Symbolic Approaches for Finding Control Strategies in Boolean Networks

Christopher James Langmead* and Sumit Kumar Jha

Carnegie Mellon University
5000 Forbes Ave.,
Pittsburgh, PA 15213, USA
E-mail: {cjl,sumit.jha}@cs.cmu.edu

We present algorithms for finding control strategies in Boolean Networks (BN). Our approach uses symbolic techniques from the field of *model checking*. We show that despite recent hardness-results for finding control policies, a model checking-based approach is often capable of scaling to extremely large and complex models. We demonstrate the effectiveness of our approach by applying it to a BN model of embryogenesis in *D. melanogaster* with 15,360 Boolean variables.

Keywords: Systems Biology, Model Checking, Control, Boolean Networks

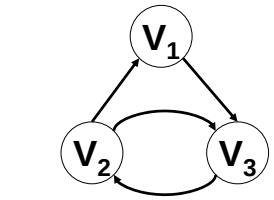
1. Introduction

Computational cellular and systems modeling is playing an increasingly important role in biology, bioengineering, and medicine. The *promise* of computer modeling is that it becomes a conduit through which reductionist data can be translated into scientific discoveries, clinical practice, and the design of new technologies. The *reality* of modeling is that there are still a number of unmet technical challenges which hinder progress. In this paper, we focus on the specific problem of automatically devising control policies for Boolean Networks (BN). That is, given a BN model with external controls, we seek a sequence of control signals that will drive the network to a pre-specified state at (or by) a pre-specified time.

Recently, it has been shown that finding control strategies for arbitrary BNs is NP-hard,[1] but that polynomial-time algorithms exist for deterministic BNs if the network topology forms a tree. In this paper, we consider a more general family of BNs with arbitrary network topologies. Our algorithm uses techniques from the field of *model checking*.[14] Model checking refers to a family of algorithms and data structures for verifying systems of concurrent reactive processes. Historically, model checking has been used to verify the correctness and safety of circuit designs, communications protocols, device drivers, and C or Java code. Abstractions of these systems can be encoded as finite-state models that are equivalent to Boolean networks. We show that existing model checking algorithms can be used to find control strategies for BNs.

Two important features of model checking algorithms are that they are exact *and* scale to real-world problem instances. For example, model checking algorithms for finite-state systems have been able to reason about systems having more than 10^{20} states since 1990,[8] and have been applied to systems with as many as 10^{120} states.[7] More recently, model checking techniques have been created for stochastic systems.[5] These algorithms can be either exact or approximate, and have also been shown to scale to systems with as many as 10^{30} states.[16] In this paper, we will show that model checking can be used to devise control strategies for very large Boolean networks (up to 15,360 nodes) within seconds or minutes. These techniques are useful in their own right, but will also lay the groundwork for future techniques for finding control strategies in models with asynchronous and stochastic dynamics.

*Corresponding author



$\psi_1 \equiv V_1(t+1) = V_1(t) \wedge V_2(t)$
$\psi_2 \equiv V_2(t+1) = \neg V_3(t)$
$\psi_3 \equiv V_3(t+1) = V_1(t) \wedge V_2(t)$

time t			time t + 1		
V_1	V_2	V_3	V_1	V_2	V_3
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Fig. 1. (**Left**) A Boolean Network (BN). A BN consists of a graph and a set of Boolean functions. The vertices of the graph correspond to Boolean variables and the edges describe functional dependencies. The Boolean functions describe the evolution of the model from time t to $t+1$. The functions can contain any combination of Boolean connectives. (**Right**) A transition relation encoding the same dynamics as the BN. Notice that the BN is a compact encoding of the transition relation.

