9-1-2006

Temporal-logics as query languages for Dynamic Bayesian Networks: Application to D. melanogaster Embryo Development

Christopher J. Langmead
Carnegie Mellon University, cjl@cs.cmu.edu

Sumit Kumar Jha
Carnegie Mellon University

Edmund M. Clarke
Carnegie Mellon University, emc@cs.cmu.edu

Temporal-logics as query languages for Dynamic Bayesian Networks: Application to *D. melanogaster* Embryo Development

C. J. Langmead[*,†], **S. Jha**[*], **E. M. Clarke**[*]

September 2006
CMU-CS-06-159

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[*]Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA 15213.
[†] Department of Biological Sciences, Carnegie Mellon University, Pittsburgh, PA, USA 15213

E-mail: cjl@cs.cmu.edu

CJL is supported by a Young Pioneer Award from the Pittsburgh Lifesciences Greenhouse and a CAREER award from the U.S. Department of Energy.

Keywords: Biological networks, Dynamic Bayesian networks, Systems Biology, Model Checking

Abstract

This paper introduces novel techniques for exact and approximate inference in Dynamic Bayesian Networks (DBNs) based on algorithms, data structures, and formalisms from the field of *model checking*. Model checking comprises a family of techniques from for formally verifying systems of concurrent reactive processes. We discuss: i) the use of temporal logics as a query language for inference over DBNs; ii) translation of DBNs into probabilistic reactive modules; and iii) the use of symbolic data structures and algorithms for deciding complex stochastic temporal logic formulas. We demonstrate the effectiveness of these new algorithms by examining the behavior of an enhanced expression model of embryogenesis in *D. melanogaster*. In particular, we converted an existing deterministic developmental model over a one-dimensional arrays of cells into a stochastic model over a two dimensional array of cells. Our results confirm that the rules which govern the one-dimensional model also display wild-type expression patterns in the two-dimensional case within certain parameter bounds.

1 Introduction

Computational cellular and systems modeling plays an important role in biology, bioengineering, and medicine. Fundamentally, a computational model is a concrete instantiation of a particular hypothesis or theory. As such, a model has two phases in its lifecycle. In the first phase, the model is constructed and refined until it accurately reproduces known behaviors. In the second phase, the refined model can then be used to: a) gain insights into the underlying phenomenon; b) make novel predictions about steady-state or transient behaviors of the system under different conditions; and c) design control strategies. The rapid growth of the field of systems biology has resulted in a variety of new models for a diverse set of biological phenomena including circadian rhythms (e.g., [41]), regulatory pathways (e.g., [46]), metabolic pathways (e.g., [27]), and bacterial infection (e.g., [22]). It can be safely assumed that the variety, scope, and complexity of such models will continue to grow. Modeling techniques will need to keep pace with these developments.

There are variety of modeling techniques in use within systems biology. *Dynamic Bayesian Networks* (DBNs) [36] are a popular method for studying state-transition systems with stochastic behavior. DBNs comprise a large number of *probabilistic graphical models* [32], including the familiar Hidden Markov Model (HMM). DBNs have been used widely in biology to model sequential (e.g., [26, 12, 17]) and temporal data (e.g., [37, 42, 6]). There are two basics tasks associated with DBNs: learning and inference. Learning involves estimating the parameters of the model from a set of training data. Inference encompasses a number of tasks involving making predictions based on the model.

This paper introduces a new set of algorithms for performing inference in DBNs using techniques from the field of *model checking* [21]. Model checking refers to a family of algorithms, and their associated data structures, for verifying systems of concurrent reactive processes. We will use these techniques to reason formally about the dynamic behavior of complex systems. 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. An important feature of model checking algorithms is that they are exact and scale to real-world problems. For example, model checking algorithms for deterministic systems have been able to reason about systems having more than 10^{20} states since 1990 [11], and have been applied to systems with as many as 10^{120} states [9, 10]. More recently, model checking techniques have been created for stochastic systems. These so-called probabilistic model checking techniques are central to this paper. Probabilistic model checking techniques can be either exact or approximate. They also scale to large systems, and have been applied to systems with as many as 10^{30} states [25].

Our key observation is that performing inference in DBNs and performing probabilistic model checking are very similar activities. This suggests that model checking algorithms can be used to perform inference in DBNs, and visa-versa. The primary contribution of our paper, however, lies in the demonstration that a model checking-based approach results in a substantially more general framework for performing inference in DBNs. DBN inference algorithms are *instance-based*. That is, one makes predictions by conditioning the model on a single, finite-length observation sequence. Model checking, on the other hand, poses queries as formulas in a *temporal logic*. These formulas can easily encode finite-length observation sequences, like those used in traditional DBN

