Analyzing and Synthesizing Genomic Logic Functions

Nicola Paoletti[1,2], Boyan Yordanov[1], Youssef Hamadi[1], Christoph M. Wintersteiger[1], and Hillel Kugler[1]

[1] Microsoft Research, Cambridge, UK
{yordanov,cwinter,youssefh,hkugler}@microsoft.com
[2] Department of Computer Science, University of Oxford, UK
nicola.paoletti@cs.ox.ac.uk
http://research.microsoft.com/z3-4biology

Abstract. Deciphering the developmental program of an embryo is a fundamental question in biology. Landmark papers [10, 11] have recently shown how computational models of gene regulatory networks provide system-level causal understanding of the developmental processes of the sea urchin, and enable powerful predictive capabilities. A crucial aspect of the work is empirically deriving plausible models that explain all the known experimental data, a task that becomes infeasible in practice due to the inherent complexity of the biological systems. We present a generic Satisfiability Modulo Theories based approach to analyse and synthesize data constrained models. We apply our approach to the sea urchin embryo, and successfully improve the state-of-the-art by synthesizing, for the first time, models that explain all the experimental observations in [11]. A strength of the proposed approach is the combination of accurate synthesis procedures for deriving biologically plausible models with the ability to prove inconsistency results, showing that for a given set of experiments and possible class of models no solution exists, and thus enabling practical refutation of biological models.

1 Introduction

Understanding the underlying developmental program of an embryo is a fascinating scientific question. How do cells divide and organize to form a body plan, each one becoming a specific cell type capable of performing specialized functions and interacting with other nearby cells to form a living organism? Answering these questions, apart from their significant scientific value, has far reaching applications, for instance in early diagnosis, disease treatment, and regenerative medicine.

At the heart of understanding the complexity of development is the challenge of understanding the process of biological computation that is performed within living cells and organisms. The information processing is implemented via highly concurrent biological machinery determining when to express specific genes, which in an abstract view is seen as turning these genes on or off. Gene regulatory networks (GRNs) control the dynamic (and spatial) patterns of gene expression, which influence the decisions cells make during development. Unraveling the structure and logic of these GRNs is thus a key research challenge.

Landmark papers studying the development of the purple sea urchin [10, 11] have recently shown how computational models of gene regulatory networks provide system-level causal understanding of the embryonic developmental processes, and enable powerful predictive capabilities. The methodology used in that work is based on modeling and simulation of Boolean systems with time delays and discrete time semantics using a form of vector-based equations that determine the dynamics of gene expression. A crucial aspect of this work is deriving biologically plausible models that explain all the known experimental data, a task that becomes infeasible when using simulation based methods, due to the inherent complexity of the biological systems.

We present an SMT-based approach that enables the analysis of realistic GRNs and synthesizes models that accurately explain all known data. Our method significantly extends the framework presented in [14] in order to incorporate time delays, spatial domains and the systematic use of uninterpreted functions. We take a pragmatic approach, and rather than introducing a new language for biological modeling we formalize the vector equation notation introduced by Peter et al. [10] which has been demonstrated to be useful for experimental biologists, shedding light on some of its constructs and features that were previously described informally.

The GRN reconstruction described in [10] is a result of over thirty years of research, and incorporates detailed experimental data from various sources and techniques. The scientific essence of a model is its ability to explain *all* existing data and make new testable predictions. The detailed data of the expression of many relevant genes at different time points and different spatial domains in the embryo make the construction of such a model very hard. We explain how we were able to construct the first model that fully explains all of the data from [10], including perturbation experiments, through the use of formal analysis and synthesis methods. Furthermore, we demonstrate that a subset of vector equations is inconsistent with experimental data, regardless of how the other vector equations are set, which is essential information for experimentalists and which is impossible to obtain through the use of simulation techniques only. The scalability of our methods as shown by analyzing the sea urchin model paves the way for practical usage of formal methods, potentially transforming the way in which computational modeling and experiments enhance our understanding of biology.

2 Background

To establish a formal basis for our analysis, we first define some basic concepts and a succinct notion of Gene Regulatory Networks with spatial and temporal domains, as well as observations and perturbations.

At the most fundamental level we require the set of Boolean values $\mathbb{B} = \{0, 1\}$ and the usual Boolean operations. A *bit-vector* $b \in \mathbb{B}^n$ is a vector of Boolean variables $b_0, b_1, \ldots, b_{n-1}$, where $b_i \in \mathbb{B}$ for each $i = 0, \ldots, n-1$. We assume the usual bit-wise operations on bit-vectors, including the arithmetic operators $+, -, *, /$. We write $b_i = b_j$ to indicate the logical equivalence $b_i \iff b_j$.

For most of the problems we investigate in this paper, it is convenient to have one compact mathematical object which carries all the information available from the biological context over the formal context. For this reason we define the notion of Gene Regulatory networks with Delays, and spatial Domains:

Definition 1 (GRNDD). *A* Gene Regulatory Network with Delays and spatial Domains (GRNDD) *is a tuple* $(G, D, SR, \mathbb{T}, F)$*, where:*

- G *is a finite set of* genes*;*
- D *is a finite set of* spatial domains*;*
- SR *is a finite set of* spatial relations *between domains;*
- $\mathbb{T} = \{0, 1, \ldots, t_{max}\}$ *is the discrete* time domain*; and*
- F *is a finite set of* vector equations*.*

A GRNDD captures the required information describing the system dynamics which can be represented as a transition system. The components of the transition system are the genes G, the spatial domains D and the vector equations (which implicitly define the transition relation). SR is a set of relations that describes the relationship between spatial domains (e.g., whether they are next to each other, or close to each other, etc). Each element of SR is a function $r : \mathbb{T} \times D \times D \to \mathbb{B}$, and $r(t, d_i, d_j) = 1$ iff the relation r holds between spatial domains d_i and d_j at time t.