2. Boolean Networks

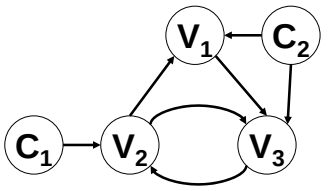
A BN is a pair, $B = (\mathcal{G}, \Psi)$, where $\mathcal{G} = \{V, E\}$ is a directed graph, and $\Psi = \{\psi_1, \psi_2, ..., \psi_{|V|}\}$ is a set of Boolean functions. Each vertex, $v_i \in V$, represents a Boolean random variable. The state of variable v_i at discrete time t is denoted by $v_i(t)$. The state of all vertices at time t is denoted by $\mathbf{v}(t)$. The directed edges in the graph specify causal relationships between variables. Let $Pa(v_i) \subset V$ be the parents of v_i in the directed graph and let $k_i = |Par(v_i) \cup \{v_i\}|$. A node can be its own parent if we add a self-edge. Each Boolean function $\psi_i : \{0,1\}^{k_i} \mapsto \{0,1\}$ defines the dynamics of v_i from time t to $t+1$ based on the state of its parents at time t. Thus, the set Ψ defines the dynamics of the entire BN. An example BN is shown in Figure 1-A. Notice that a BN is simply a compact encoding of a transition relation over V (Fig 1-B).

This basic model can be extended to define a BN with external controls by augmenting our graph with special control nodes, $\mathcal{G} = \{V, C, E\}$. Each control node, c_i, is connected to one or more nodes in V by a directed edge going from c_i to v_j (Fig. 2). The control nodes themselves are externally manipulated. That is, there is no ψ_i that defines the dynamics of c_i.

Consider a set of initial states, I, for the nodes in V specified in terms of a Boolean expression. For example, the expression $I = (v_1 \wedge \neg v_2 \wedge v_3)$ defines the set $\{(1,0,1)\}$, and $I = (v_1 \wedge v_3)$ defines the set $\{(1,0,1),(1,1,1)\}$. We define a set of goal states, F, in a similar fashion. A *control policy*, $\Gamma = \langle \mathbf{c}(0), \mathbf{c}(1), ..., \mathbf{c}(t) \rangle$, is a set of Boolean vectors that defines a sequence of signals to be applied to the control nodes. The BN control problem is to find a control policy that drives the BN such that $\mathbf{v}(0) = I$ and $\mathbf{v}(t) = F$. Our goal in this paper is to algorithmically generate Γ for a given, B, I, F, and t, or to indicate that no such policy exists.

3. Model Checking

The term *model checking*[14] refers to a family of techniques from the formal methods community for verifying systems of concurrent reactive processes. The field of model checking was born from a need to formally verify the correctness of hardware designs. Since its inception in 1981, it has expanded to encompass a wide range of techniques for formally verifying finite-state transition systems, including those with non-deterministic (i.e., asynchronous) or stochastic dynamics. Model checking algorithms are simultaneously theoretically very interesting and very useful in practice. Significantly, they have become the preferred method for formal verification in industrial settings over traditional verification methods like theorem proving, which often need guidance from an expert hu-



$\psi_1 \equiv V_1(t+1) = V_1(t) \wedge V_2(t) \vee \neg C_2(t)$

$\psi_2 \equiv V_2(t+1) = \neg V_3(t) \wedge C_1(t)$

$\psi_3 \equiv V_3(t+1) = V_1(t) \wedge V_2(t) \wedge C_2(t)$

	V_1	V_2	V_3
Start (t=0)	0	0	0
Goal(t=3)	1	0	0

t	V_1	V_2	V_3	C_1	C_2
0	0	0	0	1	0
1	1	1	0	0	1
2	1	0	1	0	0
3	1	0	0		

Fig. 2. (**Left**) A BN with two control nodes (C_1 and C_2). (**Right top**) An initial state and time-sensitive goal. (**Right bottom**) A control policy (last two columns) that achieves the goal at the specified time.

man user. A complete discussion of model checking theory and practice is beyond the scope of this paper. The interested reader is directed to [14] for a detailed treatment of the subject.

3.1. *Modeling Concurrent Systems as Kripke Structures*

An *atomic proposition*, a, is a Boolean predicate referring to some property of a given system. Let AP be a set of atomic propositions. A *Kripke structure*, M, over AP is a tuple, $M = (S, R, L)$. Here, S is a finite set of states, $R \subseteq S \times S$ is a total transition relation between states, and $L : S \mapsto 2^{AP}$ is a labeling function that labels each state with the set of atomic propositions that are true in that state. Variations on the basic Kripke structure exist. For example, if the system is stochastic, then we replace the transition relation, R, with a stochastic transition matrix, T where element $T(i, j)$ contains either a transition rates (for continuous-time Markov models) or a transition probability (for discrete-time Markov models).