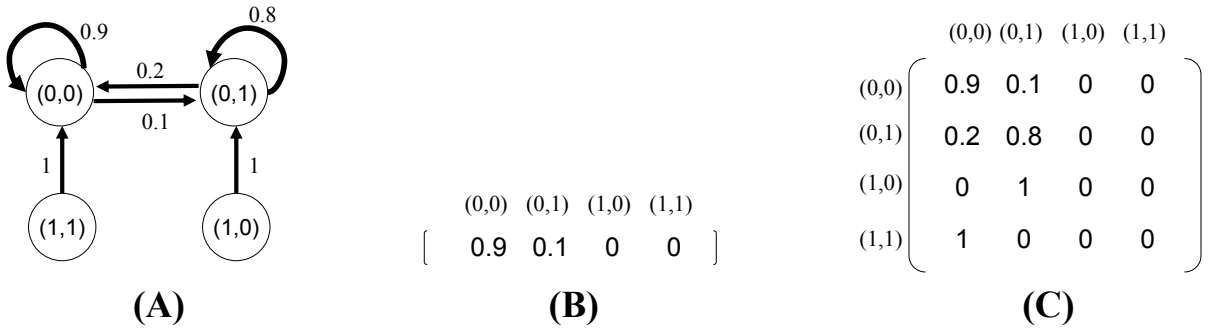


Figure 1: (**A**) A finite-state Markov process,(S, T, π), in graphical form. The nodes in the graph correspond to the state-space, S. Each state corresponds to one of the possible combinations of two Boolean random variables, Z_1 and Z_2. (**B**) A vector encoding π, the prior probability over S. (**C**) The state transition matrix, T.

inference algorithms. More importantly, temporal logic formulas can *also* encode queries over a) infinite execution sequences, b) execution "trees" representing all possible outcomes for non-deterministic systems, and c) *logical* orderings of events. That is, temporal logics can be used to ask questions about equivalence classes of behaviors well beyond the capabilities of instance-based inference methods. Model checking algorithms are then used to answer these questions either exactly, or using approximate methods. Thus, a model- checking based approach to inference can greatly enhance the power of DBNs.

The contributions of this paper are as follows:

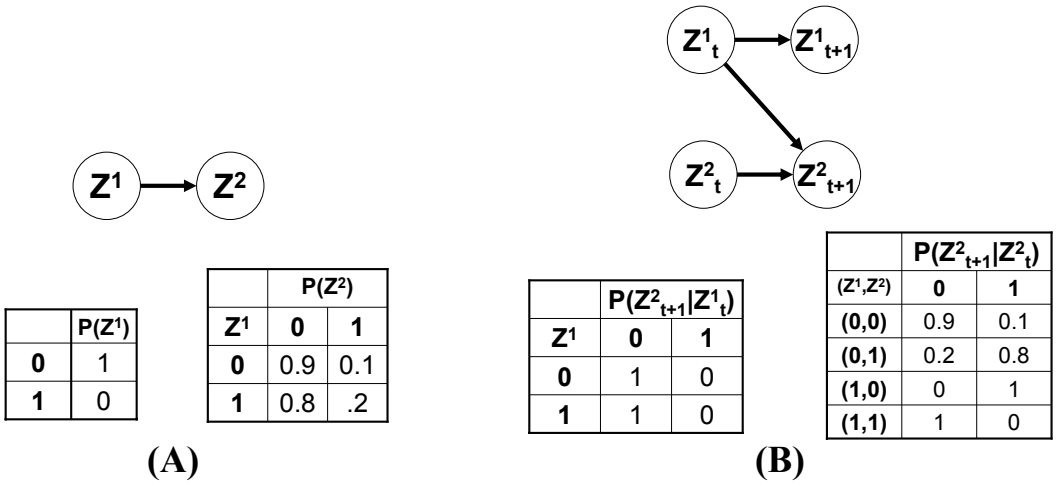
- We establish the connection between DBNs and probabilistic model checking.
- We introduce algorithms for performing exact and approximate inference in DBNs via model checking.
- We introduce temporal logics to DBN, facilitating a more general means of performing inference.
- We demonstrate our algorithms on a novel model of development in *Drosophila*, building on work initially reported by [1].

2 Background

The primary aim of this paper is to combine techniques from two fields that have evolved independently: Dynamic Bayesian Network modeling and Model Checking. We briefly review each field in the following two sub-sections.

2.1 Dynamic Bayesian Networks

A DBN is a compact encoding for Markov and semi-Markov processes. A Markov process is a triple, (S, T, π) where S is a state space of size n, $T : S \times S \rightarrow [0, 1]$ is an $n \times n$ stochastic



	P(Z¹)
0	1
1	0

	P(Z²)	
Z¹	0	1
0	0.9	0.1
1	0.8	.2

	P(Z²t+1|Z¹t)	
Z¹	0	1
0	1	0
1	1	0

	P(Z²t+1|Z²t)	
(Z¹,Z²)	0	1
(0,0)	0.9	0.1
(0,1)	0.2	0.8
(1,0)	0	1
(1,1)	1	0

Figure 2: A DBN for the finite-state Markov process in Figure 1. **(A)** A static Bayesian network encoding the prior distribution (π) over S. The nodes represent Boolean random variables Z^1 and Z^2. The CPD tables are also shown. **(B)** A two-slice temporal Bayesian network encoding the state transition matrix, T. The nodes represent Boolean random variables Z^1 and Z^2 at times t and $t+1$. The CPD tables are also shown. The pair (B_S,B_T) comprises a DBN.

transition matrix, and π is a prior probability distribution over S. For a finite-state model[1], the state space corresponds to the cartesian product of a set of m random variables, $Z = \{Z^1, Z^2, ..., Z^m\}$, over a finite domain, D. Thus, $|S| = n = O(|D|^m)$. The Markov property assures us that, in a kth-order Markov model, the probability of being in some state $s \in S$ at time t only depends on the prior k states. Hence, for $k = 1$, $T_{i,j} = P(S_t^j | S_{t-1}^i)$. Figure 1 depicts a finite state Markov process. Markov processes are a well-studied area, and numerous techniques exist for analyzing their dynamic properties and for developing control policies (e.g., [24]).