Note that our definition of a GRNDD contains a discrete and *bounded* time domain, where t_{max} is the *maximum execution time*. The problems we investigate in the remainder of this paper are always related to a set of observational data, which represent finite executions of the system, often providing experimental measurements for each time step from steps 0 to t_{max}.

A state q of a GRNDD is a valuation of the expression of each gene in each spatial domain, i.e., essentially a bit-vector that describes whether each of the genes is enabled or not in a spatial domain. Thus, we define the set of states $Q := \mathbb{B}^{|G \times D|}$. The expression (valuation) of gene g in domain d in a state q is denoted by $q(g, d)$. A path is a sequence of states and we denote the set of paths as $\Pi := \{< q_0, \ldots, q_i > \mid 0 \leq i \leq t_{max} \wedge q_i \in Q\}$ and $\pi[i]$ denotes the i-th state in path π.

The set of vector equations F of a GRNDD contains an update function for every gene $g \in G$ which has the signature $f_g : \Pi \times \mathbb{T} \times D \to \mathbb{B}$. In other words, the vector equation of a gene g is a function $f_g(\pi, t, d)$ which determines whether g is expressed at a time t in spatial domain d in a path π. Note that these update functions depend on a whole path through the system, because they may depend not only on a unique previous state, but on a sequence of previously visited states. As pointed out by Peter et al. [11], the term "vector equation" reflects the matrices of gene expression in space and time that these equations generate. We provide a syntax and semantics of these functions in Sec. 2.1.

To incorporate experimental data into our analysis, we define a set of observations as follows:

Definition 2 (Observations). *Observations are sets of tuples (C, E), where*

- *C is a set of perturbed vector equations; and*
- *E is a set of predicates $e : \Pi \times \Pi \to \mathbb{B}$ describing* effects.

The most basic observation is that of the so-called *wild type*, which means that a system is observed without making changes to the system; such an observation has the form $(\emptyset, \pi)$ where π is a predicate that describes a concrete path (of finite length). Observations with non-empty C are called *perturbation* (or *mutation*) experiments. These describe the system behavior after some change is made to the system. A common experiment is that of a *knock-out* where some gene is repressed, e.g., $C = \{f_g(\pi, t, d) := 0\}$ for some gene g. In Section 3.1, we illustrate how the predicates in E can be defined for expressing perturbation effects.

2.1 Formal syntax and semantics of vector equations

A formal syntax for vector equations is not strictly required. In practice however, the establishment of a language greatly simplifies the modeling task. Peter et al. [11] propose such a language, but they do not provide a formal syntax or semantics. To establish a fully formal basis, we revisit their operators and prescribe a formal semantics to them.[3]

Definition 3 (Vector equation language). *The syntax of vector equations is given by the following grammar:*

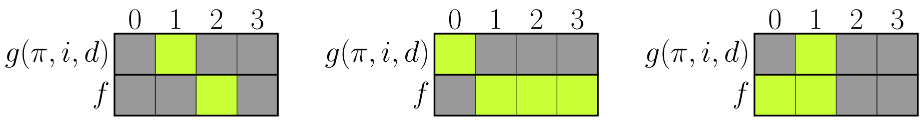
$$\begin{aligned} V ::= g \mid\ > t \mid\ < t \mid \textsc{In}\ \bar{d} \mid \neg V \mid V \wedge V \mid V \vee V \\ \mid \textsc{At-}n\ V \mid \textsc{After-}n\ V \mid \textsc{Perm-}n\ V \mid \textsc{In}\ \bar{d}\ V \mid \textsc{In}\ r\ \bar{d}\ V \end{aligned}$$

where $g \in G$, $t \in \mathbb{T}$, $n \in \mathbb{N}$, $r \in SR$ and $\bar{d} \in D$.

The semantics of a term V within the vector equation language is defined with respect to a path π, a time point i, and a spatial domain d using the Boolean connectors $\neg$, $\wedge$ and $\vee$. The semantics of temporal operators are defined as:

$$\begin{aligned} g(\pi, i, d) &\iff \pi[i](g, d) && \text{(gene expression)} \\ > t\ (\pi, i, d) &\iff i > t && \text{(time boundary)} \\ < t\ (\pi, i, d) &\iff i < t && \text{(time boundary)} \\ \textsc{At-}n\ V(\pi, i, d) &\iff n \leq i \wedge V(\pi, i - n, d) && \text{(delayed temporary effect)} \end{aligned}$$

[3] It is interesting to note that our formalization turns out to be within a subset of the past fragment of LTL (see supplementary material).