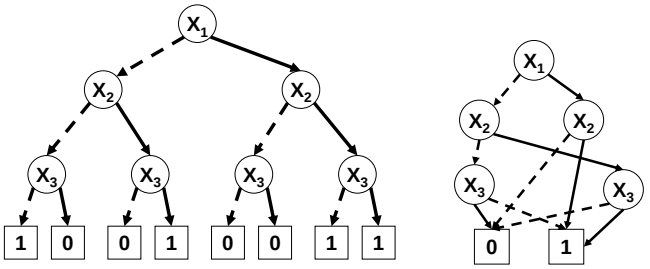
It is easy to see that, in principle, BNs can be encoded as Kripke structures. The state space, S, corresponds to the $2^{|V \cup C|}$ possible states of the BN . We will use the atomic propositions to reveal the state of each variable in the model. That is, $|AP| = |V \cup C|$ and the propositions will be of the form: "*is the state of* v_i 1?" The labeling function, L, can thus be used to define the set of initial states, I, and goal states, F (see Sec. 2). The transition relation, R, corresponds to the table in Figure 1-B. Alternatively, a stochastic transition matrix, T, can be used to encode the stochastic dynamics of the PBN. Naturally, it is generally not possible to explicitly instantiate the Kripke structure for an arbitrary BN because the state space is exponential in the number of nodes. In the next section, we discuss how Kripke structures can be efficiently encoded symbolically.

3.2. *Symbolic Encodings of Kripke Structures*

The basis for symbolic encodings of Kripke structures, which ultimately facilitated industrial applications of model checking, is the reduced ordered Binary Decision Diagrams (BDDs) introduced by Bryant[6] (Fig. 3). BDDs are directed acyclic graphs that symbolically and compactly represent binary functions, $f : \{0, 1\}^n \mapsto \{0, 1\}$. While the idea of using decision trees to represent boolean formulae arose directly from Shannon's expansion for Boolean functions, two key extensions made by Bryant were i) the use of a fixed variable ordering, and ii) the sharing of sub-graphs. The first extension made the data structure canonical, while the second one allowed for compression in its storage. A third extension, also introduced in [6], is the development of an algorithm for applying Boolean operators to pairs of BDDs, as well as an algorithm for composing the BDD representations of pairs of functions. Briefly, if f and g are Boolean functions, the algorithms implementing operators APPLY(f,g,op) and COMPOSE(f,g) compute directly on

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

(A)



(B) (C)

Fig. 3. (**A**) A truth table for the Boolean function $f(x_1, x_2, x_3) = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$ (**B**) A Binary Decision Tree of the truth table in (A). A dashed edge emanating from variable/node x_i indicates that x_i is false. A solid edge indicates that x_i is true. (**C**) A Binary Decision Diagram of the truth table in (A). Notice that it is a more compact representation that the Binary Decision Tree.

the BDD representations of the functions in time proportional to $O(|f||g|)$, where $|f|$ is the size of the BDD encoding f. In this paper, BNs and the desired behaviors are encoded symbolically using BDDs. Model checking algorithms, which call APPLY and COMPOSE as subroutines, are then used to find a valid control strategy, or prove that none exists.

In practice, the construction of the BDDs is done automatically from a high-level language describing the finite-state system and its behavior. In this paper, we use the specification language used in the symbolic model checking tool NuSMV.[12]

We note that BDDs can be generalized to Multi-terminal BDDs[13] (MTBDD), which encode discrete, real-valued functions of the form $f : \{0, 1\}^n \mapsto \mathbb{R}$. Significantly, MTBDDs can be used to encode real-valued vectors and matrices, and algorithms exist for performing matrix addition and multiplication over MTBDDs.[13] These algorithms play an important role in several model checking algorithms for stochastic systems[5] which, in turn, we have used to develop algorithms for finding control strategies in BNs with stochastic behaviors. Due to space limitations, we will focus on algorithms for deterministic BNs in this paper and report the algorithms for stochastic BNs elsewhere.

3.3. *Temporal Logics*

Temporal logic is a formalism for describing behaviors in finite-state systems. It has been used since 1977 to reason about the properties of concurrent programs.[23] There are a number of different temporal logics from which to chose, and different logics have different expressive powers. In this paper, we use a small subset of the Computation Tree Logic (CTL). CTL formulae can express properties of *computation trees*. The root of a computation tree corresponds to the set of initial states (i.e., I) and the rest of the (infinite) tree corresponds to all possible paths from the root. A complete discussion of CTL and temporal logics is beyond the scope of this paper. The interested reader is directed to [14] for more information.