Of course, when $|S|$ is large, it is not practical to explicitly represent S or T. A DBN solves this problem by encoding S, T, and π in a factored form, taking advantage of the conditional independencies between random variables. Figure 2 shows a DBN of the finite-state Markov process in Figure 1. A DBN is a pair, (B_S,B_T), where B_S is a static Bayesian network encoding the prior probability distribution, π, over S (Fig. 2-A), and B_T is a two-slice temporal Bayesian network encoding the state transition matrix, T (Fig. 2-B). A Bayesian network is a probabilistic graphical model comprising a set of nodes and edges. Nodes represent random variables and edges denote conditional dependencies among variables. Associated with each node is a conditional probability distribution (CPD) that encodes the probability of the state of that variable, given its parents in the graph. The precise form of the CPD depends on the nature of the model. When the state space is finite, CPDs are generally in tabular form. Notice that B_T (Fig. 2-B) is a more compact representation than the state transition matrix, T, in Figure 1-C in that the combined size of the two CPDs (12 table elements) is smaller than the size of T (16 matrix elements). In this example, the savings are modest, but in a larger system, with many more state-variables, the savings can be dramatic.

[1]The restriction to finite-state models is for illustration purposes only. It is not an inherent limitation of DBNs.

The efficiency of DBN learning and inference algorithms is proportional to the size of the CDPs.

The transition model for a DBN is the product of the CPDs.

$$P(Z_t|Z_{t-1}) = \prod_{i=1}^{m} P(Z_t^i|Pa(Z_t^i)) \tag{1}$$

The joint distribution for a sequence of length τ can be computed,conceptually, by "unrolling" B_T until it has τ slices. The joint distribution of the model is then:

$$P(S_{1..\tau}) = \prod_{i=1}^{m} P_{B_S}(Z_1^i|Pa(Z_1^i)) \prod_{t=2}^{\tau} \prod_{j=1}^{m} P_{B_T}(Z_t^j|Pa(Z_t^j)) \tag{2}$$

It is often the case that a particular observation sequence consists only of a subset of the random variables, Z. These variables are usually called the *observed* variables, and the complementary set are called the *hidden* (or latent) variables. By convention we refer to the observed variables by Y and the hidden variables by X. The joint distribution $P(X, Y)$ for a DBN with latent variables is computed in a manner similar to Eq. 2.

The DBN in Figure 2 model might arise in the context of gene regulation, where the individual variables/nodes represent genes and the edge-set encodes the regulatory relations amongst the genes. Given a DBN, there are a variety of tasks we may wish to perform. We use the terminology of Murphy [36] and define:

- *filtering*: computing $P(X_\tau|y_{1..\tau})$;
- *prediction*: computing $P(X_{\tau+h}|y_{1..\tau})$, for $h > 0$;
- *decoding*: computing $P(x_{1..\tau}|y_{1..\tau})$;
- *classification*: computing $P(y_{1..\tau})$.

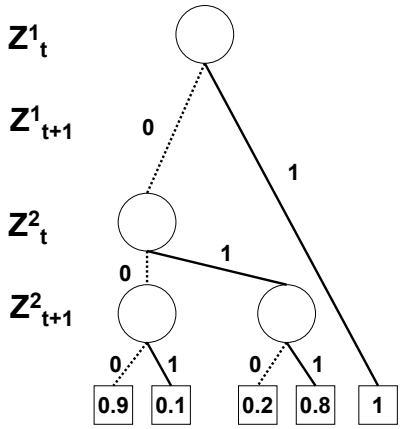
Inference algorithms for DBN are either exact or approximate. Exact inference is known to be #P-hard in general [23], but there are exact algorithms for special classes of models. For example, exact algorithms for decoding and classification in Hidden Markov Models run in time $O(T|S|^2)$. Of course, when $|S|$ is large, these algorithms are not practical, and approximate techniques are needed. Deterministic approximate inference algorithms exist, but the tightness of the bounds for these algorithms has not been established [36]. Inference techniques based on Monte Carlo sampling can be used, but these are expensive and are not well suited to answering questions about rare events (or rare sequences of events) that may be of biological significance.

2.2 Model Checking

The term *model checking* [21] refers to a family of techniques from the formal methods community for verifying systems of concurrent reactive processes. Model checking algorithms were originally developed to verify the correctness of circuit designs and communications protocols, both of which can be encoded as finite-state models. Over the past 25 years, however, new model checking

Transition	Z^1_t Z^2_t	Z^1_{t+1} Z^2_{t+1}	Z^1_t Z^1_{t+1} Z^2_t Z^2_{t+1}	P(s → s')
(0,0) → (0,0)	0 0	0 0	0 0 0 0	0.9
(0,0) → (0,1)	0 0	0 1	0 0 0 1	0.1
(0,1) → (0,0)	0 1	0 0	0 0 1 0	0.2
(0,1) → (0,1)	0 1	0 1	0 0 1 1	0.8
(1,0) → (0,1)	1 0	0 1	1 0 0 1	1
(1,1) → (0,0)	1 1	0 0	1 0 1 0	1

(A)



(B)

Figure 3: **(A)** A binary encoding of the same transition model depicted in Figures 1-C and 2-B. Columns 2 and 3 contain binary encodings of states 1-4 at times $t = i$ and $t = i + 1$, respectively, using the variables Z^1_t, Z^2_t, Z^1_{t+1} and Z^2_{t+1}. Column 4 shows one possible ordering of Z^1_t, Z^2_t, Z^1_{t+1} and Z^2_{t+1}. **(B)** The MTBDD encoding of Columns 4 and 5 of the table on the left. OBDD/MTBDD encodings do not represent internal nodes if they aren't necessary. For example, there is no need to explicitly represent Z^1_{t+1} since it is always 0. Notice that, in this example, the MTBDD encoding is more compact than the DBN encoding in Figure 2-B

algorithms have been devised for both stochastic (e.g., [4]) and hybrid models (models containing mixtures of discrete and continuous variables) (e.g.,[19, 43, 39, 38]). As previously mentioned, model checking algorithms scale to very large systems.

