in biology". In: *Frontiers in Bioengineering and Biotechnology* 2 (2014). PMID: 25566538PMCID: PMC4271700, p. 75. ISSN: 2296-4185. DOI: `10.3389/fbioe.2014.00075`.

[48] Jasmin Fisher et al. "Bounded Asynchrony: Concurrency for Modeling Cell-Cell Interactions". In: *FMSB*. 2008, pp. 17–32.

[49] Jasmin Fisher et al. "Predictive Modeling of Signaling Crosstalk during *C. elegans* Vulval Development". In: *PLoS Computational Biology* 3.5 (2007). DOI: `10.1371/journal.pcbi.0030092`. URL: `http://dx.doi.org/10.1371/journal.pcbi.0030092`.

[50] J. Fisher et al. "Predictive modeling of signaling crosstalk during C. elegans vulval development". In: *PLoS Comput. Biol.* 3.5 (May 2007), e92.

[51] Russell B Fletcher et al. "Deconstructing olfactory stem cell trajectories at single-cell resolution". In: *Cell stem cell* 20.6 (2017), pp. 817–830.

[52] Holger Fröhlich et al. "Deterministic Effects Propagation Networks for reconstructing protein signaling networks from multiple interventions". In: *BMC Bioinformatics* 10.1 (Oct. 2009). PMID: 19814779, p. 322. ISSN: 1471-2105. DOI: `10.1186/1471-2105-10-322`.

[53] Leon A Furchtgott et al. "Discovering sparse transcription factor codes for cell states and state transitions during development". In: *eLife* 6 (2017), e20488.

[54] Thomas D Garvey et al. "BioSPICE: access to the most current computational tools for biologists". In: *OMICS A Journal of Integrative Biology* 7.4 (2003), pp. 411–420.

[55] Anthony Gitter and Ziv Bar-Joseph. "Identifying proteins controlling key disease signaling pathways". In: *Bioinformatics* 29.13 (July 2013). PMID: 23812988, pp. i227–i236. ISSN: 1367-4803, 1460-2059. DOI: `10.1093/bioinformatics/btt241`.

[56] Anthony Gitter et al. "Discovering pathways by orienting edges in protein interaction networks". In: *Nucleic Acids Research* 39.4 (Mar. 2011), e22. DOI: `10.1093/nar/gkq1207`.

[57] Anthony Gitter et al. "Linking the signaling cascades and dynamic regulatory networks controlling stress responses". In: *Genome Research* 23.2 (Feb. 2013), pp. 365–376. ISSN: 1088-9051, 1549-5469. DOI: `10.1101/gr.138628.112`.

[58] Sumit Gulwani. "Automating string processing in spreadsheets using input-output examples". In: *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. POPL '11. Austin, Texas, USA: ACM, pp. 317–330.

[59] Simone Gupta et al. "Boolean network analysis of a neurotransmitter signaling pathway". In: *Journal of theoretical biology* 244.3 (2007), pp. 463–469.

[60] Carito Guziolowski et al. "Exhaustively characterizing feasible logic models of a signaling network using Answer Set Programming". In: *Bioinformatics* 29.18 (Sept. 2013). PMID: 23853063, pp. 2320–2326. ISSN: 1367-4803, 1460-2059. DOI: `10.1093/bioinformatics/btt393`.

[61] J. Heath et al. "Probabilistic model checking of complex biological pathways". In: *Theoretical Computer Science* 319.3 (2008), pp. 239–257.

[62] David Henriques et al. "Data-driven reverse engineering of signaling pathways using ensembles of dynamic models". In: *PLOS Computational Biology* 13.2 (Feb. 2017), e1005379. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1005379.

[63] Steven M. Hill et al. "Context Specificity in Causal Signaling Networks Revealed by Phosphoprotein Profiling". In: *Cell Systems* 4.1 (Jan. 2017). PMID: 28017544, 73–83.e10. ISSN: 2405-4712. DOI: 10.1016/j.cels.2016.11.013.