The syntax of CTL is given by the following minimal grammar:

$$\phi ::= a \mid true \mid (\neg\phi) \mid (\phi_1 \wedge \phi_2) \mid \mathbf{EX}\phi \mid \mathbf{E}[\phi_1 \mathbf{U} \phi_2]$$

Here, $a \in AP$, is an atomic proposition; "true" is a Boolean constant; $\neg$ and $\vee$ are the normal logical operators; $\mathbf{E}$ is the existential path quantifier (i.e., "there exists some path from the root in the computation tree"); and $\mathbf{X}$ and $\mathbf{U}$ are temporal operators corresponding to the notions of "in the next state", and "until", respectively. Given these, additional operators can be derived. For example, "false" can be derived from "$\neg true$" and the universal quantifier, $\mathbf{AX}\phi$, can be defined as $\neg\mathbf{EX}\neg\phi$.

Given some path through the computation tree, $\pi = \langle \pi[0], \pi[1], \ldots \rangle$, the semantics of a CTL formula are defined recursively:

$\pi \models a$ iff a $\in$ L($\pi[0]$)
$\pi \models true, \forall \pi$
$\pi \models \neg\phi$ iff $\pi \not\models \phi$
$\pi \models \phi_1 \wedge \phi_2$ iff $\pi \models \phi_1$ and $\pi \models \phi_2$
$\pi \models \mathbf{EX}\phi$ iff $\pi[1] \models \phi$
$\pi \models \mathbf{E}[\phi_1 \mathbf{U} \phi_2]$ iff $\exists i \geq 0, \pi[i] \models \phi_2 \wedge \forall j < i, \pi[j] \models \phi_1$

Here, the notation"$\pi \models \alpha$" means that π *satisfies* α.

3.4. *Model Checking Algorithms*

A model checking algorithm takes a Kripke structure, $M = (S, R, L)$, and a temporal logic formula, ϕ, and finds the set of states in S that satisfy ϕ: $\{s \in S \mid M, s \models \phi\}$. The complexity of model checking algorithms varies with the temporal logic and the operators used. For the types of formulas used in this paper (see Sec. 4), an explicit state model checking algorithm requires time $O(|\phi|(|S| + |R|))$, where $|\phi|$ is the number of sub-formulas in ϕ ([14] p. 38).

Of course, for very large state spaces, even linear time is unacceptable. *Symbolic model checking* algorithms operate on BDD encodings of the Kripke structure and CTL formula. Briefly, the temporal operators of CTL can be characterized in terms of fixpoints. Let $\mathcal{P}(S)$ be the powerset of S. A set $S' \subseteq S$ is a fixpoint of a function $\tau : \mathcal{P}(S) \mapsto \mathcal{P}(S)$ if $\tau(S') = S'$. Symbolic model checking algorithms define an appropriate function, based on the formula, and then iteratively find the fixpoint of the function. This is done using set operations that operate directly on BDDs. The fixpoint of the function corresponds exactly to $\{s \in S \mid M, s \models \phi\}$. The interested reader is encouraged to read [14], ch. 6 for more details.

The symbolic model checking algorithms used in this paper are exact. We note that there are also approximation algorithms for model checking (e.g., [27]), which employ sampling techniques and hypothesis testing. Such algorithms provide guarantees, in terms of the probability of the property being true, and can scale to much larger state spaces. These do not use BDDs, but rather operate on the high-level language description of the finite-state model.

4. A Symbolic Model Checking Approach to Finding Control Policies

The use of model checking algorithms for finding control strategies requires three steps:

First, the BN must be encoded using a high level language for describing finite-state models. Different model checking software use different modeling languages. In Figure 4, we show pseudo-code for encoding the BN in figure 2. This pseudo-code is based on the language used in the model-checking tool NuSMV. The code contains a block of variable definitions. In the example, we declare Boolean variables for v_1, v_2, v_3, c_1,and c_2. The set of initial states, I, is encoded using "init" statements. The update rules, Ψ, are encoded using "next" statements. A single variable COUNTER is declared that marks the passage of time. A "next" statement for COUNTER updates the counter.