Model checking has recently been used to study biological systems (e.g., [14, 3, 15, 13, 5, 43, 39, 38, 29, 34]). However, the relationship between model checking to DBNs has not been previously reported. Like DBNs, model checking is a very broad area, so we will highlight a few key aspects of model checking that are relevant to this paper, focusing on aspects of model checking stochastic system.

2.2.1 Representation:

Like DBNs, probabilistic model checking does *not* use an explicit construction of the entire state space, S, or the state transition matrix, T. Rather, the state space is encoded in a factored form using a collection of data structures known as *multi-terminal binary decision diagrams* (MTBDD) [20]. A MTBDD is a directed acyclic graph for representing boolean functions of the form $f : \{0, 1\}^n \mapsto \mathbb{R}$ (Fig. 3). MTBDDs can be used to encode arbitrary vectors and matrices and are known to require no more space than a sparse encoding of the full matrix/vector. That is, a MTBDD encoding is no worse than an explicit encoding. However, MTBDDs can sometimes require less space than a sparse-matrix representation, depending on the exact nature of the matrix. The MTBDD in Figure 3-B, for example, is a more compact representation than *both* the sparse-matrix representation of the transition matrix in Figure 1-C *and* the DBN in Figure 2-B. The size of

the MTBDD depends on the number of unique elements in the transition matrix *and* the encoding of the state transitions (e.g., Column 4 in Figure 3-A). Finding an optimal encoding is NP-hard, but very good heuristics exist and the space savings can be substantial [8].

One of the advantages of MTBDD encodings is that arithmetic computations can be performed directly on MTBDDs in time proportional to the product of their sizes. That is, if f and g are MTBDDs encoding two matrices, then the time to compute $f \circ g$ is $O(|f||g|)$ where $\circ = *, /, +, -$ and $|f|$ and $|g|$ are the number of nodes in the MTBDD representations of f and g, respectively. This complexity result follows from a theorem due to Bryant for ordered binary decision diagrams (OBDD) [8]. OBDDs are canonical encodings of boolean functions of the form $f : \{0,1\}^n \mapsto (0,1)$. MTBDDs are a generalization of OBDDs to real-valued boolean functions.

2.2.2 Queries:

In model checking, queries are expressed as formulas in one of several *temporal logics*. The most common temporal logics are those based on computation trees. The basic idea is that, conceptually, the space of all possible execution traces from a given starting state (or set of starting states) can be modeled as an infinite computation tree. A query, therefore, corresponds to a question about a particular path, or sets of possible paths. These queries are encoded as a formula in a temporal logic.

To draw a comparison with DBNs, recall that DBNs inference algorithms are instance based. That is, they require a finite-length observation sequence. If all state variables are known (that is, there are no hidden variables), then that observation sequence corresponds to a single path in the computation tree. Otherwise, the observation sequence corresponds to a set of fixed-length paths, and the likelihood of observing that sequence can be computed by integrating over those paths. It is worth noting that there have been a variety of powerful model checking techniques that have been developed for reasoning about finite-length execution sequences [7]. These so-called bounded model checking techniques are also potentially useful as an alternative for inference algorithms on DBNs.

Temporal logic formulas are also capable of expressing detailed questions about the logical ordering of specific events, and can do so over both finite and infinite executions. The syntax and semantics of temporal logics based on computation trees, known as computation tree logics (CTL), vary, but they generally include the *path quantifiers* $\mathbf{A}\phi$, and $\mathbf{E}\phi$. Here, ϕ corresponds to a path formula that encodes the attributes of interest, and **A** and **E** correspond to the notions "for all paths" and "there exists a path" in the computation tree, respectively. CTL also includes *temporal operators*, $\mathbf{X}\phi$, $\mathbf{F}\phi$, $\mathbf{G}\phi$, and $\phi_1\mathbf{U}\phi_2$. Here, $\mathbf{X}\phi$ means that ϕ holds in the next state; $\mathbf{F}\phi$ means that ϕ holds sometime in the future; $\mathbf{G}\phi$ means that ϕ holds globally in the future; and $\phi_1\mathbf{U}\phi_2$ means that ϕ_1 holds until ϕ_2 holds. There is also the notion of a bounded until operator, $\phi_1\mathbf{U}^{\leq k}\phi_2$, which means that ϕ_1 holds for up to k steps, and then ϕ_2 holds. These operators can be combined using logical connectives and modifiers. Temporal logic formulas can encode a remarkable set of complex abstract behaviors. For example, Antoniotti *et. al.* [3] define a formula that can be used to ask whether a particular quantity, say x, oscillates between two thresholds, v_1 and v_2. The corresponding temporal logic formula is:

$$\mathbf{G}(\mathbf{F}(x < v_1) \wedge [x < v_1 \implies \mathbf{F}(x > v_2)] \wedge [x > v_2 \implies \mathbf{F}(x < v_1)]) \quad (3)$$

In English this formula say that it is globally true that i) x will eventually fall below v_1, ii) when x falls below v_1 it will, eventually, rise above v_2, iii) when x rises above v_2 it will, eventually, fall below v_1. DBN inference algorithms cannot encode such complex queries.