[64] Steven M. Hill et al. "Inferring causal molecular networks: empirical assessment through a community-based effort". In: *Nature Methods* 13.4 (Apr. 2016), pp. 310–318. ISSN: 1548-7091. DOI: 10.1038/nmeth.3773.

[65] Andrew Hinton et al. "PRISM: A Tool for Automatic Verification of Probabilistic Systems". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Holger Hermanns and JensEditors Palsberg. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 441–444. ISBN: 978-3-540-33056-1. URL: http://link.springer.com/chapter/10.1007/11691372_29.

[66] Andrew L Hopkins. "Network pharmacology: the next paradigm in drug discovery". In: *Nature chemical biology* 4.11 (2008), pp. 682–690.

[67] Shao-shan Carol Huang and Ernest Fraenkel. "Integrating proteomic, transcriptional, and interactome data reveals hidden components of signaling and regulatory networks". In: *Science signaling* 2.81 (2009), ra40–ra40.

[68] Sean J. Humphrey, S. Babak Azimifar, and Matthias Mann. "High-throughput phosphoproteomics reveals in vivo insulin signaling dynamics". In: *Nature Biotechnology* 33.9 (Sept. 2015), pp. 990–995. ISSN: 1087-0156. DOI: 10.1038/nbt.3327.

[69] Martijn P van Iersel et al. "Presenting and exploring biological pathways with PathVisio". In: *BMC bioinformatics* 9.1 (2008), p. 399.

[70] Siddhartha Jain et al. "Reconstructing the temporal progression of HIV-1 immune response pathways". In: *Bioinformatics* 32.12 (June 2016). PMID: 27307624, pp. i253–i261. ISSN: 1367-4803, 1460-2059. DOI: 10.1093/bioinformatics/btw254.

[71] Sumin Jang et al. "Dynamics of embryonic stem cell differentiation inferred from single-cell transcriptomics show a series of transitions through discrete cell states". In: *eLife* 6 (2017), e20487.

[72] Hawoong Jeong et al. "Lethality and centrality in protein networks". In: *Nature* 411.6833 (2001), pp. 41–42.

[73] Kyuri Jo et al. "Influence maximization in time bounded network identifies transcription factors regulating perturbed pathways". In: *Bioinformatics* 32.12 (June 2016). PMID: 27307609, pp. i128–i136. ISSN: 1367-4803, 1460-2059. DOI: 10.1093/bioinformatics/btw275.

[74] Na'aman Kam, Irun R. Cohen, and David Harel. "The Immune System as a Reactive System: Modeling T Cell Activation With Statecharts". In: *HCC*. 2001, pp. 15–22.

[75] Na'aman Kam et al. "Formal Modeling of C. elegans Development: A Scenario-Based Approach". In: *CMSB*. 2003, pp. 4–20.

[76] Evgeny Kanshin et al. "A Cell-Signaling Network Temporally Resolves Specific versus Promiscuous Phosphorylation". In: *Cell Reports* 10.7 (Feb. 2015), pp. 1202–1214. ISSN: 2211-1247. DOI: `10.1016/j.celrep.2015.01.052`.

[77] Joost-Pieter Katoen, Maneesh Khattri, and Ivan S Zapreev. "A Markov reward model checker". In: *Second International Conference on the Quantitative Evaluation of Systems*. IEEE, Sept. 2005, pp. 243–244. DOI: `10.1109/QEST.2005.2`.

[78] Stuart A Kauffman. "Metabolic stability and epigenesis in randomly constructed genetic nets". In: *Journal of theoretical biology* 22.3 (1969), pp. 437–467.

[79] Alex Khodaverdian et al. "Steiner Network Problems on Temporal Graphs". In: *arXiv:1609.04918 [cs]* (Sept. 2016). arXiv: 1609.04918. URL: `http://arxiv.org/abs/1609.04918`.