Second, a CTL formula must be written. In this paper, we are concerned with CTL formulae that ask whether it is possible to end up in the goal state(s), F, at time t. Let ϕ_F be a formula describing the goal state. This formula can describe any subset of the variables in the BN. For example, $\phi_F := v_1 \wedge \neg v_2 \wedge v_3$ or $\phi_F := v_1 \wedge v_3$ are both valid formulas. The former chooses to specify the state of each variable, the latter does not. Let $\phi_t :=$COUNTER$= t$ be a Boolean formula that evaluates to true if the variable COUNTER is t. The formula $\phi := \mathbf{E}[\neg\phi_F \mathbf{U}(\phi_F \wedge \phi_t)]$ can be used to find a control policy. In English, this formula says: "*There exists a path that enters state F for the first time at time t*". Alternatively, if we wish to relax the restriction that the BN cannot enter state F before time t, we would use the formula $\phi' := \mathbf{E}[true \mathbf{U}(\phi_F \wedge \psi_t)]$, which translates as "*In the future, the model will be in F at time t.*" Temporal logics are very expressive and can encode a number of complex behaviors. For example, it is possible to specify particular

```
MODULE BN
VAR
  V1: boolean;          // variable node 1
  V2: boolean;          // variable node 2
  V3: boolean;          // variable node 3
  C1: boolean;          // control node 1
  C2: boolean;          // control node 2
  COUNTER: 0 .. T+1;// counter
ASSIGN
  init(V1) := 1;
  init(V3) := 1;
  next(V1) := (V1 & V2) | !C2 ;
  next(V2) := ! V3 & C1 ;
  next(V3) := V1 & V2 & C2 ;
  next(COUNTER) := COUNTER+1 ;
```

Fig. 4. Pseudocode based on the language used in the symbolic model checking program NuSMV. This code implements the BN in Figure 2. The code consists of a module with variable declaration statements, "init" statements that initialize the variables, and "next" statements that implement each ϕ_i and increment a counter.

milestones through which the model should pass en route to the final goal. That is, one can construct formula that say that the BN should enter state X_1 before X_2, must enter X_2 by time t_1, and must reach the goal state at exactly time t_2. This expressive power is one of the key advantages of a model checking based approach to the design of control policies.

Finally, we apply an appropriate symbolic model checking algorithm to find a control policy. If a control policy exists (i.e., if ϕ is true), then we ask the model checking algorithm for a *witness*, π_w, to the formula. The control policy, Γ, is then simply extracted from π_w by reading off the values of $\langle \mathbf{c}(0), \mathbf{c}(1), ..., \mathbf{c}(t) \rangle$[a].

5. Related Work

Boolean Networks have been used extensively to model complex biological systems (e.g., [2,3,17,18]). The design of control strategies for Boolean networks and related models has been considered by a number of different authors (e.g.,[1,11,15,24]). Akutsu and co-workers[1] were the first to show that the design of control policies is NP-hard. They also provide a polynomial-time algorithm that works on the special case where the topology of the BN forms a tree. The primary difference between our work and these is that our method is based on symbolic model checking and we place no restriction on the topology of the network. We will show in the next section that despite the fact that the problem is NP-hard, in practice model checking based approaches to control policy design can scale to very large models. Of course, the hardness result implies that our approach will not apply to every BN.

Recently, there has been growing interest in the application of formal methods, including model checking to biology. Most applications of model checking in biology have been directed to modeling biochemical and regulatory networks, (e.g.,[4,9,10,19, 22]), although not for the design of control policies. In our own work, we have applied model checking,[20] and a related technology based on decision procedures[21] to the protein folding problem.

6. Results

We present results from two kinds of experiment. The first experiment is designed to highlight the scalability of a model checking based approach to control policy design. The second experiment applies our approach to an existing BN model of embryo development in drosophila.

[a]Equivalently, as we performed in our experiments, we can request a counterexample to $\neg\phi$.