(a) $f = \textsc{At}\text{-}1\ g(\pi, i, d)$ (b) $f = \textsc{After}\text{-}1\ g(\pi, i, d)$ (c) $f = \textsc{Perm}\text{-}1\ g(\pi, i, d)$

Fig. 1. Examples showing the semantics of temporal operators. Yellow squares correspond to the points where the gene is on, gray squares where the gene is off.

$$\textsc{After}\text{-}n\ V(\pi, i, d) \iff$$
$$\exists k.\ 0 \leq k \leq i - n \wedge V(\pi, k, d) \qquad \text{(permanent activation)}$$
$$\textsc{Perm}\text{-}n\ V(\pi, i, d) \iff$$
$$\neg(\exists k.\ 1 \leq k \leq i - n\ \wedge\ V(\pi, k, d) \wedge \neg V(\pi, k-1, d)) \ \text{(permanent repression)}$$

The $\textsc{At}$-n $V(\pi, i, d)$ operator corresponds to the evaluation of V at n steps in the past. $\textsc{After}$-n $V(\pi, i, d)$ evaluates to true and stays true thereafter, if V is true n steps earlier, while $\textsc{Perm}$-n $V(\pi, i, d)$ evaluates to false, and stays false thereafter, if V becomes true n steps earlier. Note that $\textsc{Perm}$-n does not simply represent the negation of $\textsc{After}$-n operator, but instead implies that V has to be false $n-1$ and *become* true n steps in the past. An illustration of the path-based semantics of the $\textsc{At}$-, $\textsc{After}$- and $\textsc{Perm}$- operators is provided in Fig. 1.

The semantics of spatial operators are defined as

$$\textsc{In}\ \bar{d}\ (\pi, i, d) \iff d = \bar{d} \qquad \text{(evaluation at domain } \bar{d})$$
$$\textsc{In}\ \bar{d}\ V(\pi, i, \_) \iff V(\pi, i, \bar{d}) \qquad \text{(evaluation of } V \text{ at domain } \bar{d})$$
$$\textsc{In}\ r\ \bar{d}\ V(\pi, i, \_) \iff \exists d\ .\ r(i, \bar{d}, d) \wedge V(\pi, i, d) \ \text{(eval. of } V \text{ in a related domain)}$$

While $\textsc{In}\ \bar{d}$ and $\textsc{In}\ \bar{d}\ E(\pi, i, \_)$ evaluate whether an atom or formula E holds at a given domain, $\textsc{In}\ r\ \bar{d}\ E(\pi, i, \_)$ evaluates to 1 when E holds in some domain d related to $\bar{d}$ via the spatial relation r at time t.

3 Gene expression computation as a path synthesis problem

We adapt the method presented in [14] for the encoding of Boolean networks as finite transition systems (over bit-vectors), in order to support spatial domains and time delays.

Definition 4 (Dynamics of a GRNDD). *The* dynamics of a GRNDD $N = (G, D, SR, \mathbb{T}, F)$ *is formally a finite-state automaton with*

- *set of states* $Q = \mathbb{B}^{|G \times D|}$*;*

– *initial state* $q_0 \in Q$*; and*
– *transition relation* $\delta : \Pi \to \mathbb{B} =$

$$\delta(\pi) \iff \bigwedge_{0<i\leq\mathbb{T},g\in G,d\in D} \pi[i](g,d) = f_g(\pi,i,d)$$

Intuitively, given a path $\pi \in \Pi$, $\delta(\pi)$ holds if π is a valid execution of the system. Note that we do not require an input alphabet. This is because GRNDDs do not have external input. However, non-deterministic behaviour is possible as part of the initial state selection or for update functions with delays beyond the initial time (e.g. $f_g(\pi,i,d) = \text{AT-3}\ V(\pi,i,d)$, at $i = 2$) or referring to the current time (e.g. $f_g(\pi,i,d) = \text{AT-0}\ V(\pi,i,d)$). Such non-determinism can be limited by the requirement that the system's dynamics are consistent with certain observations.

Note that our model of dynamics is based on that of a concurrent but *synchronous* execution model, meaning that the expressions of all genes of the network are updated within one step of the execution. In principle, it is possible to integrate asynchronous dynamics, even if for this work we have followed the semantics proposed in [11].

We are now ready to state the computation of the temporal and spatial gene expression in terms of a path synthesis problem:

Problem 1 (Gene expression computation) *Let* $N = (G,D,SR,\mathbb{T},F)$ *be a GRNDD. The computation of the temporal and spatial gene expression of network* N *corresponds to the synthesis of a (set of) path(s)* $\{\pi_i\} \subseteq \Pi$ *of length* $|\mathbb{T}|$ *such that for each* π_i, $\delta(\pi_i)$ *holds.*

In our experiments, we encode this problem modulo the theory of bit-vector formulas (SMT QF_UFBV). If our constraints are satisfiable, i.e. there exist paths π_i such that $\delta(\pi_i)$ holds, then the SMT solver is able to find (synthesize) them one by one.

The dynamics of GRNDD that is perturbed are defined in the straightforward way:

Definition 5 (Perturbed Dynamics). *Let* $N = (G,D,SR,\mathbb{T},F)$ *be a GRNDD, let* $o = (C,E)$ *be an observation. Set* F' *to be* F *with all functions that have a definition in* C *replaced with this definition. Then the dynamics of the perturbed system are the dynamics of* $(G,D,SR,\mathbb{T},F')$.

Finally, we need to be able to check whether a given GRNDD indeed replicates the behavior seen in a set of observations. Formally, we do this by computing gene expressions under all perturbations and checking whether the effects that were observed experimentally are also observed in the GRNDD:

Problem 2 (Adequacy) *Let* $N = (G,D,SR,\mathbb{T},F)$ *be a GRNDD and let* O *be a set of observations. Let* Π_N *be the computed gene expressions for* N*. Determine whether for each observation* $(C_i,E_i) \in O$ *there is a path* π_i *of length* $|\mathbb{T}|$ *which is contained in* Π_N *and a path* π_i' *of the same length in* N *perturbed by* C_i*, such that* $E_i(\pi_i,\pi_i')$ *holds.*

If a GRNDD N is adequate, i.e., if Problem 2 is answered in the positive, then N does indeed allow executions that perfectly explain all observational data. In the next section we make use of this problem definition to synthesize adequate GRNDDs.