[80] Narsis A. Kiani and Lars Kaderali. "Dynamic probabilistic threshold networks to infer signaling pathways from time-course perturbation data". In: *BMC Bioinformatics* 15.1 (July 2014). PMID: 25047753, p. 250. ISSN: 1471-2105. DOI: `10.1186/1471-2105-15-250`.

[81] Min-Sik Kim et al. "A draft map of the human proteome". In: *Nature* 509.7502 (May 2014), pp. 575–581. ISSN: 0028-0836. DOI: `10.1038/nature13302`.

[82] Ali Sinan Köksal, Viktor Kuncak, and Philippe Suter. "Scala to the Power of Z3: Integrating SMT and Programming". In: *CADE*. 2011, pp. 400–406.

[83] Ali Sinan Köksal, Viktor Kuncak, and Philippe Suter. "Scala to the Power of Z3: Integrating SMT and Programming". In: *Automated Deduction – CADE-23*. Ed. by Nikolaj Bjørner and VioricaEditors Sofronie-Stokkermans. Vol. 6803. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 400–406. ISBN: 978-3-642-22437-9. URL: `http://link.springer.com/chapter/10.1007/978-3-642-22438-6_30`.

[84] Ali Sinan Köksal et al. "Synthesis of Biological Models from Mutation Experiments". In: *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '13. ACM, 2013, pp. 469–482. ISBN: 978-1-4503-1832-7. DOI: `10.1145/2429069.2429125`. URL: `http://doi.acm.org/10.1145/2429069.2429125`.

[85] Ali Sinan Köksal et al. "Synthesis of biological models from mutation experiments". In: *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*. 2013, pp. 469–482. DOI: `10.1145/2429069.2429125`. URL: `http://doi.acm.org/10.1145/2429069.2429125`.

[86] Ali Sinan Köksal et al. "Synthesizing Signaling Pathways from Temporal Phosphoproteomic Data". In: *bioRxiv* (2017). DOI: 10.1101/209676. eprint: https://www.biorxiv.org/content/early/2017/10/26/209676.full.pdf. URL: https://www.biorxiv.org/content/early/2017/10/26/209676.

[87] Smita Krishnaswamy et al. "Conditional density-based analysis of T cell signaling in single-cell data". In: *Science* 346.6213 (Nov. 2014). PMID: 25342659, p. 1250689. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1250689.

[88] Jan Krumsiek et al. "Hierarchical differentiation of myeloid progenitors is encoded in the transcription factor network". In: *PloS one* 6.8 (2011), e22649.

[89] J. G. Lees et al. "Systematic computational prediction of protein interaction networks". In: *Physical Biology* 8.3 (June 2011), p. 035008. ISSN: 1478-3975. DOI: 10.1088/1478-3975/8/3/035008.

[90] S. Li, S. M. Assmann, and R. Albert. "Predicting essential components of signal transduction networks: a dynamic model of guard cell abscisic acid signaling". In: *PLoS Biol.* 4.10 (Oct. 2006), e312.

[91] Chee Yee Lim et al. "BTR: training asynchronous Boolean models using single-cell expression data". In: *BMC bioinformatics* 17.1 (2016), p. 355.

[92] William JR Longabaugh. "BioTapestry: a tool to visualize the dynamic properties of gene regulatory networks". In: *Gene Regulatory Networks*. http://www.biotapestry.org/. Springer, 2012, pp. 359–394.

[93] Matthew E. MacGilvray et al. "Network inference reveals novel connections in pathways regulating growth and defense in the yeast salt response". In: *bioRxiv* (Aug. 2017), p. 176230. DOI: 10.1101/176230.

[94] Zohar Manna and Richard Waldinger. "A Deductive Approach to Program Synthesis". In: *ACM Transactions on Programming Languages and Systems* 2.1 (Jan. 1980), pp. 90–121. ISSN: 0164-0925. DOI: 10.1145/357084.357090.