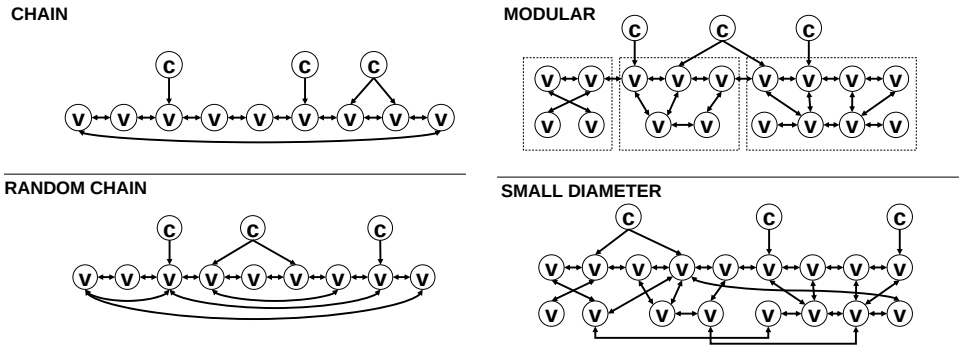


Fig. 5. Network topologies used in our experiments on scalability. *Chain* describes a model where the variables form a circular chain. *Random Chain* describes a model where the variables form a circular chain, but a random number of "long-range" edges are added. *Modular* describes a model with coupled modules. Each module is outlined. *Small Diameter* describes a model where a graph has a small diameter. In each case, the placement of the control nodes is random.

6.1. *Scalability*

We have performed a large-scale study on randomly generated BNs in order to characterize the scalability of our approach. In total, we considered 13,400 separate BNs. We considered several different network topologies, which are shown in Figure 5. These topologies are meant to reflect different kinds of networks ranging from simple feedback loops (*chains*), feedback loops with complex topologies (*random chains*), loosely coupled modules (*modular*), to a dense network (*small diameter*). Within each network category, we performed separate experiments randomly generating graphs by varying: a) the number of non-control variables over the interval [10,640]; b) the average number of parents for each node over the interval [2, 8]; c) the number of control nodes over the interval [2,64]; d) the number of variables specified in the goal state, F, over the interval [4,80]; and e) the target time, t, over the interval [1,32]. For each combination of parameters, we generated 100 BNs randomly, constructed a CTL formula, and identified a control strategy using NuSMV. Due to space limitations, we will simply report that each experiment took less than 12 minutes on a single Pentium 3 processor with 2 GB of memory. The mean and median runtimes were 2 and 0.6 seconds, respectively. The longest runtime (693 seconds) was on a random chain topology model with 80 nodes, an average in-degree of 4, 4 control nodes, a target specifying the state of 4 variables, and a time of 32. These results suggests that a model checking approach to policy design scales well to randomly generated BNs.

6.2. *Application To* D. Melanogaster *Embryo Development*

To test our approach on a BN for a real biological process, we applied it to the task of finding control policies to an existing model of fruit fly embryo development.[3] Briefly, Albert and Othmer have developed a BN model of the segment polarity gene network in *D. Melanogaster* (Fig. 6-left). The model comprises 5 RNAs: (*wingless (wg)*; *engrailed (en)*; *hedgehog (hh)*; *patched (ptc)*; and *cubitus interruptus (ci)*), and 10 proteins: (*WG*; *EN*; *HH*; *PTC*; *CI*; *smoothened (SMO)*; *sloppy-paired (SLP)*; a transcriptional repressor, *(CIR)*, for *wg*, *ptc*, and *hh*; a transcriptional activator, *(CIA)* for *wg* and *ptc*; and the *PTC-HH* complex, (*PH*)). Each molecule is modeled as a Boolean variable and the update rules are Boolean formulas that take into account both intra-cellular state, and inter-cellular communication. The Albert and Othmer research did not consider the question of control policy design.