Many variants of CTLs exist. The most important of these with regard to this paper is probabilistic computation tree logic (PCTL) [28] which adds probabilistic operators, such as $P_{=?}\phi$. This operator asks "***with what probability will*** ϕ ***hold?***" Extensions to PCTL add operators for computing upper and lower bounds (e.g., $P_{<c}\phi$), expected time, and steady-state probabilities.

The model checking community has developed techniques for converting formulas in temporal logics into symbolic forms (i.e, OBDDs and MTBDDs). Queries are then answered, exactly, using symbolic computations. In particular, the model checking community has algorithms for performing fix-point computations symbolically. This avoids the need to explicitly enumerate all paths. Additionally, approximate model checking algorithms are also available (e.g, [47]).

Different temporal logics have different expressive powers. Consequently, the complexity of model checking will vary, depending on which logic is used. For example, model checking CTL formulae can be done in polynomial time in both the size of the model and the length of the temporal logic formula [18], whereas model checking formulas in the temporal logic CTL*, which combines the operators of CTL with those of linear temporal logic (LTL), is PSPACE-hard [44]. However, this result is tempered somewhat by the fact that model checking CLT* formula is linear in the size of the model, but exponential in the size of the formula [35]. That is, for short formulas, model checking CTL* formulas may be practical. We note that, as evidenced by the multitude of real-world examples of model checking using a number of different logics (see, e.g., [21]), effective strategies have been developed to address these worst-case complexity challenges.

3 Inference in DBNs via Model Checking

We have shown that DBNs and MTBDDs can be used to encode equivalent models. In this section we outline a procedure for converting DBNs into probabilistic reactive modules and a procedure for posing and answering inference problems using model checking.

The conversion of an existing DBN into a probabilistic reactive module is a one-time operation that is easily automated. Briefly, model checking tools generally provide a high-level language for defining a system of synchronous or synchronous processes. The syntax and semantics of these system specification languages vary. Our experiments, for example, were conducing using the PRISM probabilistic model checking software [31]. PRISM provides two options for specifying systems: a probabilistic extension to the reactive module specification formalism of [2], and the stochastic process algebra PEPA [30]; we specified our models using the PRISM modeling language. A system specification is then compiled into a MTBDD (or equivalent) where optimizations are applied to minimize the size of the MTBDD.

A DBN can be converted into a form suitable for model checking by creating a separate process for each random variable in the model (Figure 4). Each process' specification will encode the CPD

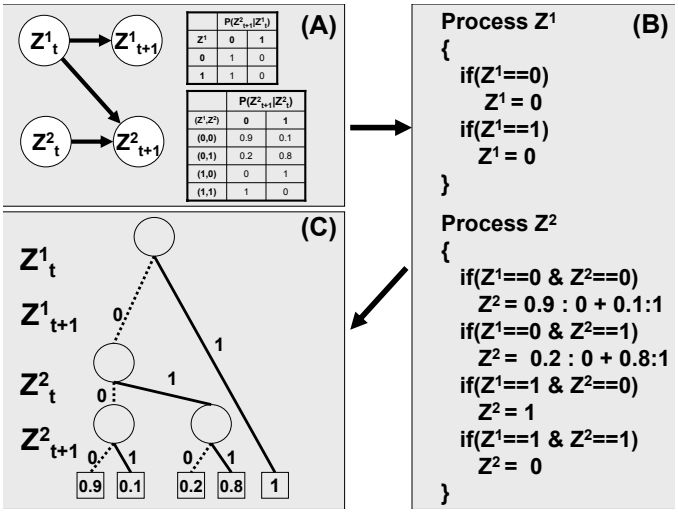


Figure 4: The conversion of B_T (panel **A**, copied from Fig. 2-B) into a probabilistic reactive module. Each variable is modeled as a process (panel **B**). Here, we use pseudo code based on the system specification language used by [31]. Basically, the specification encodes a CPD. This conversion is easily automated. Model checking software then convert the system specification into a MTBDD (panel **C**).

for that variable. DBNs encode discrete-time Markov chains. That is, each variable is updated at the same time. This implies that the processes in the system specification must be synchronous. We note, however, that the model specification languages can *also* be used to model asynchronous processes. Indeed, model checking is most often applied in other domains to asynchronous, concurrent reactive processes. This flexibility actually permits model checking of continuous-time Markov chains and Markov Decision processes (e.g., [40, 33]).