[95] Florian Markowetz et al. "Nested Effects Models for High-Dimensional Phenotyping Screens". In: *Bioinformatics* 23.13 (July 2007), pp. i305–i312. ISSN: 1367-4803, 1460-2059. DOI: 10.1093/bioinformatics/btm178.

[96] M. Masnadi-Shirazi, M.R. Maurya, and S. Subramaniam. "Time-Varying Causal Inference From Phosphoproteomic Measurements in Macrophage Cells". In: *IEEE Transactions on Biomedical Circuits and Systems* 8.1 (Feb. 2014), pp. 74–86. ISSN: 1932-4545. DOI: 10.1109/TBCAS.2013.2288035.

[97] H. H. McAdams and A. Arkin. "Stochastic mechanisms in gene expression". In: *Proc. Natl. Acad. Sci. U.S.A.* 94.3 (Feb. 1997), pp. 814–819.

[98] Victoria Moignard et al. "Decoding the regulatory network of early blood development from single-cell gene expression measurements". In: *Nature Biotechnology* 33.3 (Mar. 2015), pp. 269–276. ISSN: 1087-0156. DOI: 10.1038/nbt.3154.

[99] Evan J. Molinelli et al. "Perturbation Biology: Inferring Signaling Networks in Cellular Systems". In: *PLoS Comput Biol* 9.12 (Dec. 2013), e1003290. DOI: 10.1371/journal.pcbi.1003290.

[100] Melody K. Morris et al. "Training Signaling Pathway Maps to Biochemical Data with Constrained Fuzzy Logic: Quantitative Analysis of Liver Cell Responses to Inflammatory Stimuli". In: *PLoS Computational Biology* 7.3 (Mar. 2011), e1001099. DOI: 10.1371/journal.pcbi.1001099.

[101] Roberto Mosca et al. "Towards a detailed atlas of protein–protein interactions". In: *Current Opinion in Structural Biology*. Catalysis and regulation / Protein-protein interactions 23.6 (Dec. 2013), pp. 929–940. ISSN: 0959-440X. DOI: 10.1016/j.sbi.2013.07.005.

[102] Leonardo de Moura and Nikolaj Bjørner. "Z3: Efficient SMT Solver". In: *TACAS'08: Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 4963/2008. Lecture Notes in Computer Science. 2008, pp. 337–340.

[103] Susanne Muehlich et al. "Serum-Induced Phosphorylation of the Serum Response Factor Coactivator MKL1 by the Extracellular Signal-Regulated Kinase 1/2 Pathway Inhibits Its Nuclear Localization". In: *Molecular and Cellular Biology* 28.20 (Oct. 2008). PMID: 18694962, pp. 6302–6313. ISSN: 0270-7306, 1098-5549. DOI: 10.1128/MCB.00427-08.

[104] Christoph Müssel, Martin Hopfensitz, and Hans A Kestler. "BoolNet—an R package for generation, reconstruction and analysis of Boolean networks". In: *Bioinformatics* 26.10 (2010). http://sysbio.uni-ulm.de/?Software:BoolNet, pp. 1378–1380.

[105] Robert H. Newman, Jin Zhang, and Heng Zhu. "Toward a systems-level view of dynamic phosphorylation networks". In: *Frontiers in Genetics* 5 (Aug. 2014), p. 263. DOI: 10.3389/fgene.2014.00263.

[106] Alexander Nikitin et al. "Pathway studio—the analysis and navigation of molecular networks". In: *Bioinformatics* 19.16 (2003), pp. 2155–2157.

[107] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: a comprehensive step-by-step guide*. Artima Press, 2008.

[108] Oved Ourfali et al. "SPINE: a framework for signaling-regulatory pathway inference from cause-effect experiments". In: *Bioinformatics* 23.13 (July 2007), pp. i359–i366. DOI: 10.1093/bioinformatics/btm170.

[109] Nicola Paoletti et al. "Analyzing and Synthesizing Genomic Logic Functions". In: *Computer Aided Verification (CAV)*. Springer. 2014, pp. 343–357.