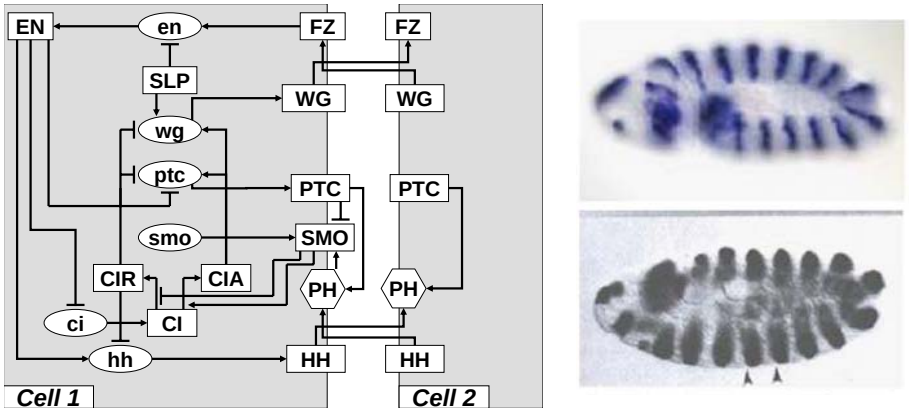


Fig. 6. **(Left)** The drosophila segment polarity BN from Albert and Othmer. The figure shows one cell in detail (large grey box), and the inter-cellular signals (WG and HH) between two adjacent cells. See text for more details. **(Right)** Expression pattern of *wg* in wild-type (top) and a "broad-stripe" mutant embryo (bottom).

Albert and Othmer have demonstrated that the Boolean model accurately reproduces both wild-type and mutant behaviors. In their experiments, they consider a 1-dimensional array of cells initialized to the experimentally characterized cellular blastoderm phase of *Drosophila* development, which immediately precedes the activation of the segment-polarity network. The purpose of the segment-polarity network is to maintain a pattern of expression throughout the life of the fly that defines the boundaries between *parasegments*, small linear groupings of adjacent cells. Two possible parasegment boundary expression patterns are shown in Figure 6-(right)[b]. In the Albert and Othmer work, the parasegments are four cells wide. We note that the steady-state expression patterns of different sub-populations of cells differ due to inter-cellular communication — this is precisely the mechanism by which the parasegment boundaries are maintained. That is, the fate of every cell is *not* the same, even though each cell is running the same regulatory network.

In our experiment, we modified the Albert and Othmer BN in two ways. *First*, we considered a 32x32, two-dimensional array of cells of dimension, instead of the 1x12 one-dimensional array of cells considered in [3]. We believe that this extension to a two-dimensional model is the first of its kind; we also believe that the 15,360 Boolean variables in our model is the largest ever considered for the purpose of control policy design. Topologically, this network most closely resembles the "modular" network in Figure 5. Adjacent cells in the network can communicate, which introduces loops in overall topology of the BN for the 16x16 array of cells. *Second*, we modified the network such that the RNAs *wg* and *hh* becomes a control node in the network. In principle, one could control RNAs through RNA-silencing or micro RNAs. We used our methods to design two control policies for *hh*. The first is designed to drive the system to either the wild-type expression pattern (Fig. 6-A (top)) and the other to a "broad-stripe" pattern (Fig. 6-A (bottom)). Our algorithms successfully found the two control policies in 6.1 and 6.2 minutes, respectively. The computation was dominated by the time to construct the BDDs. We believe these results strongly suggest that our approach can be used to find control signals for biologically relevant BNs of substantial size.

[b]The images in Fig. 6-A are taken from http://www.fruitfly.org (top) and [26] (bottom)

7. Conclusions and Future Work

We have introduced an effective means for automatically discovering control sequences for Boolean networks based on techniques from the field of model checking. Our approach scales to very large BNs, having as many as 15,360 nodes, and runs in seconds to minutes. We note that, due to the inherent computational complexity of finding control policies in BNs,[1] we cannot claim that our approach will scale to *every* BN of large size. Rather, our results suggest that the modular design of "real" biological networks may reduce the possibility of encountering worst-case instances. This is an interesting question and we believe it is related to the phenomenon of canalizing functions and other generic properties of BNs (e.g., [25]).

BNs have been used widely to model a range of biological phenomena. However, the fact that BNs made strong assumptions about the binary nature of each variable (i.e., active or inactive), the synchronous nature of the updates, the assumption that time unfolds in discrete steps, and the assumption that the dynamic are deterministic. Ultimately, these assumptions limit the overall applicability of BNs. We note that our approach to control policy design can be adapted for use to a much broader range of models including those with continuous-valued variables, asynchronous updates between variables, continuous time, and stochastic transitions. We are presently pursuing these goals as part of ongoing research.

Acknowledgments

This research was supported by a U.S. Department of Energy Career Award (DE-FG02-05ER25696), and a Pittsburgh Life-Sciences Greenhouse Young Pioneer Award to C.J.L.