3.1 Comparison operators.

In the biological literature, the effects of perturbation experiments are often only formulated in a qualitative fashion (e.g. 'if gene g_0 is knocked out, then g_1 is over-expressed'), rather than in actual execution traces. For the purposes of formal analysis however, a formal semantics of the effects of perturbations is required. To do so, we characterize the class of comparison operators that can occur in the predicates E_i in observations. We consider predicates of the form

$$(\pi_i, g_i, d_i, [t_i^s, t_i^e]) \bowtie (\pi_j, g_j, d_j, [t_j^s, t_j^e])$$

in order to compare the expression of a gene g_i, in a domain d_i, in a path π_i and in a discrete time interval $[t_i^s, \ldots, t_i^e]$, with the expression of a gene g_j, in a domain d_j, in a path π_j and in a time interval $[t_j^s, \ldots, t_j^e]$. We consider two types of operators: First, the *weak* operators are $\bowtie \in \{>, <, \leq, \geq, =\}$ and they are used to compare the "average" expression of the operands in the considered time intervals. We define

$$(\pi_i, g_i, d_i, [t_i^s, t_i^e]) \bowtie (\pi_j, g_j, d_j, [t_j^s, t_j^e]) \iff \frac{\sum_{t=t_i^s}^{t_i^e} \pi_i[t](g_i, d_i)}{t_i^e - t_i^s} \bowtie \frac{\sum_{t=t_j^s}^{t_j^e} \pi_j[t](g_j, d_j)}{t_j^e - t_j^s}$$

Second, the *strong* operators are $\bowtie \in \{\gg, \ll, \gg=, =\ll, ==\}$ and they compare gene expressions point-by-point, and not on the basis of their average over a time interval. Hence, they can be defined only if $t_i^e - t_i^s = t_j^e - t_j^s$, i.e. if the time intervals of the two operands have the same length. Let us assume that $t_i^s = t_j^s + k$ and that $t_i^e = t_j^e + k$, with $k \in \mathbb{Z}$. Then,

$$(\pi_i, g_i, d_i, [t_i^s, t_i^e]) \bowtie (\pi_j, g_j, d_j, [t_j^s, t_j^e]) \iff \bigwedge_{t=t_i^s}^{t_i^e} \pi_i[t](g_i, d_i) \bowtie_w \pi_j[t+k](g_j, d_j)$$

where $\bowtie_w$ denotes the weak version of a strong operator $\bowtie$. Fig. 2 shows two examples of how these operators apply.

4 Synthesis of vector equations

In this section we present a procedure for the automated synthesis of vector equations, so that the computed temporal expressions meet the observations,

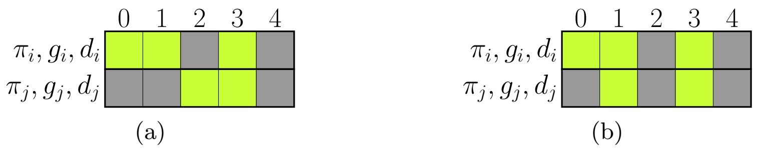


Fig. 2. Comparison of two temporal expression patterns π_i and π_j. Yellow squares indicate time-points where the gene is expressed, gray squares where it is not. Gene g_i in π_i and domain d_i is overexpressed w.r.t. gene g_j in π_j and domain d_j. However, the strong comparison operator $\gg=$ does not hold in the example (a) over the time interval $[0, 4]$, while its weak equivalent $(\geq)$ does. Contrarily in example (b), $(\pi_i, g_i, d_i, [0, 4]) \gg= (\pi_j, g_j, d_j, [0, 4])$, because the comparison is verified point-by-point, and thus $(\pi_i, g_i, d_i, [0, 4]) \geq (\pi_j, g_j, d_j, [0, 4])$ also holds.

with the aim of answering positively to Problem 2. We extend the vector equation language to enable *synthesis of basic gene interactions*, i.e. simple regulatory interactions from which more complex gene functions are constructed; and the *synthesis of generic Boolean functions*, based on the use of uninterpreted functions for finding admissible logical combinations of vector equation terms.

Definition 6 (Basic interaction). *Let $N = (G, D, SR, \mathbb{T}, F)$ be a GRNDD. A* basic interaction *(BI) of N is a tuple $f = (g, b, d, r, t, op)$ where:*

- *$g \subseteq G$ is a set of* input genes*;*
- *$b \subseteq \mathbb{B}$ is a set of Boolean values indicating whether genes in g are expressed or not;*
- *$d \subseteq D^\epsilon = D \cup \{\epsilon\}$ is a possibly empty set of ($d = \epsilon$)* spatial domains*;*
- *$r \subseteq SR^\epsilon = SR \cup \{\epsilon\}$ is a possibly empty set of* spatial relations*;*
- *$op \subseteq \{$*AT-, AFTER-, PERM-*$\}$ and $t \in \mathbb{T}$ are a set of* temporal operators *and a corresponding set of* delays*, respectively.*