[110] In-Hyun Park et al. “Disease-specific induced pluripotent stem cells”. In: *cell* 134.5 (2008), pp. 877–886.

[111] Yongjin Park and Joel S. Bader. “How networks change with time”. In: *Bioinformatics* 28.12 (June 2012). PMID: 22689777, pp. i40–i48. ISSN: 1367-4803, 1460-2059. DOI: `10.1093/bioinformatics/bts211`.

[112] Ashwini Patil and Kenta Nakai. “TimeXNet: Identifying active gene sub-networks using time-course gene expression profiles”. In: *BMC Systems Biology* 8.Suppl 4 (Dec. 2014). PMID: 25522063, S2. ISSN: 1752-0509. DOI: `10.1186/1752-0509-8-S4-S2`.

[113] Ashwini Patil et al. “Linking Transcriptional Changes over Time in Stimulated Dendritic Cells to Identify Gene Networks Activated during the Innate Immune Response”. In: *PLoS Comput Biol* 9.11 (Nov. 2013), e1003323. DOI: `10.1371/journal.pcbi.1003323`.

[114] C. P. Paweletz et al. “Reverse phase protein microarrays which capture disease progression show activation of pro-survival pathways at the cancer invasion front”. In: *Oncogene* 20.16 (Apr. 2001). PMID: 11360182, pp. 1981–1989. ISSN: 0950-9232. DOI: `10.1038/sj.onc.1204265`.

[115] T. Pawson and N. Warner. “Oncogenic re-wiring of cellular signaling pathways”. In: *Oncogene* 26.9 (2007), pp. 1268–1275. ISSN: 0950-9232. DOI: `10.1038/sj.onc.1210255`.

[116] Isabelle S Peter, Emmanuel Faure, and Eric H Davidson. “Predictive computation of genomic logic processing functions in embryonic development”. In: *Proceedings of the National Academy of Sciences* 109.41 (2012), pp. 16434–16442.

[117] Teresa M Przytycka, Mona Singh, and Donna K Slonim. “Toward the dynamic interactome: it’s about time”. In: *Briefings in bioinformatics* (2010), bbp057.

[118] Teresa M. Przytycka, Mona Singh, and Donna K. Slonim. “Toward the dynamic interactome: it’s about time”. In: *Briefings in Bioinformatics* 11.1 (Jan. 2010). PMID: 20061351, pp. 15–29. ISSN: 1467-5463, 1477-4054. DOI: `10.1093/bib/bbp057`.

[119] Raven J. Reddy et al. “Early signaling dynamics of the epidermal growth factor receptor”. In: *Proceedings of the National Academy of Sciences* 113.11 (Mar. 2016). PMID: 26929352, pp. 3114–3119. ISSN: 0027-8424, 1091-6490. DOI: `10.1073/pnas.1521288113`.

[120] Aviv Regev and Ehud Shapiro. “The pi-calculus as an abstraction for biomolecular systems”. In: (2004).

[121] Thomas W. Reps et al. “There’s Plenty of Room at the Bottom: Analyzing and Verifying Machine Code”. In: *Computer Aided Verification (CAV)*. 2010, pp. 41–56.

[122] Anna Ritz et al. “Pathways on demand: automated reconstruction of human signaling networks”. In: *npj Systems Biology and Applications* 2 (Mar. 2016), p. 16002. ISSN: 2056-7189. DOI: `10.1038/npjsba.2016.2`.

[123] Aurélien Rizk et al. "Continuous valuations of temporal logic specifications with applications to parameter optimization and robustness measures". In: *Theor. Comput. Sci.* 412.26 (2011), pp. 2827–2839.

[124] Natalie Romanov et al. "Identifying protein kinase–specific effectors of the osmostress response in yeast". In: *Science Signaling* 10.469 (Mar. 2017). PMID: 28270554, eaag2435. ISSN: 1945-0877, 1937-9145. DOI: `10.1126/scisignal.aag2435`.