In addition to converting the DBN into a MTBDD, it is also necessary to support standard inference mechanisms. As previously mentioned, DBNs perform instance-based inference where a single, finite-length observation sequence, O, is specified. Generally, these observation sequences specify an exact temporal ordering of the states of the observed variables. Let λ be a DBN expressed as a MTBDD and let $O = (o_1, o_2, ..., o_t)$ be a sequence of observations. We can compute quantities like $P(O_{1..\tau}|\lambda)$ using model checking by composing a PCTL formula of the form:

$$P_{=?}[o_1 \wedge \mathbf{X}(o_2 \wedge \mathbf{X}(o_3 \wedge ... \wedge \mathbf{X}(o_\tau)...))] \tag{4}$$

Here, o_i is a Boolean predicate that indicates the values of the observed variables in the ith state. For example, if Y_a, Y_b, and Y_b are Boolean random variables, the Boolean predicate might be $o_i \models (Y_a = 1 \vee Y_b = 0 \vee Y_c = 0)$, where $a \models b$ means that a models or satisfies b. Or, if the variables are real-valued, the predicate may have the form $o_i \models (Y_a = 0.8 \vee Y_b = 99.2 \vee Y_c = 2.1)$. In English, Eq. 4 asks "With what probability will we, starting in state o_1, immediately move to state o_2, and then move to state o_3 and so on?" Probabilistic model checking algorithms are then used to evaluate the formula. Thus, it is possible to perform classification ($P(O_{1..\tau}|\lambda)$) via model checking. It is also possible to decode (i.e., compute $P(x_{1..\tau}|y_{1..\tau})$) by taking advantage of the counter-example generation capabilities of model checking. Here we would assert that a formula

of the following is *not* satisfied:

$$P_{<=0.8}[o_1 \wedge \mathbf{X}(o_2 \wedge \mathbf{X}(o_3 \wedge ... \wedge \mathbf{X}(o_\tau)...))]. \quad (5)$$

This formula asserts that all sequences of length τ that match the observation sequence have probability less than or equal to 0.8. If this property is not true, then the model checking algorithm provides a counter-example which will reveal the state transitions for all variables, not just the observed variables. In this way, decoding can be performed.

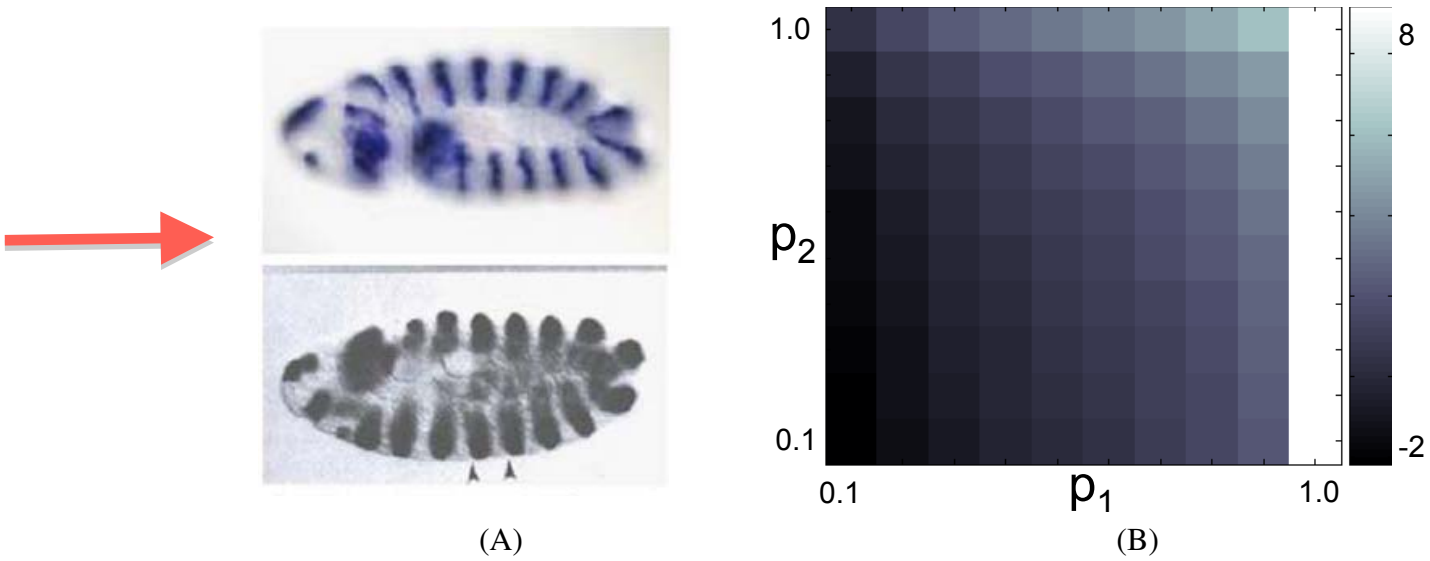
As previously mentioned, model checking also supports queries that are not expressible using traditional DBN algorithms. For example, Boolean predicates can be written to define equivalence classes of states such as $o_i \models (Y_a >= 1.2 \vee Y_b < 0 \vee Y_c = 12)$ or $o_i \models (Y_a + Y_b > 7 \wedge Y_c = 12)$. These predicates correspond to sets of states. Additionally, formulas of the form $\phi_1 \mathbf{U}^{\leq k} \phi_2$ let one specify both logical orderings of events, and a bound on the spacing between events, without having to specify an exact distance between events, as is necessary in instance-based inference. Finally, the previously cited formula expressing oscillating behavior is an example of a formula over infinite sequences.

4 Application To *D. Melanogaster* Embryo Development

We applied our approach to inference in DBNs to an existing model of fruit fly embryo development [1]. Briefly, Albert and Othmer have developed a Boolean network model of the segment polarity gene network in *D. Melanogaster* based on differential equation model of the same system developed by von Dassow and co-workers [46]. 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. We note that a Boolean network model can be encoded as a DBN where the transition probabilities are binary. That is, each element in the transition matrix, T, is either 0 or 1.