A BI $f = (g, b, d, r, t, op)$ describes a set of interactions where an input gene $g' \in g$ (whose expression depends on a $b' \in b$) affects the target gene according to a temporal operator $op' \in op$ and delay in $t' \in t$, and possibly occurring in a domain $d' \in d$, or in a domain that is in a spatial relation $r' \in r$ with d'. In order to avoid unwanted redundancies and nondeterminism, we exclude the empty temporal operator which is semantically equivalent to AT-0. The choice of this particular template for BIs was driven by observing that in the sea urchin model by Peter et al. [11], every vector equation takes the form of a logical combination of terms characterized by these same information. On the specification side, we want to allow the modeller to incorporate some degree of flexibility in the declaration of a BI to synthesize, and in turn to include gene regulations that are supported by experimental evidence. Moreover, constraining the set of potential interactions has the added benefit that the resulting functions can be easily interpreted by the domain expert who specified the templates, and that the same templates can be used to exclude unwanted or biologically unlikely interactions. Of course, there is also a performance advantage, because a smaller set of functions is considered by the solver. Formally, the declaration of a basic interaction is of the form

$$f \subseteq G \times \mathbb{B} \times D^{\epsilon} \times SR^{\epsilon} \times \mathbb{T} \times \{\textsc{At-}, \textsc{After-}, \textsc{Perm-}\} \to \mathbb{B}$$

where f is a symbol from the set $\mathcal{I}$ of declared BI symbols. In practice, we constrain every BI by a declaration of choice of (subsets of) admissible values to each of its elements. For example, $f \overset{dec}{=} (\{g\}, \mathbb{B}, \{\epsilon\}, \{\epsilon\}, \{0, 1, 2\}, \{\textsc{After-}, \textsc{Perm-}\})$ describes an interaction with a gene g whose expression is unknown (b can take any Boolean value), where no domain-specific nor inter-domain signaling occurs (both d and r are fixed to ϵ), and causing a permanent effect ($op = \{\textsc{After-}, \textsc{Perm-}\}$) on the target gene with a maximum delay of 2.

Obviously, for every declaration $f \overset{dec}{=} (g, b, d, r, t, op)$, we need to impose the BI f to be evaluated to one of the admissible values specified in its declaration, which imposes the constraint

$$\bigvee_{g' \in g, b' \in b, d' \in d, r' \in r, t' \in t, op' \in op} f = (g', b', d', r', t', op')$$

Apart from basic interactions, we also allow the specification of arbitrary Boolean functions by the use of uninterpreted functions (UF) of the form $uf : \mathbb{B}^n \to \mathbb{B}$. The only constraint we impose on such functions is whether input variables are allowed to be negated or not. This information depends on domain knowledge, e.g., it may be known that expression of some gene g_0 inhibits the expression of another gene g_1, but the precise mechanism of inhibition is unknown.

Indeed the negation of a term radically changes the regulatory input it represents, by turning an activation input into an inhibition (and viceversa). Therefore, if we require that the synthesized function does not contradict known biological hypotheses about the kind of input interaction, for a UF uf of arity n the following constraints are imposed:

$$\bigwedge_{i=1,\ldots,n} uf(b_1, \ldots, b_{i-1}, 0, b_{i+1}, \ldots b_n) \implies uf(b_1, \ldots, b_{i-1}, 1, b_{i+1}, \ldots b_n)$$

We briefly explain the rationale behind this formula. According to the sum-of-products (SoP, DNF) form of uf, if $uf(b_1, \ldots, b_{i-1}, 0, b_{i+1}, \ldots, b_n)$ holds for some choice of $b_1, \ldots, b_n$, then the function translates to a formula of the form

$$uf = (b'_1 \wedge \ldots \wedge b'_{i-1} \wedge \neg b_i \wedge b'_{i+1} \wedge \ldots \wedge b'_n) \vee \ldots$$

which contains a min-term where the i-th variable is negated and b'_j is either b_j or $\neg b_j$ for $j \neq i$. A way to get rid of the negated term $\neg b_i$ is to enforce that $uf(b_1, \ldots, b_{i-1}, 1, b_{i+1}, \ldots, b_n)$ holds for the same choice of variables $b_1, \ldots, b_{i-1}$, $b_{i+1}, \ldots, b_n$. By doing so, in the SoP representation of uf the negated term would conveniently simplify because

$$\begin{aligned}&(b'_1 \wedge \ldots \wedge b'_{i-1} \wedge \neg b_i \wedge b'_{i+1} \wedge \ldots \wedge b'_n) \vee \\ &(b'_1 \wedge \ldots \wedge b'_{i-1} \wedge \quad b_i \wedge b'_{i+1} \wedge \ldots \wedge b'_n) \vee \ldots \\ &= (b'_1 \wedge \ldots \wedge b'_{i-1} \wedge b'_{i+1} \wedge \ldots \wedge b'_n) \vee \ldots\end{aligned}$$

We now extend the vector equation language in order to support the specification of BIs and UFs to be synthesized:

Definition 7 (Vector equation language for synthesis). *The syntax of vector equations supporting the synthesis of basic interactions and Boolean functions is given by the following grammar:*

$$\begin{aligned}V ::= f \mid uf(V, \ldots, V) \mid g \mid \ > t \mid \ < t \mid \textsc{In}\ \bar{d} \mid \neg V \mid V \wedge V \mid V \vee V \\ \mid \textsc{At-}n\ V \mid \textsc{After-}n\ V \mid \textsc{Perm-}n\ V \mid \textsc{In}\ \bar{d}\ V \mid \textsc{In}\ r\ \bar{d}\ V\end{aligned}$$

where $g \in G$, $t \in \mathbb{T}$, $n \in \mathbb{N}$, $r \in SR$, $\bar{d} \in D$, $f \in \mathcal{I}$ is a declared BI symbol*; and $uf \in \mathcal{U}$ is a* declared UF symbol.