[125] Jean-Francois Rual et al. "Towards a proteome-scale map of the human protein–protein interaction network". In: *Nature* 437.7062 (2005), pp. 1173–1178.

[126] Jan Daniel Rudolph et al. "Elucidation of Signaling Pathways from Large-Scale Phosphoproteomic Data Using Protein Interaction Networks". In: *Cell Systems* 3.6 (Dec. 2016). PMID: 28009266, 585–593.e3. ISSN: 2405-4712. DOI: `10.1016/j.cels.2016.11.005`.

[127] Daniel P Ryan and Jacqueline M Matthews. "Protein–protein interactions in human disease". In: *Current opinion in structural biology* 15.4 (2005), pp. 441–446.

[128] Wouter Saelens et al. "A comparison of single-cell trajectory inference methods: towards more accurate and robust tools". In: *bioRxiv* (2018), p. 276907.

[129] Julio Saez-Rodriguez et al. "A logical model provides insights into T cell receptor signaling". In: *PLoS Computational Biology* 3.8 (2007), e163.

[130] Julio Saez-Rodriguez et al. "Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction". In: *Molecular systems biology* 5.1 (2009).

[131] Marc A Schaub, Thomas A Henzinger, and Jasmin Fisher. "Qualitative networks: a symbolic approach to analyze biological signaling networks". In: *BMC systems biology* 1.1 (2007), p. 4.

[132] Paul Shannon et al. "Cytoscape: a software environment for integrated models of biomolecular interaction networks". In: *Genome research* 13.11 (2003), pp. 2498–2504.

[133] Roded Sharan and Richard M. Karp. "Reconstructing Boolean Models of Signaling". In: *Journal of Computational Biology* 20.3 (Jan. 2013), pp. 249–257. ISSN: 1066-5277. DOI: `10.1089/cmb.2012.0241`.

[134] Roded Sharan and Richard M Karp. "Reconstructing Boolean models of signaling". In: *Journal of Computational Biology* 20.3 (2013), pp. 249–257.

[135] Kirti Sharma et al. "Ultradeep Human Phosphoproteome Reveals a Distinct Regulatory Nature of Tyr and Ser/Thr-Based Signaling". In: *Cell Reports* 8.5 (Sept. 2014), pp. 1583–1594. ISSN: 2211-1247. DOI: `10.1016/j.celrep.2014.07.036`.

[136] Ilya Shmulevich et al. "Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks". In: *Bioinformatics* 18.2 (2002), pp. 261–274.

[137] Dana Silverbush and Roded Sharan. “Network orientation via shortest paths”. In: *Bioinformatics* 30.10 (May 2014). PMID: 24470573, pp. 1449–1455. ISSN: 1367-4803, 1460-2059. DOI: `10.1093/bioinformatics/btu043`.

[138] Armando Solar-Lezama, Christopher Grant Jones, and Rastislav Bodik. “Sketching concurrent data structures”. In: *Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation*. PLDI ’08. Tucson, AZ, USA: ACM, pp. 136–148.

[139] Armando Solar-Lezama et al. “Combinatorial Sketching for Finite Programs”. In: *ASPLOS*. Oct. 2006.

[140] Armando Solar-Lezama et al. “Combinatorial sketching for finite programs”. In: *ASPLOS-XII*. San Jose, California, USA: ACM, 2006, pp. 404–415. ISBN: 1-59593-451-0.

[141] Armando Solar-Lezama et al. “Programming by Sketching for Bit-streaming Programs”. In: *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI ’05. ACM, 2005, pp. 281–294. ISBN: 978-1-59593-056-9. DOI: `10.1145/1065010.1065045`. URL: `http://doi.acm.org/10.1145/1065010.1065045`.

[142] Laura A Solt et al. “ROR Inverse Agonist Suppresses Insulitis and Prevents Hyperglycemia in a Mouse Model of Type 1 Diabetes”. In: *Endocrinology* (2015).