References

1. T. Akutsu, M. Hayashida, W.K. Ching, and M. Ng. On the complexity of finding control strategies for boolean networks. *Proc. 4th Asian Pacific Bioinf. Conf.*, pages 99–108, 2006.
2. T. Akutsu, S. Miyano, and S. Kuhara. Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics*, 16(8):727–734, 2000.
3. R. Albert and H. G. Othmer. The topology of the regulatory interactions predics the expression pattern of the segment polarity genes in drosophila melanogaster. *Journal of Theoretical Biology*, 223:1–18, 2003.
4. M. Antoniotti, A. Policriti, N. Ugel, and B. Mishra. Model building and model checking for biochemical processes. *Cell Biochem Biophys.*, 38(3):271–286, 2003.
5. C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. *Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP'97)*, 1256:430–440, 1997.
6. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.
7. J.R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 3(4):401–424, 1993.
8. J.R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Proc. Fifth Ann. IEEE Symposium on Logic in Computer Science*, pages 428–439, 1990.
9. M. Calder, V. Vyshemirsky, D. Gilbert, and R. Orton. Analysis of signalling pathways using the PRISM model checker. *Proc. Computational Methods in Systems Biology (CMSB'05)*, pages 179–190, 2005.
10. N. Chabrier and F. Fages. Symbolic Model Checking of Biochemical Networks. *Proc*

1st Internl Workshop on Computational Methods in Systems Biology, pages 149–162, 2003.
11. P. C. Chen and J. W. Chen. A markovian approach to the control of genetic regulatory networks. *Biosystems*, 90(2):535–45, 2007.
12. A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, P. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, pages 359–364, 2002.
13. E.M. Clarke, M. Fujita, P. C. McGeer, J.C.-Y. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient datastructure for matrix representation. *IWLS '93 International Workshop on Logic Synthesis*, 1993.
14. E.M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.
15. A. Datta, A. Choudhary, M. L. Bittner, and E.R. Dougherty. External control in markovian genetic regulatory networks. *Mach. Learn.*, 52(1-2):169–191, 2003.
16. L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. *Proc. 6th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, 1785:395–410, 2000.
17. S.E. Harris, B.K. Sawhill, A. Wuensche, and S. Kauffman. A model of transcriptional regulatory networks based on biases in the observed regulation rules. *Complex.*, 7(4):23–40, 2002.
18. S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.
19. M. Kwiatkowska, G. Norman, D. Parker, O. Tymchyshyn, J. Heath, and E. Gaffney. Simulation and verification for computational modelling of signalling pathways. *WSC '06: Proceedings of the 38th conference on Winter simulation*, pages 1666–1674, 2006.
20. C.J. Langmead and S. K. Jha. Predicting protein folding kinetics via model checking. *Lecture Notes in Bioinformatics: The 7th Workshop on Algorithms in Bioinformatics (WABI)*, pages 252–264, 2007.
21. C.J. Langmead and S. K. Jha. Using bit vector decision procedures for analysis of protein folding pathways. *Fourth Workshop on Constraints in Formal Verification*, page in press, 2007.
22. C. Piazza, M. Antoniotti, V. Mysore, A. Policriti, F. Winkler, and B. Mishra. Algorithmic Algebraic Model Checking I: Challenges from Systems Biology. *17th Internl Conf. on Comp. Aided Verification (CAV)*, pages 5–19, 2005.
23. A. Pnueli. The temporal logic of programs. *Proceedings of the 18th IEEE. Foundations of Computer Science (FOCS)*, pages 46–57, 1977.
24. P. Ranadip, D. Aniruddha, L. Bittner, and R. Dougherty. Intervention in context-sensitive probabilistic boolean networks. *Bioinformatics*, 21(7):1211–1218, 2005.
25. I. Shmulevich, H. Lhdesmki, E. R. Dougherty, J. Astola, and W. Zhang. The role of certain post classes in boolean network models of genetic networks. *Proc Natl Acad Sci U S A*, 100(19):10734–10739, 2003.
26. T. Tabata, S. Eaton, and T. B. Kornberg. The drosophila hedgehog gene is expressed specifically in posterior compartment cells and is a target of engrailed regulation. *Genes Dev.*, 6(12B):2635–2645, 1992.
27. H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, pages 223–235, 2002.