The semantics of the expression $uf(V_1, \ldots, V_n)$ simply consists in the application of the UF uf over its arguments:

$$uf(V_1, \ldots, V_n)\ (\pi, i, d) \iff uf(V_1(\pi, i, d), \ldots, V_n(\pi, i, d))\ .$$

The semantics of a BI f is defined as a conjunction of formulas of the form $f = (g', b', d', r', t', op') \implies V(\pi, i, d)$ which relate any synthesizable evaluation of f to the corresponding vector equation term V:

$$\bigwedge_{\substack{g' \in g, b' \in b, \\ d' \in d, r' \in r, \\ t' \in t, op' \in op}} f = (g', b', d', r', t', op') \implies op'\ t'(IN\ r'd'(b' \iff g'))\ (\pi, i, d)$$

where $IN\ r'd'\ V$ corresponds to $INd'\ V$ if $r' = \epsilon \wedge d' \neq \epsilon$, or to V if $d' = \epsilon$. A BI is naturally mapped to a vector equation term describing the same interaction. For instance, the interaction $(g, 1, d, \epsilon, 2, \textsc{At-})$ corresponds to term $\textsc{At-}2(\textsc{In}\ d(g))$, while $(g, 0, d, r, 0, \textsc{Perm-})$ corresponds to $\textsc{Perm-}0(\textsc{In}\ r\ d(\neg g))$.

Bit-vector encoding of basic interactions. To make use of specialized simplification and solving procedures in the solver, we use a bit-vector encoding of BIs. Given an interaction $f = (g, b, d, r, t, op)$, let us denote with $\overline{f}$ its bit-vector encoding, and with $\overline{f}(g) \in \mathbb{B}^{\lceil log_2|G|\rceil}$, $\overline{f}(b) \in \mathbb{B}$, $\overline{f}(d) \in \mathbb{B}^{\lceil log_2(|D|+1)\rceil}$, $\overline{f}(r) \in \mathbb{B}^{\lceil log_2(|SR|+1)\rceil}$, $\overline{f}(t) \in \mathbb{B}^{\lceil log_2|\mathbb{T}|\rceil}$ and $\overline{f}(op) \in \mathbb{B}^2$ the fields of $\overline{f}$ encoding the finite-ranging elements g, b, d, r, t and op, respectively, as subsequences of $\overline{f}$. Thus, the total length of $\overline{f}$ is:

$$N = \lceil log_2|G|\rceil + 1 + \lceil log_2(|D| + 1)\rceil + \lceil log_2(|SR| + 1)\rceil + \lceil log_2|\mathbb{T}|\rceil + 2\ .$$

Note that it is typically unnecessary to allocate $\lceil log_2|\mathbb{T}|\rceil$ bits to describe a delay t, since in any non-trivial model, it is very unlikely to have delays as long as the total execution time. Hence, in most cases it is worth fixing a maximum delay $\bar{t} < |\mathbb{T}|$ for a more efficient bit-vector representation of the interaction.

5 Predicting Sea Urchin Development

We evaluated our approach by considering one of the most complete models of the sea urchin embryonic development [11], which contains *45 genes* (and the vector equations describing their dynamics); *4 spatial domains*; and *2 spatial relations* between domains. The sea urchin is a well established model organism, allowing to study fundamental biological processes in a simpler setting, with mature and powerful experimental methods, and the advantage that many of the underlying mechanisms are conserved in higher level organisms. The model execution spans the discrete time interval $[0, 30]$ hours post fertilization (hpf), corresponding to the early stages of development. We implemented the vector equation language and the synthesis methods in a prototype where we make extensive use of the theory of bit-vectors and uninterpreted functions provided by the Z3 theorem prover [4].

Here we summarize the SMT-based procedure followed for synthesizing a set of vector equations that completely explain all experimental data (both wild-type expression and perturbation effects). The resulting equations have been obtained by considerably changing the original formulation of the sea urchin model that explains most but not all the data. Although the synthesis of vector equations in terms of their regulatory logic and input interactions is fully automated, the final model is derived after a number of manual refinement steps, which allows to progressively add assumptions that are both biologically plausible and logically consistent. This semi-automated strategy helps in excluding models able to reproduce experimental observations but that are not biologically meaningful.

Initially, we validated the correctness of our formalization of the language, by showing that the SMT-based implementation of the original model produced the same expression patterns as reported in [11].

1. Removal of hard-coded data constraints. The original model contains a number of terms formulated in a way that reproduces exactly wild-type observations, of the form $IN\ d \wedge t > t_s \wedge t < t_e$. Such a term basically forces the expression of the output gene to be expressed in a specific domain (d) and time-interval ($[t_s + 1, t_e - 1]$), regardless of the initial conditions or interactions with other genes, thus providing just a description of the observation and no predictive capabilities. We removed from the original equations any occurrence of such terms (present in 8 vector equations), and we found that the resulting model still does not admit solutions that meet experimental observations.

2. Exploration of unsatisfiable vector equations. Identifying the components of a biological model which fail in reproducing the expected behaviour is a hard task, especially if the system is characterized by a large numbers of interacting components like in our case. We address this problem with the automated extraction of unsatisfiable subsets of vector equations, that are not able to explain experimental data regardless of how the other equations are defined. This kind of analysis can point the biologist to the exact source of inconsistency, thus prompting further investigation of the underlying regulatory interactions that make the model contradict experimental data.