[143] Saurabh Srivastava, Sumit Gulwani, and Jeffrey S. Foster. “From Program Verification to Program Synthesis”. In: *POPL*. 2010.

[144] Matthias Stadtfeld and Konrad Hochedlinger. “Induced pluripotency: history, mechanisms, and applications”. In: *Genes & development* 24.20 (2010), pp. 2239–2263.

[145] Kelly Street et al. “Slingshot: Cell lineage and pseudotime inference for single-cell transcriptomics”. In: *bioRxiv* (2017), p. 128843.

[146] Meera V Sundaram. “The love–hate relationship between Ras and Notch”. In: *Genes & development* 19.16 (2005), pp. 1825–1839.

[147] Camille D. A. Terfve et al. “Large-scale models of signal propagation in human cells derived from discovery phosphoproteomic data”. In: *Nature Communications* 6.8033 (Sept. 2015). DOI: `10.1038/ncomms9033`. URL: `http://www.nature.com/ncomms/2015/150910/ncomms9033/full/ncomms9033.html`.

[148] Camille Terfve et al. “CellNOptR: a flexible toolkit to train protein signaling networks to data using multiple logic formalisms”. In: *BMC systems biology* 6.1 (2012). `http://www.cellnopt.org/`, p. 133.

[149] Aditya V. Thakur et al. “Directed Proof Generation for Machine Code”. In: *Computer Aided Verification (CAV)*. 2010, pp. 288–305.

[150] Cole Trapnell et al. "The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells". In: *Nature biotechnology* 32.4 (2014), pp. 381–386.

[151] Nurcan Tuncbag et al. "Simultaneous Reconstruction of Multiple Signaling Pathways via the Prize-Collecting Steiner Forest Problem". In: *Journal of Computational Biology* 20.2 (Feb. 2013), pp. 124–136. ISSN: 1066-5277, 1557-8666. DOI: 10.1089/cmb.2012.0092.

[152] Moignard V. et al. "Decoding the Transcriptional Program for Blood Development from Whole Tissue Single Cell Gene Expression Measurements". In: *Nature Biotechnology, in press* (Jan. 2015). URL: http://research.microsoft.com/apps/pubs/default.aspx?id=217707.

[153] Martin Vechev and Eran Yahav. "Deriving linearizable fine-grained concurrent objects". In: *SIGPLAN Not.* 43.6 (June 2008), pp. 125–135. ISSN: 0362-1340. DOI: 10.1145/1379022.1375598. URL: http://doi.acm.org/10.1145/1379022.1375598.

[154] Arunachalam Vinayagam et al. "A Directed Protein Interaction Network for Investigating Intracellular Signal Transduction". In: *Science Signaling* 4.189 (Sept. 2011), rs8. DOI: 10.1126/scisignal.2001699.

[155] Rui-Sheng Wang, Assieh Saadatpour, and Réka Albert. "Boolean modeling in systems biology: an overview of methodology and applications". In: *Physical biology* 9.5 (2012), p. 055001.

[156] Xin Wang et al. "Reconstructing evolving signalling networks by hidden Markov nested effects models". In: *The Annals of Applied Statistics* 8.1 (Mar. 2014). Zbl: 06302243, pp. 448–480. ISSN: 1932-6157, 1941-7330. DOI: 10.1214/13-AOAS696.

[157] Caleb Weinreb et al. "Fundamental limits on dynamic inference from single-cell snapshots". In: *Proceedings of the National Academy of Sciences* (2018), p. 201714723.

[158] Andrew J Wilson. "Inhibition of protein–protein interactions using designed molecules". In: *Chemical Society Reviews* 38.12 (2009), pp. 3289–3300.

[159] Steven Woodhouse et al. "Synthesising Executable Gene Regulatory Networks from Single-Cell Gene Expression Data". In: *Computer Aided Verification*. 2015.

[160] Sean M Wu and Konrad Hochedlinger. "Harnessing the potential of induced pluripotent stem cells for regenerative medicine". In: *Nature cell biology* 13.5 (2011), pp. 497–505.