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 5-A. In the Albert and Othmer work, the parasegments are four cells wide.

In a follow-up study, Chaves, Albert, and Sontag [16] considered a somewhat different model wherein the synchrony of the updates was broken. That is, rather than updating every molecule in every cell at the same time, different molecules were allowed to update at different times. One of the primary findings of that work was that it is important for the proteins to be updated before the RNAs. When this property is violated, mutant patterns of expression are observed, such as the



(A) (B)

Figure 5: **(A)** Expression pattern of *wg* in wild-type (top) and a "broad-stripe" mutant embryo (bottom). Figures taken from http://www.fruitfly.org (top) and [45] (bottom). **(B)** Log-ratio of $\frac{P_{=?}(\mathbf{F}\ wild{-}type)}{P_{=?}(\mathbf{F}\ broad{-}stripe)}$ for different values of p_1 (x-axis), the update probability for the proteins, and p_2 (y-axis), the update probability for the RNAs.

"broad-stripe" pattern in (Figure 5-A, bottom). Our experiments further explore this property by computing the probability of the system converging onto either the wild-type expression pattern or the broad-stripe expression pattern under different scenarios using our model-checking based approach to inference. Additionally, we demonstrate the scalability of our approach by considering a two-dimensional array of cells, instead of the one-dimensional array of cells considered in [1] and [16]. We believe that this extension to the two-dimensional model is the first of its kind. There were a total of 192 Boolean variables in our model.

4.1 Experiments and Results

In Albert and Othmer model, the proteins are updated with probability 1 before the RNAs are allowed to update. In our experiment, we decided to further explore this property by exploring constructing a DBN where, at each step, the proteins update with probability p_1 and the RNAs update with probability p_2 for different values of p_1 and p_2 over the range [0.1,1.0]. Thus, when $p_1 = p_2 = 1$, the proteins and RNAs update synchronously, and the model is equivalent to the Boolean network model in [1]. When either p_1 or p_2 are less than 1.0, then the molecules either update according to the Boolean function, or they remain in the same state. We note that this defines a continuous-time Markov chain (CTMC) for each molecule type. We are thus simulating a pair of coupled CTMCs via a DBN in our experiments. Under this model, there are different possible interleaving of updates for the RNAs and proteins. The question we set out to answer is: *for what combinations of p_1 and p_2 does the model converge on the wild-type expression pattern with high probability*?

In our experiment we used model checking to compute the probabilities $P_{=?}(\mathbf{F}\ wild - type)$, and $P_{=?}(\mathbf{F}\ broad - stripe)$. That is, what is the probability that the system reaches either the wild-type or broad-stripe pattern from a given starting state. Our starting state was the same as that used in [1] and [16] —*wg* expressed in the posterior cell of the parasegment, *en* and *hh* expressed in the anterior cell of the parasegment, *ptc* and *ci* expressed in all cells except the anterior cell of the parasegment, and all other molecules are off. We considered a 4 by 4 array of cells. That is one parasegment wide and 4 cells high. In contrast, the corresponding experiment in [16] work considered a single parasegment (i.e., 4 cells). Like [1] and [16] we used periodic boundary conditions. In all, 100 different combinations of p_1 and p_2 were considered. Each experiment took between 18 seconds to 19 minutes on a single Pentium 3 processor.

Figure 5 shows the log-ratio of $\frac{P(\mathbf{F}\ wild-type)}{P(\mathbf{F}\ broad-stripe)}$. Our results are consistent with [16]; under the model, the system is more likely to converge on the wild-type pattern when the proteins are more likely to be updated before the RNAs. Further analysis shows that the likelihood of the broad-stripe pattern is greatest (89%) when $p_1 = p_2 = 0.1$. The wild-type pattern is most likely (100%) when $p_1 = p_2 = 1.0$, as expected. The probability of going to the wild-type or the broad-stripe pattern is roughly equal when $p_1 = 0.3$ and $p_2 = 0.5$.

5 Conclusions and Future Work

We have introduced a new method for performing inference in DBNs by first translating the DBN into a reactive module formalism, and then using existing probabilistic model checking algorithms to perform the inference. We believe that the primary advantage of a model- checking based approach lies in the richer set of inference problems that can be expressed using temporal logics. We demonstrated the practical use of this method on a model of *Drosophila* embryo development over a two dimensional array of cells. Previous experiments had considered only the one dimensional case. Our model had a total of 192 Boolean variables and runtimes range from less than 20 seconds to less than 20 minutes on the model.

There are many areas for future work. As previously mentioned, model checking techniques exist for continuous-time Markov chains and Markov decision processes. Thus, model checking is applicable to a larger variety of models than can be expressed using DBNs. Additionally, model checking algorithms have often been used in other domains to develop control policies. We are presently extending our method for the design of control strategies for biological systems. Such techniques may have application in fields such as synthetic biology, where the goal is to design biological system that have a pre-defined behavior. Finally, we note that model checking algorithms exist for hybrid systems — models containing mixtures of discrete and continuous variables. Our experiments were limited to finite-state DBNs, but we are interested in developing similar techniques for hybrid models.

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] 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, 2003.

[2] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design: An International Journal*, 15(1):7–48, 1999.

[3] 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.

[4] C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP'97)*, volume 1256 of *LNCS*, pages 430–440. Springer, 1997.