In our case, we found that 14 out of the 39 original equations were inconsistent on their own, i.e. solving Problem 2 with a GRNDD model that contains only one among those equations determines inadequacy. Note that the removal of inconsistent equations from the model does not necessarily make it satisfiable. Indeed, our model was found inconsistent even after excluding all these problematic equations, suggesting that any minimal unsatisfiable subset would include a major part of the original equations. This motivates the approach we followed of synthesizing a new equation for every gene, as explained in Step 3.

3. Reformulation of vector equations into functions to synthesize. With our framework we are able to synthesize correct models that also incorporate biological knowledge and assumptions. The procedure consists of replacing each vector equation with an uninterpreted Boolean function over a set of basic interactions to synthesize, one for each term of the original equation. The final equations are obtained through an iterative refinement where initially, each term to synthesize is constrained to meet the original specification only in the input gene and in the temporal operator, which are the most fundamental information of a regulatory interaction. Then, we restrict the space of solutions by gradually restoring the satisfiable features of the initial model (and removing hard-coded observations as described in Step 1).

Specifically, consider a general vector equation f of the form:

$$f = \dots op\ t(IN\ r\ d(b \iff g)) \dots$$

where op is the temporal operator, t the delay, d and r the (possibly empty) domain and spatial relation, respectively, g the input gene and b the Boolean indicating whether g is on or off. A UF uf is considered in place of the initial logical combination of terms, and each term is replaced by a BI as follows:

$$f = uf(\dots, (\{g\}, \mathbb{B}, D^{\epsilon}, SR^{\epsilon}, \mathbb{T}', \{op\}), \dots)$$

where only g and op are constrained to their original values, and $\mathbb{T}' = [0, 12]$ hpf is a reasonably large set of admissible delays. The subsequent refinement steps include restoring b and constraining each function uf so that its interpretation does not negate the involved terms; setting back the original spatial relation and domain; and restricting the delay to the interval $[max(0, t-3), t+3]$ so that we admit an error of 3 hpf, which is reported in [11] to be the resolution of experimental observations. Using the following procedure we have synthesized a biologically plausible model that fully explains all the experimental data represented in [11].[4] The proposed SMT-based approach paves the way for new kinds of in silico experiments, infeasible without automated reasoning and synthesis techniques. A class of useful properties that can be proven concerns the

[4] This synthesized model and computed expressions, together with additional material on the sea urchin GRN, can be found in the supplementary material.

temporal dynamics of gene networks. An example consists in *finding the least maximum delay able to explain observations*, i.e. the minimum upper bound t'_{max} on the synthesizable delays for which the model has a solution. The results and biological interpretation of this analysis are however beyond the scope of this paper.

6 Related Work

6.1 Program Synthesis

In our work synthesis is used to support the abstract reasoning process biologists perform in their research towards deriving mechanistic predictive models. Our system takes observation data and equations templates as specifications and produces new vector equations which fully reproduce the observations and respect the initial templates. In program synthesis the aim is to automate the process of implementation and ensure that the program is correct by construction. Program synthesis accepts specifications and generates code fulfilling those specifications.

Solar-Lezama et al. [12] used sketches to synthesize programs. A sketch is a program with placeholders. Their synthesizer takes a sketch along a working reference implementation and generates an optimized version of the reference implementation. This work has been extended to take into account Boolean constraints and a numerical qualitative objective [3].

Srivastava et al. [13] introduced a so called proof-theoretic synthesis that interprets program synthesis as generalized program verification. It requires the user to provide an input-output functional specification, a description of the atomic operations in the programming language, and a specification of the synthesized programs looping structure, allowed stack space, and bound on usage of certain operations. Their system works by using the Z3 theorem prover to reconcile constraints that relate unknown statements, guards, inductive invariants, and ranking functions.

These synthesis tools require a higher level of specification effort. This is different from our work which uses less information to start with. In particular, we do not ask the biologist to provide a working set of fully specified vector equations, or fine details on the generated equations. However our approach enables biologists to provide additional constraints that capture information inferred from experimental results or biological intuition, which can help the synthesis methods derive more realistic models.

6.2 Computational Biology and Synthesis

The use of synthesis as a method to construct models of biological systems has gained interest in the setting of discrete models that capture biological behavior and enable important predictions and understanding. In particular several approaches have been developed and applied to a classical system in developmental biology describing fate specification of Vulval Precursor Cells (VPCs) in the *C. elegans* nematode. In [7] a method that introduces a set of don't care (dc) Boolean variables that must be assigned values in order to obtain a concrete model is presented. When a dc variable is set to 1, this indicates that the information from the corresponding component does not contribute to the observed result. The problem is formulated as maximizing the number of dc variables that are set to 1, while retaining a model solution that is consistent and can explain all the given known data. This amounts to solving a QBF formula with a maximization goal for the dc variables. In [5] the question of synthesizing concurrent models with bounded asynchrony satisfying a set of experiments that include genetic mutations is addressed and the developed language and tools are applied to the VPC system. Technically this turns out to be a QBF formula with three levels of alternation, and new algorithms are developed to solve it. Additional work on synthesis from scenario-based specifications and its application to the VPC system is described in [8, 6], there the approach considered views the problem as a game between the environment (the experimentalist) and the system (the nematode) and the solution relies on computing fix-points symbolically via compositional methods.