[161] Ioannis Xenarios et al. "DIP, the Database of Interacting Proteins: a research tool for studying cellular networks of protein interactions". In: *Nucleic acids research* 30.1 (2002), pp. 303–305.

[162] Brian S. Yandell. *Practical Data Analysis for Designed Experiments*. Chapman & Hall/CRC Texts in Statistical Science. Chapman & Hall, Jan. 1997. ISBN: 978-0-412-06341-1.

[163] Chen-Hsiang Yeang, Trey Ideker, and Tommi Jaakkola. “Physical Network Models”. In: *Journal of Computational Biology* 11.2–3 (2004), pp. 243–262. DOI: `10.1089/1066527041410382`.

[164] Esti Yeger-Lotem et al. “Bridging high-throughput genetic and transcriptional data reveals cellular responses to alpha-synuclein toxicity”. In: *Nature Genetics* 41.3 (Mar. 2009), pp. 316–323. ISSN: 1061-4036. DOI: `10.1038/ng.337`.

[165] A. S. Yoo, C. Bais, and I. Greenwald. “Crosstalk between the EGFR and LIN-12/Notch pathways in C. elegans vulval development”. In: *Science* 303.5658 (Jan. 2004), pp. 663–666.

[166] Boyan Yordanov et al. “A method to identify and analyze biological programs through automated reasoning”. In: *NPJ systems biology and applications* 2 (2016), p. 16010.

[167] Boyan Yordanov et al. “SMT-Based Analysis of Biological Computation.” In: *NASA Formal Methods* 7871 (2013), pp. 78–92.

[168] Boyan Yordanov et al. “Z34Bio: An SMT-based Framework for Analyzing Biological Computation”. In: *SMT Workshop 2013 11th International Workshop on Satisfiability Modulo Theories*. `http://research.microsoft.com/en-us/projects/z3-4biology/`. 2013.

[169] Nir Yosef et al. “Dynamic regulatory network controlling T H 17 cell differentiation”. In: *Nature* 496.7446 (2013), p. 461.

[170] Nir Yosef et al. “Toward accurate reconstruction of functional protein networks”. In: *Molecular Systems Biology* 5.248 (Mar. 2009). DOI: `10.1038/msb.2009.3`. URL: `http://dx.doi.org/10.1038/msb.2009.3`.

[171] Chiou-Hwa Yuh, Hamid Bolouri, and Eric H Davidson. “Cis-regulatory logic in the endo16 gene: switching from a specification to a differentiation mode of control”. In: *Development* 128.5 (2001), pp. 617–629.

[172] Hongtao Zhang et al. “ErbB receptors: from oncogenes to targeted cancer therapies”. In: *The Journal of clinical investigation* 117.8 (2007), pp. 2051–2058.

[173] Ranran Zhang et al. “Network model of survival signaling in large granular lymphocyte leukemia”. In: *Proceedings of the National Academy of Sciences* 105.42 (2008), pp. 16308–16313.

[174] Feng Zhou et al. “Genome-scale proteome quantification by DEEP SEQ mass spectrometry”. In: *Nature Communications* 4 (July 2013), p. 2171. DOI: `10.1038/ncomms3171`.

[175] Qiao Zhou et al. “In vivo reprogramming of adult pancreatic exocrine cells to &bgr;-cells”. In: *nature* 455.7213 (2008), pp. 627–632.

[176] Pietro Zoppoli, Sandro Morganella, and Michele Ceccarelli. “TimeDelay-ARACNE: Reverse engineering of gene networks from time-course data by an information theoretic approach”. In: *BMC Bioinformatics* 11 (Mar. 2010), p. 154. ISSN: 1471-2105. DOI: `10.1186/1471-2105-11-154`.

[177] Blaz Zupan et al. “GenePath: a system for automated construction of genetic networks from mutant data”. In: *Bioinformatics* 19.3 (2003). `http://genepath.org/`, pp. 383–389.