[5] G. Batt, D. Ropers, H. de Jong, J. Geiselmann, R. Mateescu, M. Page, and D. Schneider. Validation of qualitative models of genetic regulatory networks by model checking: analysis of the nutritional stress response in Escherichia coli. *Bioinformatics*, 25(1):i19–i28, 2005.

[6] M. J. Beal, F. Falciani, Z. Ghahramani, C. Rangel, and D. L. Wild. A bayesian approach to reconstructing genetic regulatory networks with hidden factors. *Bioinformatics*, 21(3):349–356, 2005.

[7] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 317–320, New York, NY, USA, 1999. ACM Press.

[8] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.

[9] J.R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. *Proc. 1991 Conf. on VLSI*, pages 49–58, 1991.

[10] 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.

[11] 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.

[12] C. Bystroff, V. Thorsson, and D. Baker. Hmmstr: a hidden markov model for local sequence-structure correlations in proteins. *J Mol Biol.*, 301(1):173–190, 2000.

[13] M. Calder, V. Vyshemirsky, D. Gilbert, and R. Orton. Analysis of signalling pathways using the PRISM model checker. In *Proc. Computational Methods in Systems Biology (CMSB'05)*, 2005. To appear.

[14] 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.

[15] N. Chabrier-Rivier, M. Chiaverini, V. Danos, F. Fages, and V. Schächter. Modeling and querying biomolecular interaction networks. *Theor. Comput. Sci.*, 325(1):25–44, 2004.

[16] M. Chaves, R. Albert, and E.D. Sontag. Robustness and fragility of boolean models for genetic regulatory networks. *Journal of Theoretical Biology*, 235:431, 2005.

[17] W. Chu, Z. Ghahramani, and D. L. Wild. A graphical model for protein secondary structure prediction. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 21, 2004.

[18] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1981. Springer-Verlag.

[19] E. M. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *International Journal of Foundations of Computer Science*, 14(4):583–604, 2003.

[20] 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.

[21] E.M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.

[22] G. Clermont, J. Bartels, R. Kumar, G. Constantine, Y. Vodovotz, and C. Chow. In silico design of clinical trials: A method coming of age. *Crit. Care Med.*, 32(10):2061–2070, 2004.

[23] P. Dagum and R. M. Chavez. Approximating probabilistic inference in bayesian belief networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(3):246–255, 1993.

[24] M. H. A. Davis. *Markov Models and Optimization*. Chapman & Hall, New York, 1993.

[25] 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. In S. Graf and M. Schwartzbach, editors, *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 395–410. Springer, 2000.

[26] A.L. Delcher, S. Kasif, H. R. Goldberg, and W. H. Hsu. Protein secondary structure modelling with probabilistic networks. *Proc Int Conf Intell Syst Mol Biol.*, 1:109–117, 1993.

[27] C. M. Ghim, K. I. Goh, and B. Kahng. Lethality and Synthetic Lethality in Genome-Wide Metabolic Network of *Escherichia coli*. *J. Theor. Biol.*, 237:401–411, 2005.

[28] H. Hansson and B. Honsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6:512–535, 1994.

[29] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. In *Proc. Computational Methods in Systems Biology (CMSB'06)*, 2006. To appear.

[30] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[31] A Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[32] M.I. Jordan. An Introduction to Probabilistic Graphical Models. Unpublished Book, in preparation.

[33] M. Kwiatkowska, G Norman, and D. Parker. Game-based abstraction for Markov decision processes. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 157–166. IEEE CS Press, 2006.

[34] M. Kwiatkowska, G. Norman, D. Parker, O. Tymchyshyn, J. Heath, and E. Gaffney. Simulation and verification for computational modelling of signalling pathways. *Proc. 2006 Winter Simulation Conference*, page in press, 2006.

[35] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL '85: Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 97–107, New York, NY, USA, 1985. ACM Press.

[36] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley; Computer Science Division, 2002.

[37] K. Murphy and S. Mian. Modelling gene expression data using dynamic bayesian networks, 1999.

[38] V. Mysore and B. Mishra. Algorithmic Algebraic Model Checking III: Approximate Methods. *7th International Workshop on Verification of Infinite-State Systems (INFINITY)*, pages 61–77, 2006.

[39] V. Mysore, C. Piazza, and B. Mishra. Algorithmic Algebraic Model Checking II: Decidability of Semi-Algebraic Model Checking and its Applications to Systems Biology. *Automated Technology for Verification and Analysis (ATVA)*, pages 217–233, 2005.

[40] G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta. Formal analysis and validation of continuous time Markov chain based system level power management strategies. In W. Rosenstiel, editor, *Proc. 7th Annual IEEE International Workshop on High Level Design Validation and Test (HLDVT'02)*, pages 45–50. IEEE Computer Society Press, 2002.

[41] V. Paetkau, R. Roderick Edwards, and R. Illner. A model for generating circadian rhythm by coupling ultradian oscillators. *Theor Biol Med Model.*, 3(1):495–505, 2006.

[42] B.E. Perrin, L. Ralaivola, A. Mazurie, S. Bottani, J. Mallet, and F. dAlchBuc. Gene networks inference using dynamic Bayesian networks. *Bioinformatics*, 11(2):ii138–ii148, 2002.

[43] 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.

[44] .A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[45] 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.

[46] G. von Dassow, E. Meir, E. M. Munro, and G. M. Odell. The segment polarity network is a robust developmental module. *Nature*, 406(6792):188–192, 2000.

[47] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, pages 223–235, London, UK, 2002. Springer-Verlag.