Besides being applicable to the reverse engineering of biological systems, approaches based on the use of formal synthesis and verification are also relevant for their design. For example, in the field of synthetic biology one goal is to engineer genetic networks with specific, useful properties. In [2] temporal logic was used to capture such properties and model-checking was applied to study if a synthetic gene network design was consistent with these specifications. A parameter synthesis strategy was also developed to tune a design in order to achieve the required behavior in [2] and, more recently, similar modular design strategies were proposed in [1].

In [14] the authors introduce Z34Bio, a SMT-based framework for the automated analysis and design of several classes of biological systems, where a common symbolic representation as transition systems over bit-vectors is used to encode multiple classes of biological models, including Boolean networks and chemical reaction networks. The framework was applied in [14] to the verification of DNA circuit design and to the stability analysis of gene regulatory networks, in particular showing how multiple gene knockouts that affect the stability of the system can be automatically identified.

7 Conclusion

Deciphering and understanding the underlying developmental program of an embryo is a fundamental problem. In particular, biologists have spent several decades detailing how a complex gene regulatory network controls the development of sea-urchin embryos. In [11], they have shown, for the first time, that a model of the sea urchin GRN can be turned empirically into a predictive dynamic Boolean computations model. Beyond the mere understanding of a complex process, this result provides them with a tool with which to test *in silico* regulatory circuitry and developmental perturbations.

In this paper, we have presented a generic SMT-based approach that can analyse a GRN in order to synthesize predictive dynamic Boolean computations models. We applied our approach to the sea urchin embryo, and successfully improved the current state-of-the art by providing, for the first time, biologists with models that perfectly explain all known data.

There are several major benefits to our method. First, our approach can save biologists and modelers months or even years of tedious work trying to derive the *best* predictive model. Second, biologists can have greater confidence in in silico tests since the provided models perfectly explain all known data. Third, our method enables to determine that part of a model is inconsistent and cannot explain all known experimental data, helping to focus research efforts on specific biological mechanisms and assumptions that require reevaluation.

This last benefit is related to the finding of minimal unsatisfiable cores in verification applications. However, in our context, it represents much more than the discovery of a transient problem with a particular version of a chip design or software component. Instead, it represents a new way to push the boundaries of knowledge, by driving biologists towards new scientific findings and empowering their capacity to *understand life*.

Acknowledgments. Nicola Paoletti is part supported by the ERC Advanced Grant VERIWARE. The research was carried out during his internship at Microsoft Research Cambridge, UK.

References

1. E. Bartocci, L. Bortolussi, and L. Nenzi. A temporal logic approach to modular design of synthetic biological circuits. In A. Gupta and T. Henzinger, editors, *Computational Methods in Systems Biology*, volume 8130 of *Lecture Notes in Computer Science*, pages 164–177. Springer Berlin Heidelberg, 2013.
2. G. Batt, B. Yordanov, R. Weiss, and C. Belta. Robustness analysis and tuning of synthetic gene networks. *Bioinformatics*, 23(18):2415–2422, 2007.
3. S. Chaudhuri, M. Clochard, and A. Solar-Lezama. Bridging Boolean and quantitative synthesis using smoothed proof search. In S. Jagannathan and P. Sewell, editors, *POPL*, pages 207–220. ACM, 2014.
4. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, volume 4963 of *LNCS*. Springer, 2008.
5. A. Koksal, Y. Pu, S. Srivastava, R. Bodik, J. Fisher, and N. Piterman. Synthesis of biological models from mutation experimentss. In *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 469–482. ACM, 2013.
6. H. Kugler, C. Plock, and A. Roberts. Synthesizing Biological Theories. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23^{rd} Int. Conf. on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 579–584. Springer-Verlag, 2011.
7. H. Kugler, A. Pnueli, M. Stern, and E. Hubbard. "Don't Care" Modeling: A logical framework for developing predictive system models. In Orna Grumberg and Michael Huth, editor, *Proc. 13^{th} Intl. Conference on Tools and Algorithms for*

the Construction and Analysis of Systems (TACAS'07),, volume 4424 of *LNCS*, pages 343–357. Springer-Verlag, 2007.

8. H. Kugler and I. Segall. Compositional Synthesis of Reactive Systems from Live Sequence Chart Specifications. In S. Kowalewski and A. Philippou, editors, *Proc. 15^{th} Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09)*, volume 5505 of *LNCS*, pages 77–91. Springer-Verlag, 2009.
9. F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on*, pages 383–392. IEEE, 2002.
10. I. S. Peter and E. H. Davidson. A gene regulatory network controlling the embryonic specification of endoderm. *Nature*, 474(7353):635–639, 2011.
11. I. S. Peter, E. Faure, and E. H. Davidson. Predictive computation of genomic logic processing functions in embryonic development. *Proceedings of the National Academy of Sciences*, 109(41):16434–16442, 2012.
12. A. Solar-Lezama, R. M. Rabbah, R. Bodík, and K. Ebcioglu. Programming by sketching for bit-streaming programs. In V. Sarkar and M. W. Hall, editors, *PLDI*, pages 281–294. ACM, 2005.
13. S. Srivastava, S. Gulwani, and J. S. Foster. From program verification to program synthesis. In M. V. Hermenegildo and J. Palsberg, editors, *POPL*, pages 313–326. ACM, 2010.
14. B. Yordanov, C. M. Wintersteiger, Y. Hamadi, and H. Kugler. Z34Bio: An SMT-based framework for analyzing biological computation. In *Proceedings of SMT'13*, 2013.